Memoria

Diseño de Sistema modular basando en microcontroladores para el control de maquetas ferroviarias

Alan Moreno Criado

Julio 2021

Director: Enric X. Martin Rull
Departamento: ESAII

Facultad de informática de Barcelona - UPC

Grado en ingeniería informática

Especialización en Ingeniería de Computadores

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

// Agradecimientos

Quisiera dedicar este trabajo a toda la gente que forma parte de mi vida: mi madre y mi padre, familia, amigos, compañeros de carrera… Todos han aportado algo a mi vida y sin ellos no sería igual. Pero lo más importante es que todos me han animado, cada uno a su manera, a continuar y no rendirme.

Agradecer también a mi director Enric X. Martin Rull por guiarme y ayudarme en las consultas y dudas que han ido surgiendo a lo largo de la realización del proyecto, y a Dídac Rodríguez Cejudo ya que fue quien me dio la idea, una tarde charlando mientras estudiábamos MP (*Multiprocesadores*), bromeando acerca de cómo me conozco los modelos de cercanías que *RENFE* utiliza. Quisiera volver a mencionar de nuevo a mis padres por sus infinitos ánimos, consejos y paciencia, sin ellos no habría sido posible realizar este trabajo.

Y por supuesto también a la FIB y a sus profesores, ya que ha sido gracias a los conocimientos que me han dado que he sido capaz de realizar este proyecto.

Resumen

Este trabajo tiene como objetivo el diseño y la implementación de un sistema modular para controlar maquetas ferroviarias, basado en microcontroladores PIC. Las siglas PIC significan *Programmable Integrated Circuited*, y se trata de una familia de microcontroladores basados en tecnología RISC. Fabricados por *Microchip Technology Inc*, contienen los componentes necesarios para poder realizar y controlar tareas como las realizadas en este proyecto.

El mundo del modelismo ferroviario es extenso, y puede abarcar tamaños muy diversos: desde la maqueta con un circuito muy simple en forma de cero, a las maquetas que intentan imitar el trazado real a escala de la red ferroviaria de una ciudad. Un sistema para poder controlar las diferentes partes que componen la maqueta que fuese accesible desde el móvil sería ideal para el usuario medio. En mi caso, partiré de una maqueta con forma de cero con una vía alternativa a la que se accede por un desvío; y el diseño del sistema constará de una placa Arduino que se encargará de gestionar el sistema (actuará como *master*), y varios PICs que realizarán las tareas que la Arduino les indique (actuarán como *slave*). El objetivo principal es lograr controlar estos elementos desde un móvil a través de una aplicación que se comunicará con este sistema.

Resum

Aquest treball te com objectiu el disseny i la implementació d'un sistema modular per controlar maquetes ferroviàries, basat en microcontroladors PIC. Les sigles PIC signifiquen *Programmable Integrated Circuited*, i es tracta d'una família de microcontroladors basats en tecnologia *RISC*. Fabricats per *Microchip Technology Inc*, contenen els components necessaris per poder realitzar i controlar tasques com les realitzades en aquest projecte.

El món del modelisme ferroviari es extens, i pot comprendre mides molt diverses: des de la maqueta d'un circuit molt simple en forma de zero, a les maquetes que intenten imitar el traç real a escala de la xarxa ferroviària d'una ciutat. Un sistema per poder controlar les diferents part que composen la maqueta que fos accessible des de el mòbil seria ideal per l'usuari mitjà. En el meu cas, partiré des de una maqueta en forma de zero amb una via alternativa a la que s'accedeix per un desviament; i el disseny constarà d'una placa Arduino que s'encarregarà de gestionar el sistema (actuarà com a *master*), i de diversos PICs que realitzaran les tasques que la Arduino els hi indiqui (actuaran com a *slave*). L'objectiu principal es aconseguir controlar aquests elements des de un mòbil a través d'una aplicació que es comunicarà amb el sistema.

Abstract

This project aims to design and implement a modular system for controlling railway models, based on PIC microcontrollers. The acronym PIC stands for *Programmable Integrated Circuited*, and it is a family of microcontrollers based on *RISC* technology. Manufactured by *Microchip Technology Inc*, they contain the components needed to perform and control tasks such as those performed in this project.

The world of railway modeling is vast, and can include many different sizes: from the model of a very simple circuit that has a zero form, to the models that try to imitate a real city's railway network in scale. A system to control the different parts that compose the model which is accessible from the mobile phone would be ideal for the average user. In my case, I will start from a model that has a zero form with an alternative path to which is accessed by a detour; and the design will consist of an Arduino board that will manage the system (it will act as a *master*), and of several PICs that will carry out the tasks that the Arduino indicates to them (they will act as a *slave*). The main goal is to be able to control these elements from a mobile phone through an *APP* that will communicate with the system.

Contenido

Resumen IV

Lista de Figuras IX

Lista de Tablas X

1. Introducción 1

1.1 Contexto 1
1.1.2 Motivación 1
1.1.3 Problema a resolver 2
1.1.4 Partes interesadas 2
1.1.5 Justificación 3
1.2 Términos y conceptos 3
1.3 Alcance del proyecto 5
1.3.1 Objetivos 5
1.3.2 Requisitos 7
1.3.3 Obstáculos y riesgos 7
1.4 Metodología y rigor 8
1.5 Descripción de las tareas 9
1.5.1 Gestión del proyecto 9
1.5.2 Desarrollo del proyecto 9
1.5.2.1 Preparación de la maqueta 9
1.5.2.2 Módulo de control de velocidad 10
1.5.2.3 Módulo de control de cambios de vía 11
1.5.2.4 Módulo de control de semáforos 11
1.5.2.5 Módulo de detección de posición 12
1.5.2.6 Programa principal 13
1.5.2.7 Aplicación móvil 13
1.5.3 Documentación del proyecto 13
1.6 Estimación y diagrama de Gantt 14
1.7 Documentación del proyecto 16

2. Microcontroladores PIC 17

2.1 Características de los PIC 17
2.2 Programación del PIC 20
2.3 Set de instrucciones 21
2.4 MSSP - Master Synchronous Serial Port 23
2.4.1 I2C 24
2.4.2 I2C - slave recieve mode 26
2.4.3 I2C - slave transmit mode 28
2.5 Test de los PICs 30
2.5.1 I2C 30

3. **Diseño del sistema y los módulos** 36

3.1 MCV - Módulo de control de velocidad 38
3.2 MCCV - Módulo de control de cambio de vía 40
3.3 MCS - Módulo de control de semáforos 42
3.4 MDP - Módulo de detección de posición 43
3.5 Sistema con todos los módulos juntos 45

4. **Diseño de la aplicación móvil** 50

5. **Evaluación de la funcionalidad del proyecto** 56

5.1 Montaje de la maqueta 57
5.2 Conexiones 60
5.3 Test 61
5.4 Análisis de los resultados 65

6 **Conclusiones** 66

Apéndices 68

A **Gestión de riesgos: planes alternativos y obstáculos** 69

B **Presupuesto** 69

B.1 Identificación de costes 69
B.2 Estimación de costes 72
B.3 Control de gestión 73

C **Sostenibilidad** 74

C.1 Dimensión ambiental 74
C.2 Dimensión económica 75
C.3 Dimensión social 76

Glosario 78

Bibliografía 80

Lista de Figuras

Figura 1.1: Central digital de control 2
Figura 1.2: Diseño modular de un tranvía 4
Figura 1.3: Comunicación serie (conceptual) 4
Figura 1.4: UPC – Campus Nord 8
Figura 1.5: Diagrama de Gantt 15
Figura 2.1: Microcontroladores PIC de Microchip 17
Figura 2.2: Características principales de los PIC de 12, 14 y 16 bits de Microchip 17
Figura 2.3: Pinout PIC16F15313 18
Figura 2.4: Pinout PIC16F15323 18
Figura 2.5: Diagrama de bloques PIC16F15313 19
Figura 2.6: Diagrama de bloques PIC16F15323 19
Figura 2.7: Disposición de pines del programador PICKIT 3 21
Figura 2.8: Diagrama de bloques del *MSSP* (I2C slave) 24
Figura 2.9: Conexión entre *master* y *slave* 25
Figura 2.10: Condiciones de start y stop en I2C 25
Figura 2.11: I2C slave recieve mode con direccionamiento de 7 bits 27
Figura 2.12: I2C slave transmit mode con direccionamiento de 7 bits 29
Figura 2.13: Árbol de ficheros y características del proyecto en MPLAB X IDE 31
Figura 2.14: Parte del main de un proyecto en MPLAB X IDE 32
Figura 2.15: Función *SYSTEM_Initialize* y ejemplo de función de configuración 33
Figura 2.16: Estructura de los bits que componen el registro *ANSELA* 33
Figura 2.17: Gestor de interrupciones general 34
Figura 2.18: Código para probar si el PIC se comunica por I2C 35
Figura 2.19: Test del código anterior: read1Byte con valor de 0b00100000 (RA5 = 1) 35
Figura 3.1: Arduino Uno Rev. 3 36
Figura 3.2: Boceto de la idea conceptual del sistema 37
Figura 3.3: Esquemático del circuito para el control de velocidad 38
Figura 3.4: Esquemático interno del transistor *P24NF10* 39
Figura 3.5: Ejemplo de uso del *PWM* 39
Figura 3.6: Circuito con un puente rectificador de onda completa 40
Figura 3.7: Semiciclo positivo 40
Figura 3.8: Semiciclo negativo 40
Figura 3.9: Onda de salida resultante rectificada 41
Figura 3.10: Rectificador de onda completa con condensador de suavizado 41
Figura 3.11: Esquemático del circuito para el control de cambio de vía 42
Figura 3.12: Esquemático del circuito para el control de semáforos 42
Figura 3.13: Semáforo utilizado en la maqueta, controlado por el módulo 42
Figura 3.14: Esquemático del circuito para la detección de presencia 43
Figura 3.15: Test del sensor de presencia con la resistencia de *pull-down* 44
Figura 3.16: Esquemático definitivo del sistema con los módulos integrados 46
Figura 3.17: Módulo de control de velocidad 47
Figura 3.18: Módulo de control de cambio de vía 48
Figura 3.19: Módulo de control del semáforo 48
Figura 3.20.a: Módulo de detección de posición (vista superior) 49
Figura 3.20.b: Módulo de detección de posición (vista inferior) 49
Figura 4.1: Interfaz de usuario de la aplicación en el Diseñador 50
Figura 4.2: Test de la aplicación para controlar el semáforo 1 y ponerlo en rojo (STOP) 51
Figura 4.3: Lógica del interruptor de cambio de vía 52
Figura 4.4: Función *formatData* 52
Figura 4.5: Rutina de la interrupción del *timer* 53
Figura 4.6: Código para enviar un byte de datos a la aplicación desde la Arduino 54

Figura 4.7: Código para recibir seis bytes de datos de la aplicación a la Arduino 54
Figura 4.8: Código para enviar la información tratada hacia los módulos y la aplicación 55
Figura 4.9: Módulo de bluetooth añadido al sistema 55
Figura 5.1: Estructura del cableado del sistema en la base 56
Figura 5.2: Ejemplo de sensor encajado con éxito entre dos traviesas 57
Figura 5.3: Proceso de construcción de la base 57
Figura 5.4: Proceso de construcción de la base: módulo de control de posición 58
Figura 5.5: Proceso de construcción de la base: módulo de control de semáforo 59
Figura 5.6: Proceso de construcción de la base: resto de módulos instalados 59
Figura 5.7.a: Conexiones en la protoboard 60
Figura 5.7.b: Esquema de las conexiones de la protoboard 60
Figura 5.8.a: Controlando el solenoide desde la aplicación (DISABLED) 61
Figura 5.8.b: Controlando el solenoide desde la aplicación (ENABLED) 61
Figura 5.9.a: Semáforo en verde (GO) 62
Figura 5.9.b: Semáforo en rojo (STOP) 62
Figura 5.10: Comprobando el funcionamiento del módulo detector de posición 63
Figura 5.11: Comprobando el funcionamiento del módulo de control de velocidad 64

Lista de Tablas

Tabla 1.1: Resumen de las tareas a realizar 14
Tabla 2.1: Set de instrucciones para el PIC16F15313/23 22-23
Tabla a.1: Presupuesto 71
Tabla a.2: Salario anual de los perfiles involucrados 72

1 Introducción

Ideado por mí persona y coordinado por mi director, espero poder aumentar mis conocimientos en las competencias tratadas en relación a los microcontroladores. En este apartado os introduciré en los esfuerzos que he dedicado para llevarlo a cabo por mi cuenta. Dicho esto, comencemos con el marco contextual de este proyecto y algunos conceptos básicos relacionados.

1.1 Contexto

El objetivo de este proyecto es el de realizar el diseño y la implementación de un sistema modular basado en microcontroladores para el control de maquetas ferroviarias, que además podrá ser controlado desde una aplicación móvil. Cada uno de los módulos tendrá que poder controlar un aspecto de la maqueta según se le ordene desde la aplicación, así como facilitarle información a esta para que pueda ser consultada por el usuario.

En este mundo del modelismo ferroviario existen módulos para las maquetas de gran envergadura que permiten controlar ciertos aspectos, pero tienen el inconveniente de que son complejos. Además de no ser una solución barata si estamos hablando de una maqueta más humilde.

Por lo que he decidido realizar este proyecto para conseguir un sistema que sea más humilde, simple, y que tenga la capacidad de poder ser ampliado si así se requiere. [1] Los módulos del sistema deberían ser capaces de controlar los principales elementos de una maqueta básica: la velocidad de las máquinas, los cambios de vía (se mecanizarán para poder ser controlados), los semáforos y la detección de la posición de la máquina.

Como se ha mencionado antes, he decidido crear también una pequeña aplicación móvil con la cual controlar los módulos y ver el estado de la maqueta. Desde ella el usuario debería de poder controlar la velocidad, ver la posición del tren en un pequeño mapa… Por lo que la aplicación ha de ser capaz de recibir información de la maqueta (posición de la máquina), y de enviarla (cambio de vía, o de velocidad).

1.1.2 Motivación

La motivación de este proyecto surge de mi agrado por las asignaturas cursadas a lo largo de la carrera, que han tratado sobre microcontroladores (como CI, DSBM y PEC), y que tienen como objetivo que el estudiante aprenda a programar y a hacer uso de estos. De la misma forma, también siento especial apego por el mundo ferroviario, por lo que decidí juntar ambos conceptos para ver qué soy capaz de desarrollar. De ahí surgió la idea de diseñar el sistema que presento en este proyecto.

La maqueta que se usará para el desarrollo de este proyecto es propia, y las modificaciones necesarias que necesitará serán realizadas por mi. Modificaciones tales como la instalación de sensores o la mecanización de los cambios de vía.

1.1.3 Problema a resolver

En el mundo de las maquetas ferroviarias, sobretodo el profesional, existe una pequeña variedad de dispositivos, muy potentes, llamados *centrales digitales* que sirven para operar estas maquetas. [6] Compañías conocidas como *Märklin* ofrecen estos dispositivos, capaces de dotar de un control extenso y avanzado sobre maquetas de modelismo ferroviario.

Figura 1.1: Central digital de control [6].

Estas centrales digitales permiten el control y circulación de varias máquinas a la vez, y poseen capacidad de decisión si prevén que dos máquinas podrían colisionar, haciendo que tomen desvíos. También son capaces de controlar semáforos y de asegurarse que las máquinas cumplen las señales de estos.

Sin embargo, estos dispositivos están orientados a un modelismo profesional, para aquellas personas que tienen una maqueta de gran envergadura y con varias máquinas corriendo a la vez. Y esto hace que su precio también sea de gran envergadura, por lo que no es una opción para un usuario que dispone de una maqueta más humilde, y con una sola maquina, por ejemplo.

Afortunadamente, podemos encontrar soluciones más baratas a día de hoy gracias a los microcontroladores. Con este proyecto partiremos de esa premisa. Gracias al auge de estos en los últimos años, y con los conocimientos adecuados, no hay motivos para creer que no es posible obtener un resultado similar.

1.1.4 Partes Interesadas

Los resultados de este esfuerzo pueden ser usados por las siguientes personas y entidades.

Comunidad del modelismo ferroviario

Es natural pensar que las primeras personas que puedan beneficiarse de este trabajo sea toda la comunidad por razones obvias, principalmente los que no se lo tomen como algo profesional. Cualquiera que tenga una maqueta ferroviaria y tenga interés en controlarla a distancia, puede hacer uso de este trabajo.

Educación universitaria

Como mencioné, una de las motivaciones de este proyecto surge de las asignaturas que, a lo largo de mi carrera, han tratado sobre el aprendizaje del uso de microcontroladores. Por lo que me parece apropiado pensar que los conocimientos plasmados en este documento puedan ser usados por instituciones académicas para aquellos alumnos que se estén familiarizando en este campo. O como base para otro proyecto futuro.

1.1.5 Justificación

El principal objetivo es lograr una alternativa que sea eficaz, barata, y que tenga la posibilidad de actualizarse en el futuro, si así se requiere. Partimos de la base de que los dispositivos que el mercado ofrece para el modelismo ferroviario no están orientados a un usuario que no se dedique a ello profesionalmente. Algunos de estos dispositivos pueden llegar a valer incluso más dinero que la propia maqueta de iniciación. Por lo que considero justificado el diseño y desarrollo de esta nueva solución.

Para alcanzar el objetivo será necesario adecuar la maqueta para dotarla de las características apropiadas, algo en lo que estoy trabajando mediante la adición de sensores, y la mecanización de los cambios de vía, por ejemplo. Unos cambios de vías ya equipados para ser controlados a distancia son notablemente más caros que unos que se accionan de forma convencional (palanca manual). [7] Sin embargo, podemos mecanizarlos haciendo uso de solenoides, una solución bastante más barata.

Gracias a esta serie de adiciones, nos será posible conocer el estado de las vías. Lo que nos permitirá también controlar, desde el sistema de microcontroladores, tanto los desvíos y los semáforos, como la máquina que circula sobre ella.

Para poder trabajar de forma segura y cómoda con la maqueta, será instalada sobre una base. No obstante, esta es una acción que tomo deliberadamente y sin mayor impacto en el proyecto, ya que estas maquetas acostumbran a instalarse siempre sobre una base. En mi caso, la tenía guardada, y por tanto no tenía una base. Esto tiene como ventaja que los sensores podrán ir fijados en la base, lo que facilitará la lectura de estos.

1.2 Términos y conceptos

A continuación, listaré los términos y conceptos que son necesarios conocer para entender mejor el objeto de estudio de este proyecto.

Microcontrolador

[2] Un microcontrolador es un circuito integrado digital que puede ser usado por diversos propósitos, ya que es programable. Se compone de una CPU, memorias RAM y ROM, y líneas de *input/output* para los periféricos. Dotado de los mismos bloques de funcionamiento básicos que una computadora, nos permite tratarlo como un pequeño ordenador.

Las aplicaciones de este dispositivo son enormes: desde avisos lumínicos y alarmas, hasta juegos e incluso robótica. En este caso, los usaremos para el manejo de sensores y controladores principalmente.

Debido a que el *hardware* ya viene integrado en un solo chip, su funcionamiento se especifica mediante un *software* que previamente ha de cargarse en memoria.

Sistema Modular

Cuando hablamos de un sistema modular, nos referimos al enfoque modular que se le da al diseño de un sistema. Esto es, cada una de las partes o módulos del sistema realiza una función o tarea concreta y no se ve afectado por el resto de módulos. Además, tiene la característica de ser escalable, es decir, puede ser ampliado en el futuro si así se desea. Y por último, si una de los módulos necesita ser reemplazado, no entorpece al resto, permitiendo así que el sistema siga funcionando con los módulos restantes.

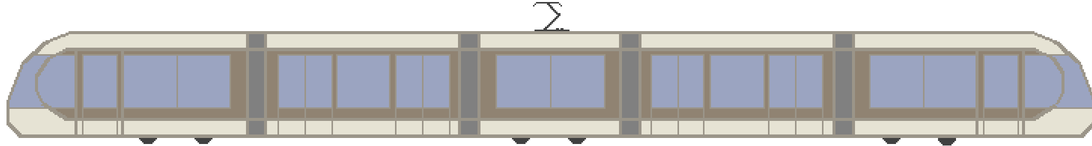


Figura 1.2: Diseño modular de un tranvía.

Arduino

[3] Las Arduino son placas SBC (de las siglas en ingles *Single-Board Computer*) para la construcción de dispositivos digitales e interactivos que puedan detectar y controlar objetos del mundo real. [4] Estos productos son distribuidos por la compañía de desarrollo Arduino, como *hardware* y *software* libre, bajo la Licencia Pública General de GNU, y la Licencia Pública General Reducida de GNU.

[3] El *hardware* de las Arduino acostumbra a consistir en un microcontrolador AVR (*RISC*) sobre una placa de circuito impreso. [3] Se puede expandir con más placas, llamadas *shields*, que añaden funcionalidades al agregar más circuitería, sensores, y módulos externos a la placa original.

El *software* de las Arduino está basado en C, y se desarrolla en un IDE específico para las placas Arduino. Además ofrece la posibilidad de cargarlo a la propia placa una vez compilado el código, usando comunicación serie.

Comunicación serie

Es el proceso de envío de datos de un bit a la vez, secuencialmente, sobre un canal de comunicación o bus. Se utiliza para conectar físicamente dispositivos asíncronos con un sistema. La comunicación serie es muy importante porque gran parte de los protocolos utilizados actualmente son serie. Además, muchos dispositivos de comunicación inalámbrica usan la comunicación serie para establecer conexión con un Arduino, como los módulos bluetooth, por ejemplo.

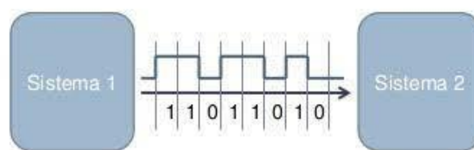


Figura 1.3: Comunicación serie (conceptual).

Bluetooth

[5] Es una tecnología de red desarrollada como estándar industrial para conexiones inalámbricas de área personal, y sirve para la transferencia de datos punto a punto entre dos dispositivos digitales diferentes. Al ser inalámbrica, supone una ventaja frente a las conexiones por cable, sobretodo en dispositivos móviles o para la comunicación de módulos que estén a corta distancia. Se caracteriza por establecer conexiones sencillas y de bajo consumo, pero en general con bajas velocidades de transferencia de datos.

Mecanización del cambio de vía

La maqueta que se usará tiene cambios de vía que permiten elegir entre dos circuitos diferentes. Este cambio es manual, y se activa accionando una palanca mecánicamente conectada a la vía. Cuando hablo de mecanizar, me refiero a añadir los ítems necesarios para poder controlar este cambio digitalmente y a distancia.

1.3 Alcance del Proyecto

En esta sección definiré los objetivos globales, así como sub-objetivos, los requisitos funcionales, y los posibles obstáculos y riesgos del proyecto.

1.3.1 Objetivos

Para considerar que el proyecto ha sido un éxito, he de conseguir lograr estos objetivos tanto como sea posible. Para poder considerar cumplidos cada uno de los siete objetivos principales, hay que completar varios sub-objetivos o pasos.

- **Adecuación de la maqueta** => Hay que adaptarla para que pueda acoger las nuevas adiciones y modificaciones. Para lograr este objetivo, es necesario completar estos pasos:
  - Preparación de una base para la maqueta: tomas las medidas y hacer las hendiduras necesarias.
  - Preguntarse a uno mismo si ya dispongo de todos los elementos que conforman la maqueta, y si ya es posible instalarla, sensores y el resto de elementos incluidos.
  - Instalación de la maqueta: una vez establecidas las nuevas modificaciones, ya podemos instalar la maqueta sobre la base.

  Estas decisiones en el primer objetivo se deben a que quiero intentar acomodar los sensores en la base, para que destaquen lo menos posible, pero para ello antes he de estar seguro de cual va a ser la cantidad, y de qué tamaño son. Es posible que algunos vayan debajo de la vía, por lo que no puedo instalar la vía hasta el final.

  Los siguientes 4 objetivos tienen pasos semejantes porque son los 4 módulos que conformarán el sistema, y el proceso a seguir en cada uno es similar. Esto no significa que se tengan que hacer de manera secuencial. [8] Para ser eficiente, lo ideal será pedir todos los materiales de todos los módulos a la vez.

- **Módulo de control de velocidad** => La velocidad del tren debe controlarse a distancia. Para lograr este objetivo, es necesario completar estos pasos:
    - Diseñar la implementación del módulo: de qué elementos se va a componer el módulo.
    - Generar un *bill of materials*: con el diseño establecido, hay que buscar las mejores opciones que el mercado puede ofrecer. A partir de ahora, nos referiremos a esto como BOM.
    - Comprar lo decidido en el BOM.
    - Ensamblaje del módulo.
    - Testeo eléctrico del módulo: conexiones correctas, sin cortocircuitos.

- **Módulo de control de cambios de vía** => El cambio de vía debe ser controlado a distancia. Para lograr este objetivo, es necesario completar estos pasos:
    - Mecanización del cambio de vía: idear e implementar una manera de llevarlo a cabo.
    - Diseñar la implementación del módulo: de qué elementos se va a componer el módulo.
    - Generar un BOM: con el diseño establecido, tanto de la implementación del módulo, como de la mecanización del cambio de vías, hay que buscar las mejores opciones que el mercado puede ofrecer.
    - Comprar lo decidido en el BOM.
    - Ensamblaje del módulo.
    - Testeo eléctrico del módulo y del cambio de vía: conexiones correctas, sin cortocircuitos.

- **Módulo de control de semáforos** => Los semáforos deben de poder controlarse a distancia. Para lograr este objetivo, es necesario completar estos pasos:
    - Diseñar la implementación del módulo: de qué elementos se va a componer el módulo.
    - Generar un BOM: con el diseño establecido, hay que buscar las mejores opciones que el mercado puede ofrecer.
    - Comprar lo decidido en el BOM.
    - Ensamblaje del módulo.
    - Testeo eléctrico del módulo: conexiones correctas, sin cortocircuitos.

- **Módulo de detección de posición** => Se debe de poder conocer la ubicación de la máquina a distancia, es decir, a través de la aplicación. Para lograr este objetivo, es necesario completar estos pasos:
    - Considerar la precisión con la que queremos saber la posición de la máquina, y definir tramos: esto decidirá la cantidad de sensores que serán necesarios
    - Diseñar la implementación del módulo: de qué elementos se va a componer el módulo.
    - Generar un BOM: con el diseño establecido, hay que buscar las mejores opciones que el mercado puede ofrecer.
    - Comprar lo decidido en el BOM.
    - Ensamblaje del módulo.
    - Testeo eléctrico del módulo y sensores: conexiones correctas, sin cortocircuitos. Los sensores funcionan correctamente.

- **Programa principal que gobernará los módulos** => Es el que se encargará de mandar las señales necesarias a los módulos, así como de procesar lo que los sensores detectan e informar la aplicación móvil. Para lograr este objetivo, es necesario completar estos pasos:
    - Codificación del programa.
    - Testeo del programa.

- **Aplicación móvil** => Una vez que todo el sistema funciona, la última parte será la creación de una aplicación móvil para poder controlar al sistema a distancia. Para lograr este objetivo, es necesario completar estos pasos:
    - Codificación de la aplicación.
    - Añadir funcionalidades de comunicación al programa que gobernará los módulos: el programa y la aplicación tienen que poder comunicarse para poder controlar el sistema a distancia.
    - Testeo de todo el sistema con las nuevas funcionalidades.

1.3.2 Requisitos

A continuación, comentamos las exigencias que se espera que cumpla este proyecto:

- Todos los módulos deben funcionar, y no deben de provocar colisiones unos con otros.

- Los módulos deben de poder gobernarse con el programa principal.

- La aplicación debe de poder comunicarse con el programa principal.

- El sistema tiene que poder controlarse a distancia.

1.3.3 Obstáculos y riesgos

Como en todo proyecto, no todo será un camino de rosas, habrá problemas que se tendrán que solucionar, y riesgos que habrá que afrontar. Por ejemplo:

- **Las funcionalidades de comunicación:** los módulos de comunicación, ya sean Wi-Fi o *Bluetooth*, tienen la mala costumbre de no querer funcionar bien a la primera, ya que a priori desconocemos qué *firmware* tienen. Esto puede causar una pérdida de tiempo y recursos que hay que prever.

- **Tratar con elementos con corriente eléctrica:** no hay que olvidar que la maqueta está alimentada por un transformador a 12V, y esto puede causarnos algún que otro susto si no vamos con cuidado.

- **Retrasos:** afrontamos dos tipos de retrasos, el retraso por el servicio de mensajería, y el retraso que esto nos puede causar de cara a la fecha de entrega. No hay que perder de vista la fecha.

- **Pandemia:** desearía que esto se quedase solo en una anécdota, pero no podemos olvidar que a día de la creación de este documento, todavía estamos en alerta por pandemia.

1.4 Metodología y rigor

Durante los primeros días del proyecto, mi director Enric X. y yo, hemos usado tanto el correo electrónico las videollamadas en *meet* para mantener el contacto y ver como avanza el proyecto.

Meet

[9] Se realiza periódicamente para comentar el avance del proyecto y compartir si ha habido alguna complicación, para así intentar encontrar una solución. Se trata también sobre los materiales y recursos que hacen falta para avanzar en el proyecto, así como del diseño del sistema.

Aula de CI

Con el fin de poder hacer uso de material y dispositivos tales como el osciloscopio y los generadores, se me ha autorizado el acceso a esta aula cuando así lo requiera. De esta forma podré probar ideas y recopilar resultados que de otra forma me resultaría mucho más difícil.

Debido a la naturaleza del proyecto, que no está asociado a ningún equipo de desarrollo ni a ninguna empresa, y que requiere primero de la investigación y obtención de los materiales adecuados, no hay una mayor metodología adoptada. Ha sido una decisión acordada en conjunto con el director, por lo que no tengo más que añadir en este apartado de momento. Esto puede cambiar conforme avancen las fases del proyecto.

Figura 1.4: UPC – Campus Nord.

1.5 Descripción de las tareas

La duración aproximada del proyecto es aproximadamente de cinco meses. Comenzó con la concepción de la idea a finales de enero del 2021, y está planeado para finales de Junio o principios de Julio. El tiempo a dedicar cada semana está alrededor de 20 horas, pero puede variar debido a factores externos.

En esta sección explicaré cómo está planeado el proyecto. Estará compuesto de tres tareas generales: gestión del proyecto, desarrollo del proyecto, y documentación del proyecto. Y cada una de esas tareas generales está compuesta de otras sub-tareas.

1.5.1 Gestión del proyecto

Principalmente la documentación de la organización del proyecto, y las reuniones periódicas con mi director. Solamente se requiere el uso del ordenador para llevarlas a cabo.

- **GP1: Contexto y alcance del proyecto** => La definición del contexto y del alcance del proyecto tomará unas 10 horas, y se documenta con un procesador de textos.

- **GP2: Planificación del proyecto** => Realizar la planificación del proyecto tomará unas 10 horas, y se documenta con un procesador de textos y generando un diagrama de Gantt. Requiere que se realice GP1 para completarla.

- **GP3: Presupuesto y sostenibilidad** => Hacer un plan de presupuesto y un análisis de sostenibilidad tomará unas 10 horas, y se documenta con un procesador de textos y unas hojas de cálculo. Requiere que se realice GP2 para completarla

- **GP4: Integración en el documento final** => Agrupar todos los documentos mencionados anteriormente para generar un archivo con la definición del proyecto tomará unas 5 horas utilizando un procesador de textos. Requiere que se realicen GP1, GP2 y GP3 para completarla.

- **GP5: Reuniones periódicas** => Hay reuniones periódicas con mi director para analizar la evolución del proyecto. Normalmente serán de 1 hora a la semana.

1.5.2 Desarrollo del proyecto

Como se podía imaginar, muchas de estas tareas coinciden con los objetivos descritos en la sección de objetivos. Todas estas tareas son de vital importancia para el proyecto.

1.5.2.1 Preparación de la maqueta

Preparar la maqueta para poder, tanto trabajarla, como presentarla. Esto nos permite poder trabajar mejor con los módulos. Requerirá un poco de trabajo manual, y del diseño del mapa de sensores. También se hará uso de un tester.

- **PM1: Mediciones y determinación del tamaño de la base** => Medir las dimensiones de la maqueta para poder determinar de qué tamaño queremos la base. Además de definir el grosor, pues me permitirá realizar las hendiduras necesarias para poder alojar los distintos elementos. Tomará unas 10 horas tener la base lista y requerirá de herramientas.

- **PM2: Diseño estructural de la maqueta** => Una vez disponibles y funcionales todos los elementos, hay que determinar de qué manera colocarlos, aprovechando las hendiduras hechas en la tarea anterior. Esto decidirá también el cableado necesario que ha de instalarse. Tomará unas 15 horas determinar y preparar esta estructura. Requiere que se realicen PM1, MCV5, MCCV6, MCS5, MDP6 y PP2 para completarla.
- **PM3: Instalación de la maqueta y test eléctrico** => Cuando la estructura de sensores ya está colocada en la base, ya podemos colocar encima la maqueta (todo el tramo de vías), fijarla y conectarla. Una vez conectada, hay que asegurarse que todos los terminales continúan funcionando. Tomará 10 horas instalar y fijar la maqueta, y testear que todo funciona (se requerirá un tester). Requiere que se realice PM2 para completarla.

1.5.2.2 Módulo de control de velocidad

Para poder controlar la velocidad, es necesario crear un módulo con el que podernos comunicar mediante el programa principal. Estas tareas requerirán un PC con el *software EAGLE*, un tester, y habilidades de soldadura dependiendo de la complejidad del ensamblado del módulo.

- **MCV1: Diseñar la implementación del módulo** => Diseñar un esquemático del módulo con *EAGLE* y revisarlo por posibles errores. Tomará unas 8 horas realizar el diseño.

- **MCV2: Generar un *bill of materials*** => Considerando la disponibilidad, precio y stock de los materiales, hay que generar un *bill of materials*. A partir de ahora, nos referiremos a esto como BOM. Tomará unas 2 horas y requiere que se realice MCV1 para completarla.

- **MCV3: Comprar lo decidido en el BOM** => Tomará alrededor de 1 semana, ya que es el tiempo que requerirá que envíen los materiales, en condiciones normales. Requiere que se realice MCV2 para completarla.

- **MCV4: Ensamblaje del módulo** => Montar todos los elementos para formar el módulo, haciendo uso de un soldador cuando sea necesario. Tomará unas 6 horas montar el módulo y requiere que se realice MCV3 para completarla.

- **MCV5: Testeo eléctrico del módulo** => Para comprobar que el módulo es funcional, hay que realizarle un test eléctrico. Tomará 2 horas realizar el test eléctrico (se requerirá un tester), y requiere que se realice MCV4 para completarla.

1.5.2.3 Módulo de control de cambios de vía

Para poder controlar el cambio de vía, no solo es necesario crear un módulo con el que comunicarnos mediante el programa principal, si no que también será necesario mecanizar los cambios de vía, para que puedan ser accionados a distancia. Estas tareas requerirán un PC con el *software EAGLE*, un tester y habilidades de soldadura dependiendo de la complejidad el ensamblado del módulo.

- **MCCV1: Diseñar la implementación del módulo** => Diseñar un esquemático del módulo con *EAGLE* y revisarlo por posibles errores. Tomará unas 8 horas realizar el diseño.

- **MCCV2: Generar un BOM** => Considerando la disponibilidad, precio y stock de los materiales, hay que generar un *BOM*. Tomará unas 2 horas y requiere que se realice MCCV1 para completarla.

- **MCCV3: Comprar lo decidido en el BOM** => Tomará alrededor de 1 semana, ya que es el tiempo que requerirá que envíen los materiales, en condiciones normales. Requiere que se realice MCCV2 para completarla.

- **MCCV4: Mecanización del cambio de vía** => Mecanizar el cambio de vía mediante el uso de solenoides, para que estos puedan ser accionados a distancia. Tomará unas 6 horas y requiere que se realice MCCV3 para completarla.

- **MCCV5: Ensamblaje del módulo** => Montar todos los elementos para formar el módulo, haciendo uso de un soldador cuando sea necesario. Tomará unas 6 horas montar el módulo y requiere que se realice MCCV3 para completarla.

- **MCCV6: Testeo eléctrico del módulo y del cambio de vía** => Para comprobar que el módulo y los cambios de vía son funcionales, hay que realizarles un test eléctrico. Tomará 3 horas realizar el test eléctrico (se requerirá un tester). Requiere que se realicen MCCV4 y MCCV5 para completarla.

1.5.2.4 Módulo de control de semáforos

Para poder controlar los semáforos, es necesario crear un módulo con el que podernos comunicar mediante el programa principal. Estas tareas requerirán un PC con el *software EAGLE*, un tester, y habilidades de soldadura dependiendo de la complejidad del ensamblado del módulo.

- **MCS1: Diseñar la implementación del módulo** => Diseñar un esquemático del módulo con *EAGLE* y revisarlo por posibles errores. Tomará unas 8 horas realizar el diseño.

- **MCS2: Generar un BOM** => Considerando la disponibilidad, precio y stock de los materiales, hay que generar un *BOM*. Tomará unas 2 horas y requiere que se realice MCS1 para completarla.

- **MCS3: Comprar lo decidido en el BOM** => Tomará alrededor de 1 semana, ya que es el tiempo que requerirá que envíen los materiales, en condiciones normales. Requiere que se realice MCS2 para completarla.

- **MCS4: Ensamblaje del módulo** => Montar todos los elementos para formar el módulo, haciendo uso de un soldador cuando sea necesario. Tomará unas 6 horas montar el módulo y requiere que se realice MCS3 para completarla.

- **MCS5: Testeo eléctrico del módulo** => Para comprobar que el módulo es funcional, hay que realizarle un test eléctrico. Tomará 2 horas realizar el test eléctrico (se requerirá un tester), y requiere que se realice MCS4 para completarla.

1.5.2.5 Módulo de detección de posición

Para poder detectar la posición, es necesario crear un módulo que gobierne los sensores, con el que podernos comunicar mediante el programa principal. Estas tareas requerirán un PC con el *software EAGLE*, un tester, y habilidades de soldadura dependiendo de la complejidad del ensamblado del módulo.

- **MDP1: Determinar la precisión de la detección** => Analizar cuantos sensores serán necesarios para tener el control de todo el tramo de la vía (las divisiones del tramo), y el lugar en el que colocarán en cada división. Tomará unas 5 horas realizar el análisis.

- **MDP2: Diseñar la implementación del módulo** => Diseñar un esquemático del módulo con *EAGLE* y revisarlo por posibles errores. Tomará unas 8 horas realizar el diseño. Requiere que se realice MDP1 para completarla.

- **MDP3: Generar un BOM** => Considerando la disponibilidad, precio y stock de los materiales, hay que generar un *BOM*. Tomará unas 2 horas y requiere que se realice MDP2 para completarla.

- **MDP4: Comprar lo decidido en el BOM** => Tomará alrededor de 1 semana, ya que es el tiempo que requerirá que envíen los materiales, en condiciones normales. Requiere que se realice MDP3 para completarla.

- **MDP5: Ensamblaje del módulo** => Montar todos los elementos para formar el módulo, haciendo uso de un soldador cuando sea necesario. Tomará unas 6 horas montar el módulo y requiere que se realice MDP4 para completarla.

- **MDP6: Testeo eléctrico del módulo y de los sensores** => Para comprobar que el módulo y los sensores son funcionales, hay que realizarles un test eléctrico. Tomará 3 horas realizar el test eléctrico (se requerirá un tester), y requiere que se realice MDP5 para completarla.

1.5.2.6 Programa principal

Implementación del programa principal para controlar los módulos. Para poder comenzar a desarrollar el programa es necesario que antes se completen los módulos y que estos sean funcionales (hayan pasado el test). Esta tarea requerirá el uso del PC.

- **PP1: Codificación del programa** => Implementar el programa que controla todos los módulos, y que en función de lo que detecten los sensores, actúe en consecuencia. Esta será la tarea de desarrollo que más tiempo consuma, alrededor de unas 40 horas para codificar el programa, y requiere que se realicen MCV5, MCCV6, MCS5 y MDP6 para completarla.

- **PP2: Testeo del programa** => Testeo y verificación de que los módulos responden según lo que el programa ordena. Tomará unas 20 horas testear el programa y requiere que se realice PP1 para completarla.

1.5.2.7 Aplicación móvil

Implementación de la aplicación móvil para poder controlar la maqueta desde el móvil. Para poder comenzar a desarrollar la aplicación, es necesario que el programa principal funcione y haya pasado el testeo. Esta tarea requerirá el uso del PC.

- **AM1: Codificación de la aplicación** => Implementar la aplicación de móvil, que enviará las órdenes para controlar la maqueta. Tomará alrededor de unas 30 horas para codificar la aplicación y requiere que se realice PP2 para completarla.

- **AM2: Añadir funcionalidades de comunicación al programa principal** => Añadir las funcionalidades para que el programa principal y la aplicación puedan comunicarse. Tomará alrededor de unas 20 horas y requiere que se realice AM1 para completarla.

- **AM3: Testeo final** => Testeo y verificación de que todo el sistema funciona (programa y aplicación). Tomará unas 20 horas testear todo el sistema y requiere que se realice AM2 para completarla.

1.5.3 Documentación del proyecto

A lo largo del proyecto, se debe documentar todos los eventos que se van realizando. Para esta tarea solo es necesario el uso de un procesador de texto.

- **DP1: Anotación de eventos** => Documentar a medida que avanzamos en el proyecto, por lo que esta tarea se irá haciendo siempre en paralelo.

- **DP2: Revisión de eventos** => Una vez acabado el proyecto, hay que revisar toda la documentación realizada previamente para así poder reorganizarla y ordenarla de manera apropiada. Tomará unas 20 horas revisarlo todo y requiere que se realice DP1 para completarla.

- **DP3: Escribir documentación** => Con todos los eventos ordenados y organizados, es el momento de escribir el documento final. Esta tarea tomará alrededor de 50 horas o más, y requiere que se realice DP2 para completarla.

1.6 Estimación y diagrama de Gantt

En esta sección tenemos una vista preliminar de la estimación del tiempo (Tabla 1), así como el diagrama de Gantt (Figura 6).

ID	Nombre	Duración (h)	Requisitos	Recursos
	Gestión del proyecto	50		
GP1	*Contexto y alcance del proyecto*	10		PC (ofimática)
GP2	*Planificación del proyecto*	10	GP1	PC (ofimática), GanttProject
GP3	*Presupuesto y sostenibilidad*	10	GP2	PC (ofimática)
GP4	*Integración en el documento final*	5	GP1, GP2, GP3	PC (ofimática)
GP5	*Reuniones periódicas*	15		PC (meet)
	Desarrollo del proyecto	254		
	Preparación de la maqueta	35		
PM1	*Mediciones y determinación del tamaño de la base*	10		PC, herramientas
PM2	*Diseño estructural de la maqueta*	15	PM1, MCV5, MCCV6, MCS5, MDP6, PP2	PC
PM3	*Instalación de la maqueta y test eléctrico*	10	PM2	PC, tester
	Módulo de control de velocidad	19		
MCV1	*Diseñar la implementación del módulo*	8		PC (EAGLE)
MCV2	*Generar un BOM*	2	MCV1	PC
MCV3	*Comprar lo decidido en el BOM*	1	MCV2	PC
MCV4	*Ensamblaje del módulo*	6	MCV3	PC, soldador
MCV5	*Test eléctrico del módulo*	2	MCV4	PC, tester
	Módulo de control de cambios de vía	26		
MCCV1	*Diseñar la implementación del módulo*	8		PC (EAGLE)
MCCV2	*Generar un BOM*	2	MCCV1	PC
MCCV3	*Comprar lo decidido en el BOM*	1	MCCV2	PC
MCCV4	*Mecanización del cambio de vía*	6	MCCV3	PC, soldador
MCCV5	*Ensamblaje del módulo*	6	MCCV3	PC, soldador
MCCV6	*Test eléctrico del módulo y del cambio de vía*	3	MCCV4, MCCV5	PC, tester
	Módulo de control de semáforos	19		
MCS1	*Diseñar la implementación del módulo*	8		PC (EAGLE)
MCS2	*Generar un BOM*	2	MCS1	PC
MCS3	*Comprar lo decidido en el BOM*	1	MCS2	PC
MCS4	*Ensamblaje del módulo*	6	MCS3	PC, soldador
MCS5	*Test eléctrico del módulo*	2	MCS4	PC, tester
	Módulo de detección de posición	25		
MDP1	*Determinal la precisión de la detección*	5		PC
MDP2	*Diseñar la implementación del módulo*	8	MDP1	PC (EAGLE)
MDP3	*Generar un BOM*	2	MDP2	PC
MDP4	*Comprar lo decidido en el BOM*	1	MDP3	PC
MDP5	*Ensamblaje del módulo*	6	MDP4	PC, soldador
MDP6	*Test eléctrico del módulo y de los sensores*	3	MDP5	PC, tester
	Programa principal	60		
PP1	*Codificación del programa*	40	MCV5, MCCV6, MCS5, MDP6	PC (Arduino IDE y MPLAB X IDE)
PP2	*Testeo del programa*	20	PP1	PC, Arduino, maqueta
	Aplicación móvil	70		
AM1	*Codificación de la aplicación*	30	PP2	PC (App Inventor)
AM2	*Añadir funcionalidades de comunicación al programa principal*	20	AM1	PC (Arduino IDE)
AM3	*Testeo final*	20	AM2	PC, Arduino, smatphone, maqueta
	Documentación del proyecto	60		
DP1	*Anotación de eventos*	10		PC (ofimática)
DP2	*Revisión de eventos*	10	DP1	PC (ofimática)
DP3	*Escribir documentación*	40	DP2	PC (ofimática)
	Total (tiempo de envío de materiales incluido)	532		

Tabla 1.1: Resumen de las tareas a realizar.

* En el siguiente diagrama de Gantt se han omitido algunos de los requisitos de PM2 (el requisito PP2 incluye de forma implícita los requisitos de los módulos) para evitar complejidad visual. Nótese que durante el desarrollo de los módulos he decidido esperar a llegar en todos a la misma tarea (comprar el BOM), para poder realizarlos todos a la vez y que el tiempo de espera del envío sea mínimo.

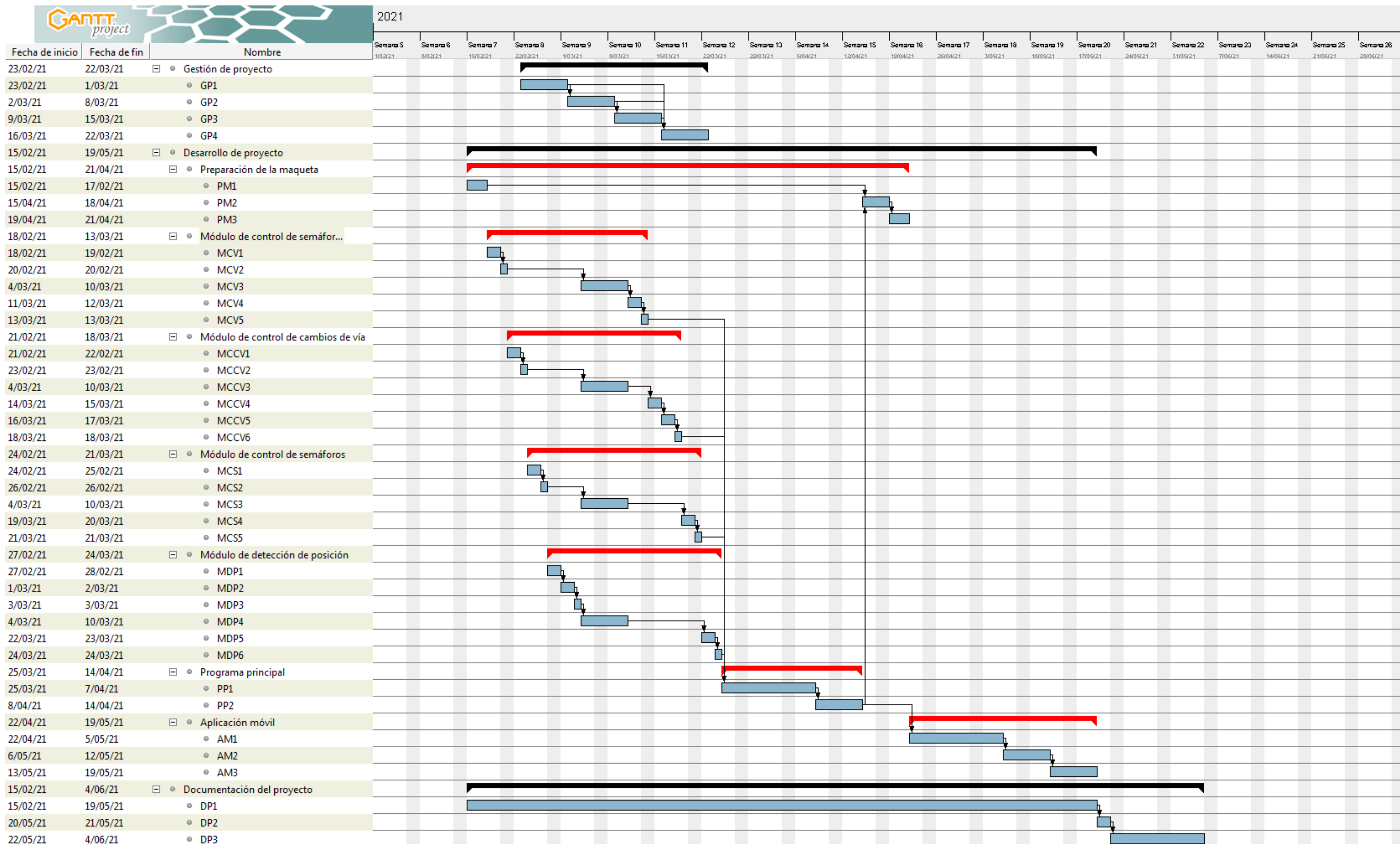


Figura 1.5: Diagrama de Gantt.

1.7 Documentación del proyecto

El resto del documento está estructurado de la siguiente forma:

- Capítulo 2: Introducción y características de los Microcontroladores PIC.
- Capítulo 3: Diseño del sistema con un Arduino como *master*, y los PICs como *slaves*.
- Capítulo 4: Diseño de la aplicación móvil y integración con el sistema.
- Capítulo 5: Evaluación de la funcionalidad del proyecto.
- Capítulo 6: Conclusiones.
- Apéndices, glosario y bibliografía para aportar más información.

2 Microcontroladores PIC

Como hemos dicho en un comienzo, las siglas PIC hacen referencia a *Programmable Integrated Circuited*. Sin embargo el verdadero nombre del microcontrolador PIC es PICmicro, y sus siglas significan algo diferente: *Perihperial Interface Controller*. Como nota curiosa, el primero creado fue el PIC1650 alrededor del 1975 por la compañía *General Instruments*. Diez años más tarde y tras añadir una memoria *EEPROM*, siglas de *Electrically Erasable Programable Read-Only Memory*, fue cuando los microcontroladores PIC se convirtieron en lo que tenemos hoy en día [14].

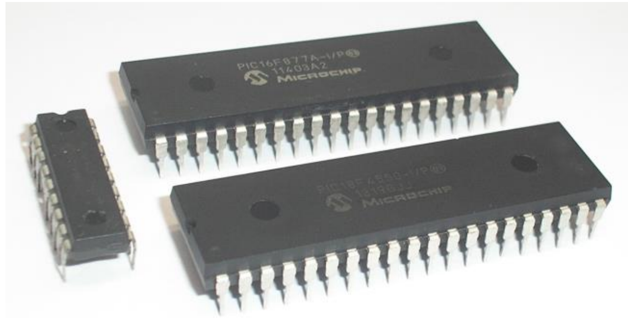


Figura 2.1: Microcontroladores PIC de *Microchip* [14].

El uso de estos microcontroladores es perfecto, ya que dada la cantidad enorme de periféricos para los cuales se pueden configurar, los convierten en una opción muy versátil. Además, es muy factible hacerse con unas cuantas unidades a un coste muy reducido. Estas y otras ventajas comentadas más adelante hacen que sean los motivos por los cuales han sido escogidos para este proyecto.

2.1 Características de los PIC

Todos los microcontroladores PIC hacen uso de una arquitectura *Harvard*, y dependiendo de la anchura del bus hay microcontroladores de 12, 14, y 16 bits. La Figura 2.2 muestra algunas características de esas categorías:

Familia	ROM [Kbytes]	RAM [bytes]	Pines	Frecuencia de reloj. [MHz]	Entradas A/D	Resolución del convertidor A/D	Comparadores	Temporizadores de 8/16 bits	Comunicación serial	Salidas PWM	Otros
Arquitectura de la gama baja de 8 bits, palabra de instrucción de 12 bits											
PIC10FXXX	0.375 - 0.75	16 - 24	6 - 8	4 - 8	0 - 2	8	0 - 1	1 x 8	-	-	-
PIC12FXXX	0.75 - 1.5	25 - 38	8	4 - 8	0 - 3	8	0 - 1	1 x 8	-	-	EEPROM
PIC16FXXX	0.75 - 3	25 - 134	14 - 44	20	0 - 3	8	0 - 2	1 x 8	-	-	EEPROM
PIC16HVXXX	1.5	25	18 - 20	20	-	-	-	1 x 8	-	-	Vdd = 15V
Arquitectura de la gama media de 8 bits, palabra de instrucción de 14 bits											
PIC12FXXX	1.75 - 3.5	64 - 128	8	20	0 - 4	10	1	1 - 2 x 8 1 x 16	-	0 - 1	EEPROM
PIC12HVXXX	1.75	64	8	20	0 - 4	10	1	1 - 2 x 8 1 x 16	-	0 - 1	-
PIC16FXXX	1.75 - 14	64 - 368	14 - 64	20	0 - 13	8 or 10	0 - 2	1 - 2 x 8 1 x 16	USART I2C SPI	0 - 3	-
PIC16HVXXX	1.75 - 3.5	64 - 128	14 - 20	20	0 - 12	10	2	2 x 8 1 x 16	USART I2C SPI	-	-
Arquitectura de la gama alta de 8 bits, palabra de instrucción de 16 bits											
PIC18FXXX	4 - 128	256 - 3936	18 - 80	32 - 48	4 - 16	10 or 12	0 - 3	0 - 2 x 8 2 - 3 x 16	USB2.0 CAN2.0 USART I2C SPI	0 - 5	-
PIC18FXXJXX	8 - 128	1024 - 3936	28 - 100	40 - 48	10 - 16	10	2	0 - 2 x 8 2 - 3 x 16	USB2.0 USART Ethernet I2C SPI	2 - 5	-
PIC18FXXKXX	8 - 64	768 - 3936	28 - 44	64	10 - 13	10	2	1 x 8 3 x 16	USART I2C SPI	2	-

Figura 2.2: Características principales de los PIC de 12, 14 y 16 bits de *Microchip* [14].

Estos microcontroladores tienen una arquitectura *Harvard* de 8 bits y la categoría la determina el tamaño de palabra de programa (*word*). Esta arquitectura tiene dos tipos de buses, un bus de programa y un bus de datos, lo que permite acceder a la vez a la memoria del programa y a la memoria de datos. Como tienen la misma arquitectura, la estructura básica de su *hardware* es similar, y utilizan el mismo juego de instrucciones. La diferencia está en la cantidad de periféricos a los que ese *hardware* está conectado.

Los microcontroladores PIC de *Microchip* gozan de varias ventajas, entre otras [14]:

- Eficiencia de código que permite compactar programas (y ahorrar memoria).
- Buenos parámetros de frecuencia que permiten una ejecución veloz.
- Seguridad en los accesos debido a la separación de las memorias de datos y programa.
- Al ser *RISC*, facilita su aprendizaje y uso.
- A pesar de las diferentes cantidades de pines y disposición entre varios modelos, estos son compatibles: configurar un registro de puerto PORT# (donde, dependiendo del modelo, # puede ser A, B, C...) se hace igual en un PIC de 8 pines que en un PIC de 20 pines, por ejemplo. Esto facilita la reutilización y compatibilidad de código.
- Gran variedad de versiones en distintos encapsulados sin reducción de prestaciones: versiones para placas troqueladas, para placas *PCBs*…
- Dispone de herramientas *software* para proteger el código con el que se programará.
- Herramientas de desarrollo accesibles (de hecho, haremos uso de una de ellas en este proyecto: *MPLAB X IDE*)

Los PICs que usaremos en este proyecto son el PIC16F15313 y el PIC16F15323, de 8 y 14 pines respectivamente, y con una arquitectura de 8 bits. Estos PICs son de la misma familia, por lo que comparten muchas características y ahorrará tiempo al estudiar su funcionamiento en el *datasheet* [13]. En la Figura 2.3 y 2.4 podemos ver la disposición de pines de los respectivos PICs:

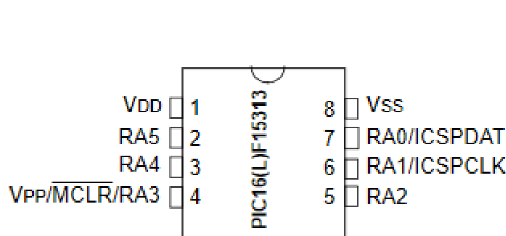


Figura 2.3: Pinout PIC16F15313 [13].

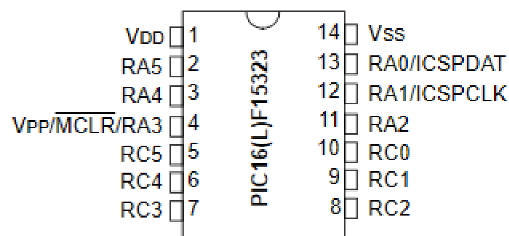


Figura 2.4: Pinout PIC16F15323 [13].

A continuación tenemos la Figura 2.5 y 2.6 que nos muestran unos diagramas de bloques con los periféricos que tiene cada PIC.

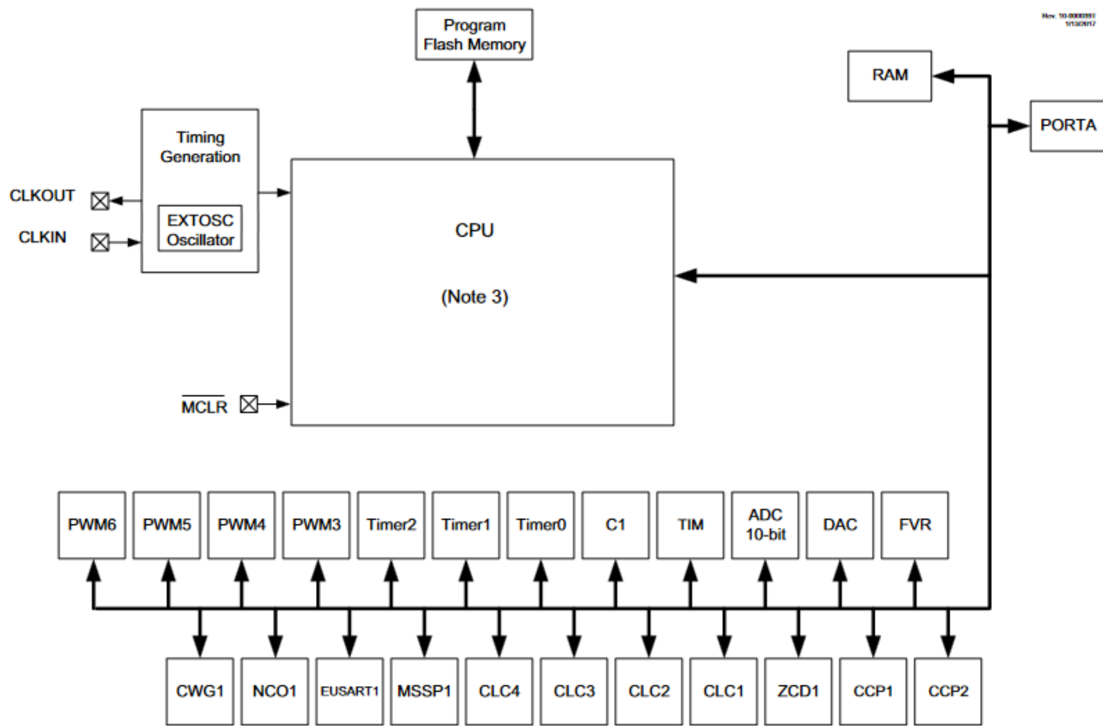


Figura 2.5: Diagrama de bloques PIC16F15313 [13].

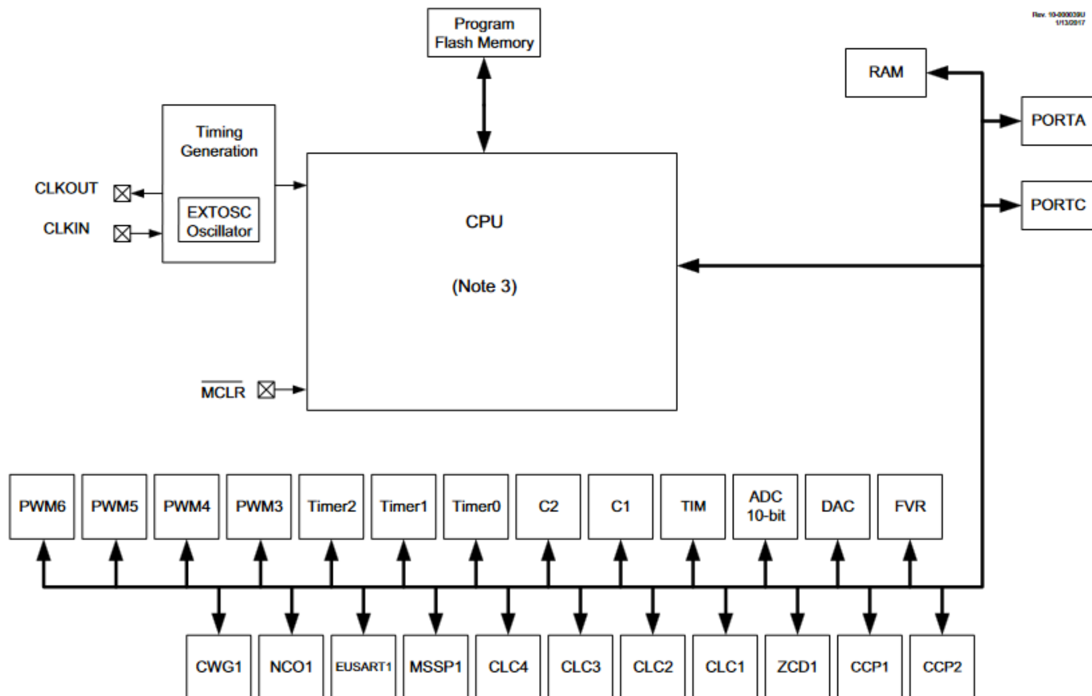


Figura 2.6: Diagrama de bloques PIC16F15323 [13].

Cuyas siglas significan [13]:

- FVR: *Fixed Voltage Reference*
- DAC: *Digital-to-Analog Converter*
- ADC: *Analog-to-Digital Converter*
- TIM: *Temperature Indicator Module*
- C#: *Comparator Module*
- Timer#: *Timers*
- PWM#: *Pulse-Width Modulator*
- CCP#: *Capture/Compare/PWM Modules*
- ZCD#: *Zero-Cross Detect*
- CLC#: *Configurable Logic Cell*
- MSSP#: *Master Synchronous Serial Ports*
- EUSART#: *Enhanced Universal Synchronous/Asynchronous Receiver/Transmitter*
- NCO#: *Numerically Controlled Oscillator*
- CWG#: *Complementary Waveform Generator*

Como podemos ver, tenemos una cantidad de periféricos importante. Prestaremos especial atención al periférico MSSP, que detallaremos más adelante.

2.2 Programación del PIC

Para poder transferir el código del programa del ordenador al PIC se hace uso de un dispositivo: el programador. Además, actualmente los PICs de *Microchip* incorporan *ICSP* y *LVP*, siglas de *In Circuit Serial Programming* y *Low Voltatge Programming* respectivamente. Esto permite programar el PIC directamente en el circuito al que está destinado.

Existen varios programadores de PICs, desde los más sencillos que dejan al *software* todos los detalles de las comunicaciones, a los más complejos que pueden verificar el dispositivo a distintas tensiones de alimentación y tienen implementadas en *hardware* la mayoría de sus funcionalidades. De hecho, algunos de estos programadores hacen uso propio de PICs preprogramados como interfaz para enviar las órdenes al PIC que se quiere programar.

En este proyecto haremos uso del programador PICKIT 3 de *Microchip*, y quisiera agradecer a los laboratorios del departamento ESAII de la FIB - UPC que me proporcionaran uno de ellos, ya que facilitará considerablemente el trabajo de testeo. En la Figura 2.7 podemos observar la disposición de pines del programador, y aunque vemos que tiene seis pines, solo necesitamos usar los cinco primeros para programar un PIC mediante *ICSP*.

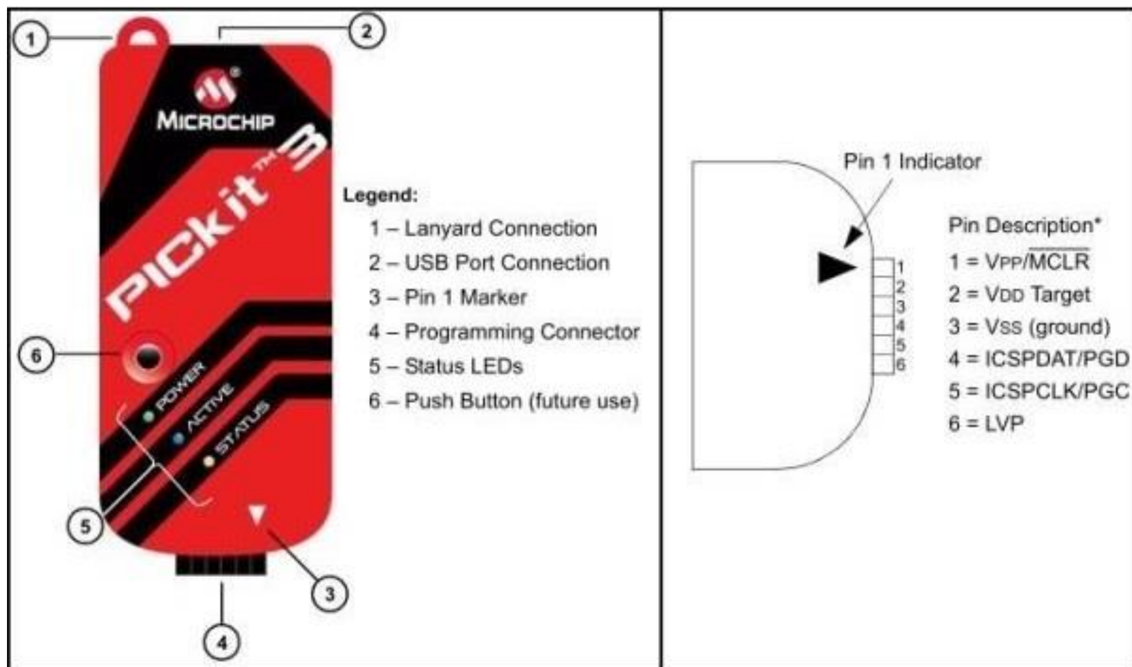


Figura 2.7: Disposición de pines del programador PICKIT 3.

2.3 Set de Instrucciones

Como los PICs que usaremos son de la misma familia, comparten el mismo set de instrucciones. Cada instrucción es un *word* de 14 bits que contiene el código de operación (*opcode*) y todos los operandos requeridos. Según el código de operación, distinguimos tres categorías:

- Operaciones orientadas a Byte
- Operaciones orientadas a Bit
- Operaciones de Literales y Control

Esta última categoría se refiere a las operaciones que tratan con valores inmediatos y de control de secuencia. Es la categoría que contiene los *word* más variados para sus instrucciones.

En la Tabla 2.1 están listadas todas las instrucciones reconocidas por *MPASM*, que es el ensamblador estándar para los PIC de 8 bits. Todas las instrucciones tienen un *CPI* (ciclos por instrucción) de valor 1, a excepción de las siguientes, que pueden tener un *CPI* de valor 2 [13]:

- La llamada a una subrutina requiere 2 ciclos (*CALL*, *CALLW*)
- El retorno de una interrupción o subrutina requiere 2 ciclos (*RETURN*, *RETLW*, *RETFIE*)
- Los saltos condicionales e incondicionales requieren 2 ciclos (*GOTO*, *BRA*, *BRW*, *BTFSS*, *BTFSC*, *DECFSZ*, *INCSFZ*)

Un ciclo de instrucción consta de cuatro ciclos del oscilador; para una frecuencia de oscilador de 4 MHz, da un ratio de ejecución de instrucciones de 1 MHz.

Mnemonic, Operands		Description	Cycles	14-Bit Opcode				Status Affected	Notes
				MSb			LSb		
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C, DC, Z	2
ADDWFC	f, d	Add with Carry W and f	1	11	1101	dfff	ffff	C, DC, Z	2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	2
ASRF	f, d	Arithmetic Right Shift	1	11	0111	dfff	ffff	C, Z	2
LSLF	f, d	Logical Left Shift	1	11	0101	dfff	ffff	C, Z	2
LSRF	f, d	Logical Right Shift	1	11	0110	dfff	ffff	C, Z	2
CLRF	f	Clear f	1	00	0001	lfff	ffff	Z	2
CLRW	–	Clear W	1	00	0001	0000	00xx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	2
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z	2
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	2
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	2
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	2
MOVWF	f	Move W to f	1	00	0000	1fff	ffff		2
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	2
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	2
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C, DC, Z	2
SUBWFB	f, d	Subtract with Borrow W from f	1	11	1011	dfff	ffff	C, DC, Z	2
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff		2
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	2
BYTE ORIENTED SKIP OPERATIONS									
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00	1011	dfff	ffff		1, 2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00	1111	dfff	ffff		1, 2
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff		2
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff		2
BIT-ORIENTED SKIP OPERATIONS									
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff	ffff		1, 2
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bfff	ffff		1, 2
LITERAL OPERATIONS									
ADDLW	k	Add literal and W	1	11	1110	kkkk	kkkk	C, DC, Z	
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z	
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLB	k	Move literal to BSR	1	00	0000	001k	kkkk		
MOVLP	k	Move literal to PCLATH	1	11	0001	1kkk	kkkk		
MOVLW	k	Move literal to W	1	11	0000	kkkk	kkkk		
SUBLW	k	Subtract W from literal	1	11	1100	kkkk	kkkk	C, DC, Z	
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z	

Note 1: If the Program Counter (PC) is modified, or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

2: If this instruction addresses an INDF register and the MSb of the corresponding FSR is set, this instruction will require one additional instruction cycle.

Tabla 2.1: Set de instrucciones para el PIC16F15313/23 [13].

Mnemonic,	Operands	Description	Cycles	14-Bit Opcode MSb			LSb	Status Affected	Notes
CONTROL OPERATIONS									
BRA	k	Relative Branch	2	11	001k	kkkk	kkkk		
BRW	–	Relative Branch with W	2	00	0000	0000	1011		
CALL	k	Call Subroutine	2	10	0kkk	kkkk	kkkk		
CALLW	–	Call Subroutine with W	2	00	0000	0000	1010		
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk		
RETFIE	k	Return from interrupt	2	00	0000	0000	1001		
RETLW	k	Return with literal in W	2	11	0100	kkkk	kkkk		
RETURN	–	Return from Subroutine	2	00	0000	0000	1000		
INHERENT OPERATIONS									
CLRWDT	–	Clear Watchdog Timer	1	00	0000	0110	0100	$\overline{TO}$, $\overline{PD}$	
NOP	–	No Operation	1	00	0000	0000	0000		
RESET	–	Software device Reset	1	00	0000	0000	0001		
SLEEP	–	Go into Standby or IDLE mode	1	00	0000	0110	0011	$\overline{TO}$, $\overline{PD}$	
TRIS	f	Load TRIS register with W	1	00	0000	0110	0fff		
C-COMPILER OPTIMIZED									
ADDFSR	n, k	Add Literal k to FSRn	1	11	0001	0nkk	kkkk		
MOVIW	n mm	Move Indirect FSRn to W with pre/post inc/dec modifier, mm	1	00	0000	0001	0nmm	Z	2, 3
	k[n]	Move INDFn to W, Indexed Indirect.	1	11	1111	0nkk	kkkk	Z	2
MOVWI	n mm	Move W to Indirect FSRn with pre/post inc/dec modifier, mm	1	00	0000	0001	1nmm		2, 3
	k[n]	Move W to INDFn, Indexed Indirect.	1	11	1111	1nkk	kkkk		2

Note 1: If the Program Counter (PC) is modified, or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

2: If this instruction addresses an INDF register and the MSb of the corresponding FSR is set, this instruction will require one additional instruction cycle.

3: See Table in the MOVIW and MOVWI instruction descriptions.

Tabla 2.1 (continuación): Set de instrucciones para el PIC16F15313/23 [13].

2.4 MSSP – Master Synchronous Serial Port

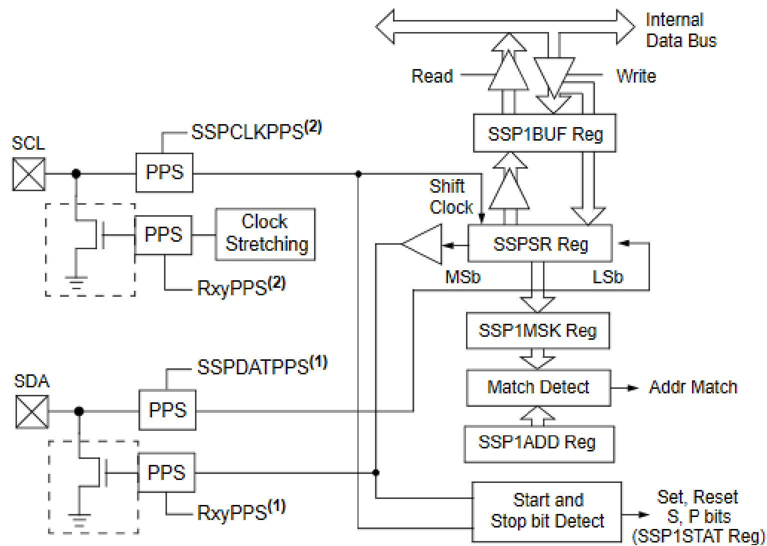
El PIC16F15313/23 posee un módulo *MSSP* para la comunicación mediante interfaz serie que le permite comunicarse con otros periféricos (como podría ser una pantalla *LCD*, conversores A/D, *EEPROMs*) o microcontroladores. Este último es nuestro caso, ya que los PICs que usaremos en el diseño se comunicarán con una Arduino. El módulo *MSSP* puede operar en uno de estos dos modos [13]:

- SPI: *Serial Perihperial Interface*
- I2C: *Inter-Integrated Circuit*

En este proyecto solo haremos uso de la interfaz I2C, que soporta los siguientes modos y posee las siguientes características [13]:

- *Master mode*
- *Slave mode*
- *Byte NACKing (Slave mode)*
- *Limited multi-master support*
- *7-bit and 10-bit addressing*
- *Start and Stop interrupts*
- *Interrupt masking*
- *Clock stretching*
- *Bus collision detection*
- *General call address matching*
- *Address masking*
- *Selectable SDA hold times*

Los PICs que configuraremos en este proyecto sólo actuarán en modo *slave*. En la Figura 2.8 podemos ver un diagrama de bloques del *MSSP* haciendo uso del I2C en modo *slave*.



Note 1: SDA pin selections must be the same for input and output.
2: SCL pin selections must be the same for input and output.

Figura 2.8: Diagrama de bloques del *MSSP* (I2C *slave*) [13].

Los más importantes a destacar son el registro de buffer *SSP1BUF* que es donde se guardan los datos a enviar posteriormente, o donde se almacenan los datos que se reciben; y los registros *SSP1ADD* y *SSP1MSK* que contienen la dirección del dispositivo y la máscara respectivamente. *SSP1ADD* y *SSP1MSK* se usan para detectar cuando un dispositivo *master* se quiere comunicar con el dispositivo en cuestión. Son los tres registros usados principalmente a la hora de hacer el código para programar un PIC para que haga uso del I2C. El resto de registros solo contienen información de la configuración I2C.

2.4.1 I2C

El bus I2C es un bus serie para la comunicación de datos donde los dispositivos se comunican en un entorno *master/slave*, donde el dispositivo *master* inicia siempre la comunicación. Los dispositivos *slave* se controlan mediante un direccionamiento. El bus I2C requiere la conexión de dos señales [13]:

- *Serial Clock (SCL)*
- *Serial Data (SDA)*

Ambas conexiones son bidireccionales y requieren de resistencias *pull-up* para el suministro de tensión. Un VIL se considera un cero lógico, y un VIH se considera un uno lógico [22]. En la Figura 2.9 se muestra una conexión típica entre dos microcontroladores configurados como dispositivos *master* y *slave*.

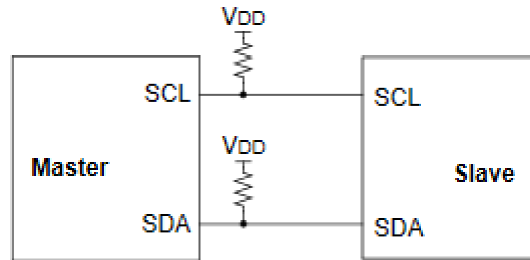


Figura 2.9: Conexión entre *master* y *slave* [13].

El bus I2C puede operar con uno o varios *master* y uno o varios *slave*. Existen cuatro modos diferentes dado un dispositivo [13]:

- *Master Transmit mode*: el *master* transmite al *slave*.
- *Master Recieve mode*: el *master* recibe del *slave*.
- *Slave Transmit mode*: el *slave* transmite al *master* (a petición del *master*).
- *Slave Recieve mode*: el *slave* recibe del *master* (a petición del *master*).

Para iniciar la comunicación, un dispositivo *master* comienza en *master transmit mode*. El dispositivo *master* envía un bit de start seguido de un byte que contiene la dirección del *slave* con quien intenta comunicarse (las direcciones son de 7 bits típicamente, aunque existe un modo de direccionamiento de hasta 10 bits). En la Figura 2.10 vemos la condición de start por parte del *master*.

A continuación le sigue un bit que determina la operación que el *master* desea iniciar, *read*/*write*: si el master quiere recibir será un *read*, si el master quiere transmitir será un *write*. Si el dispositivo con quien el *master* se quiere comunicar existe en el bus, este responderá con un bit de *ACK* (siglas de *acknowledge*). A partir de este momento y en función de la operación que el *master* ha querido iniciar, el *slave* atenderá la petición del *master* de forma complementaria: si el *master* quería transmitir, el *slave* recibirá, y si el *master* quería recibir, el *slave* transmitirá. Una vez se ha transmitido el último byte de datos, el *master* enviará un bit de stop y la comunicación se da por finalizada. La condición de stop se puede apreciar en la Figura 2.10.

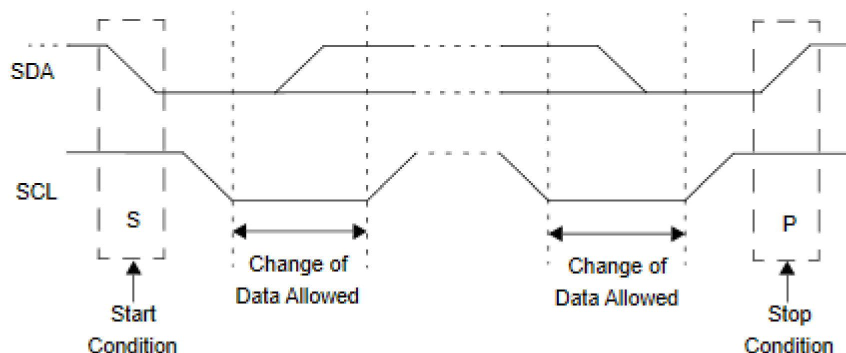


Figura 2.10: Condiciones de start y stop en I2C [13].

Cada *master* monitorea el bus en busca de un bit de start y stop. Si el dispositivo detecta que el bus está ocupado, esperará a que esté libre para usarlo. Si hay varios *master* y se intentan comunicar a la vez, se inicia el periodo de arbitraje en el que los dispositivos afectados monitorean la línea de *SDA*. Aquel dispositivo que no vea ninguna diferencia del valor esperado en la línea *SDA*, será el que transmitirá. Normalmente sucede que es el último dispositivo en llegar el que se queda la línea, precisamente porque no viene nadie detrás de el que se la modifique, y que por tanto la línea *SDA* no tenga el valor esperado por el dispositivo.

2.4.2 I2C – *Slave Recieve mode*

En este modo el PIC trabaja como *slave* en modo recepción, es decir, espera a que el *master* le envíe una solicitud de escritura. Una conexión exitosa típica en I2C para este modo, sigue estos pasos [13]:

- 1. Un bit de start por parte del *master* se envía al bus *SDA* seguido de un byte que contiene una dirección y el tipo de petición (recordemos: *read*/*write*).
- 2. El PIC *slave* detecta el bit de start. Este suceso queda registrado en el PIC mediante un bit (*S*) en un registro llamado *SSP1STAT* que contiene información del transcurrir de la comunicación.
- 3. El PIC compara la dirección recibida con la suya, almacenada en *SSP1ADD*, pero solo hace caso al resultado de la comparación si la petición es un *write* (recordemos: estamos en modo recepción).
- 4. Si la comparación es exitosa, la dirección se almacena en *SSP1BUF* y el *slave* manda automáticamente un *ACK* al *master* (automáticamente implica que es realizado por el *hardware*). Además, se genera una interrupción mediante la activación de un *flag*.
- 5. El *software* del PIC *slave* librea el *flag* y lee el valor de *SSP1BUF* (que contiene la dirección), vaciándolo en el acto. Esto es necesario, pues para poder almacenar el siguiente byte el registro *SSP1BUF* debe estar vacío.
- 6. El *master* envía un byte con datos.
- 7. El *slave* lo recibe, genera una interrupción de la misma forma, y le manda un *ACK*.
- 8. El *software* del PIC *slave* librea el *flag* y lee el valor de *SSP1BUF* (esta vez contiene datos), vaciándolo en el acto nuevamente.
- 9. Los pasos 6 – 8 se repiten por cada byte que el *master* le envíe.
- 10. El *master* envía un bit de stop al bus *SDA* que es detectado por el PIC *slave*. Este suceso queda registrado en el PIC mediante otro bit (*P*) en el registro *SSP1STAT*. La comunicación se detiene y el bus queda desocupado.

La Figura 2.11 puede usarse como referencia visual para la descripción de pasos anterior. En ella podemos observar un cronograma del proceso de comunicación I2C como *slave* en modo recepción.

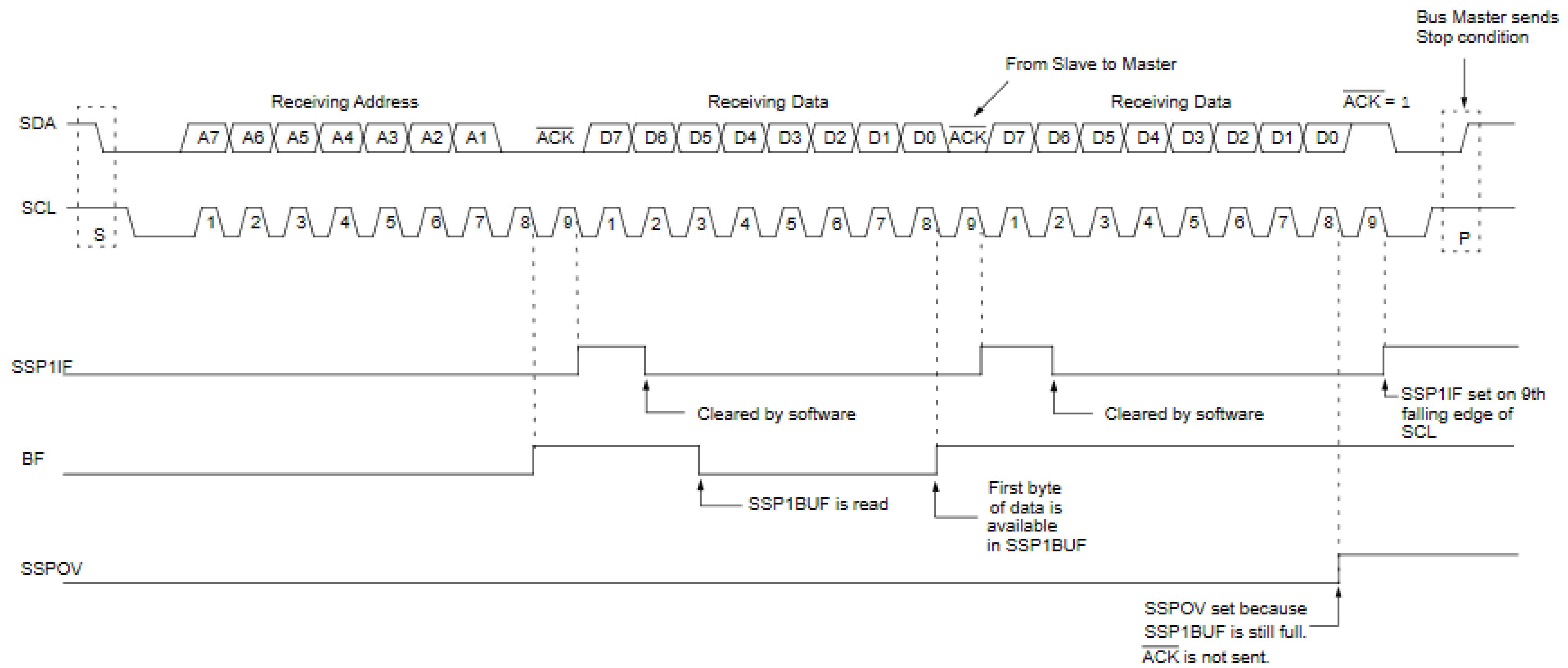


Figura 2.11: I2C *slave recieve mode* con direccionamiento de 7 bits [13].

2.4.3 I2C – *Slave Transmit mode*

En este otro modo el PIC trabaja como *slave* en modo transmisión, es decir, espera a que el *master* le envíe una solicitud de lectura. Una conexión exitosa típica en I2C para este modo, sigue estos pasos [13]:

- 1. Un bit de start por parte del *master* se envía al bus *SDA* seguido de un byte que contiene una dirección y el tipo de petición (recordemos: *read*/*write*).
- 2. El PIC *slave* detecta el bit de start. Este suceso queda registrado en el PIC mediante el bit (*S*) en el registro *SSP1STAT*.
- 3. El PIC compara la dirección recibida con la suya, almacenada en *SSP1ADD*, pero solo hace caso al resultado de la comparación si la petición es un *read* (recordemos: estamos en modo transmisión).
- 4. Si la comparación es exitosa, la dirección se almacena en *SSP1BUF* y el *slave* manda automáticamente un *ACK* al *master*. Además, se genera una interrupción mediante la activación de un *flag*.
- 5. El *software* del PIC *slave* librea el *flag* y lee el valor de *SSP1BUF* (que contiene la dirección), vaciándolo en el acto. Esto es necesario, pues debemos cargar en *SSP1BUF* el valor que queremos transmitir.
- 6. El *slave* carga un byte de datos en *SSP1BUF*.
- 7. El *slave* envía un byte con datos.
- 8. El *master* lo recibe y responde con un *ACK* al *slave*.
- 9. El *ACK* por parte del *master* genera una interrupción en el PIC *slave*.
- 10. El *software* del PIC *slave* librea el *flag* y consulta el bit *ACKSTAT* del registro de configuración *SSP1CON2*, que contiene el *ACK* enviado por el *master*. Este registro también contiene información de la comunicación y sirve para comprobar si hay que continuar transmitiéndole bytes de datos al *master*.
- 11. Los pasos 6 – 10 se repiten por cada byte que el *master* le envíe.
- 12. El *master* envía un bit de stop al bus *SDA* que es detectado por el PIC *slave*. Este suceso queda registrado en el PIC mediante otro bit (*P*) en el registro *SSP1STAT*. La comunicación se detiene y el bus queda desocupado.

La Figura 2.12 puede usarse como referencia visual para la descripción de pasos anterior de forma análoga al apartado previo. En ella podemos observar un cronograma del proceso de comunicación I2C como *slave* en modo transmisión.

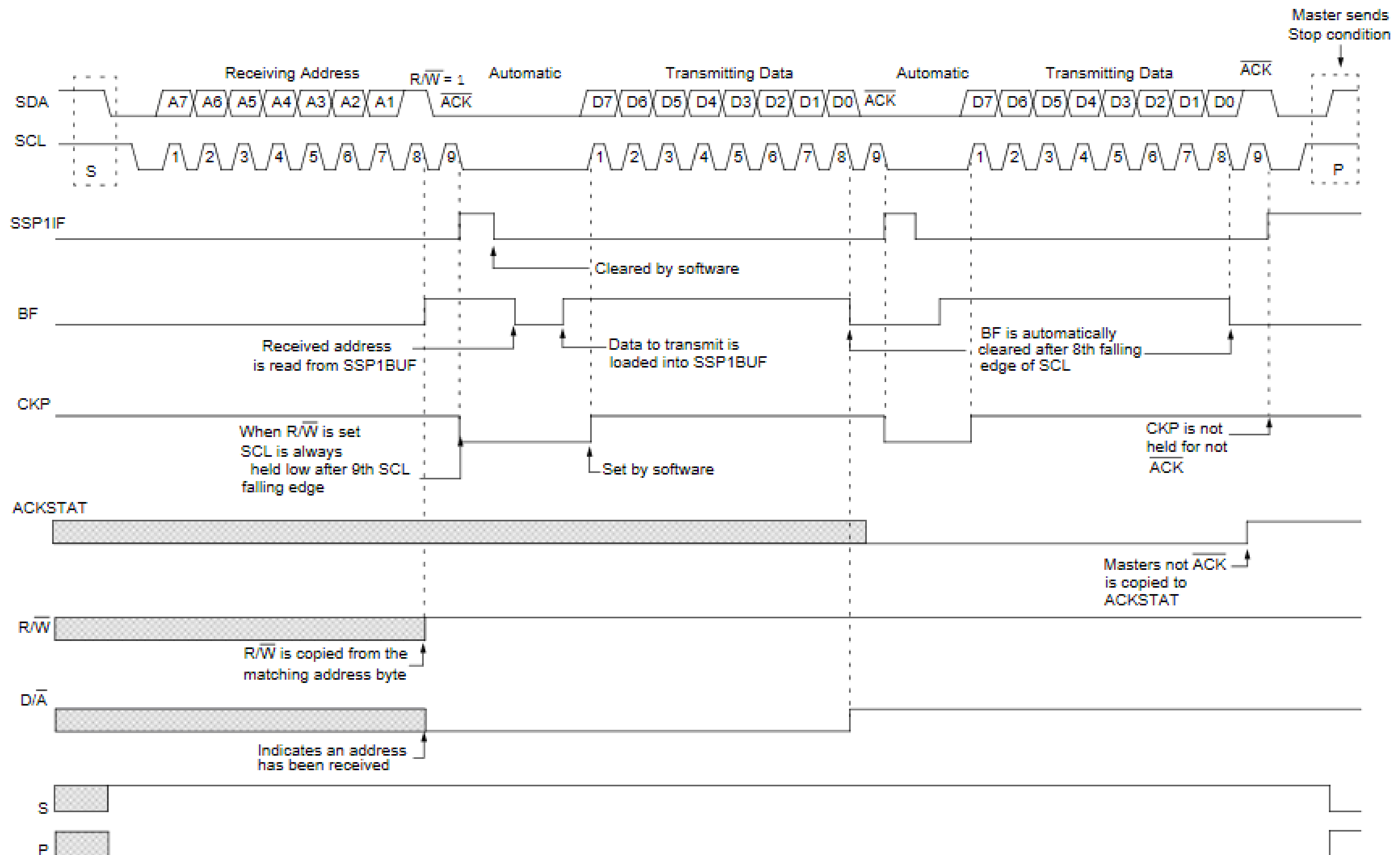


Figura 2.12: I2C *slave transmit mode* con direccionamiento de 7 bits [13].

2.5 Test de los PICs

Ahora que ya hemos contemplado las herramientas de las que haremos uso en los PICs, podemos pasar a realizar pruebas sobre ellos para ver que se adecuan a las necesidades del proyecto.

Aunque pueden programarse mediante lenguaje ensamblador, lo más apropiado es hacer uso de la IDE que *Microchip* nos ofrece: MPLAB X IDE. Siendo la IDE por excelencia para programar los PICs, nos permite hacerlo además en C, un lenguaje de programación de nivel medio con estructuras típicas de lenguajes de alto nivel. Al compilar el código, nos generará automáticamente un fichero *.hex*, que será posteriormente cargado en el PIC mediante el programador. De no hacer uso de una IDE que nos permita programar en alto nivel, tendríamos que programar los PICs en lenguaje ensamblador y posteriormente convertirlo a lenguaje máquina en un *.hex*, complicando esta tarea de manera innecesaria.

2.5.1 I2C

A pesar de que tendremos PICs trabajando en distintos modos del I2C, probar su funcionamiento lo podemos hacer prácticamente igual en ambos. El compilador que usaremos será el XC8 versión 2.32. Además, a la hora de crear un proyecto, MPLAB nos deja escoger el PIC al que va a estar destinado, configurando así de forma automática todos los parámetros de dirección de memoria de programa y datos.

También se configura para reconocer los nombres de los registros que el PIC en cuestión usa. En la Figura 2.13 podemos ver la información del proyecto para el PIC16F15313 que, en este caso, hemos utilizado para simular el funcionamiento del semáforo.

Los ficheros más destacados son:

- *device_config*: los bits de los registros de configuración del PIC están especificados en este fichero (oscilador externo, programación a bajo voltaje, el *master clear*…).
- *i2c1_slave*: todos los bits y registros de configuración del periférico *MSSP* para el I2C están contenidos aquí, así como el tratamiento de la interrupción por I2C (es la interrupción de la que hablábamos en la descripción de pasos anterior, generada por el *flag* llamado *SSP1IF* en los diagramas de la Figura 2.11 y 2.12).
- *interrupt_manager*: es el gestor general de interrupciones (cuando una interrupción es generada, decide qué *handler* se ha de ejecutar en función del origen de la interrupción).
- *mcc*: contiene las funciones de inicialización del PIC, el oscilador y de los periféricos disponibles (los que aparecen en los diagramas de bloques de la Figura 11 y 12).
- *pin_manager*: se encarga de la inicialización y configuración de los pines del PIC (si los pines son de entrada o salida, si funcionarán como analógico o digital…).

- *main*: el bucle principal que mantiene actualizado el registro que esperamos cambiar cada vez que nos llega información por el I2C (el registro es la salida del *PORTA*, encargado de encender o apagar las luces del semáforo).

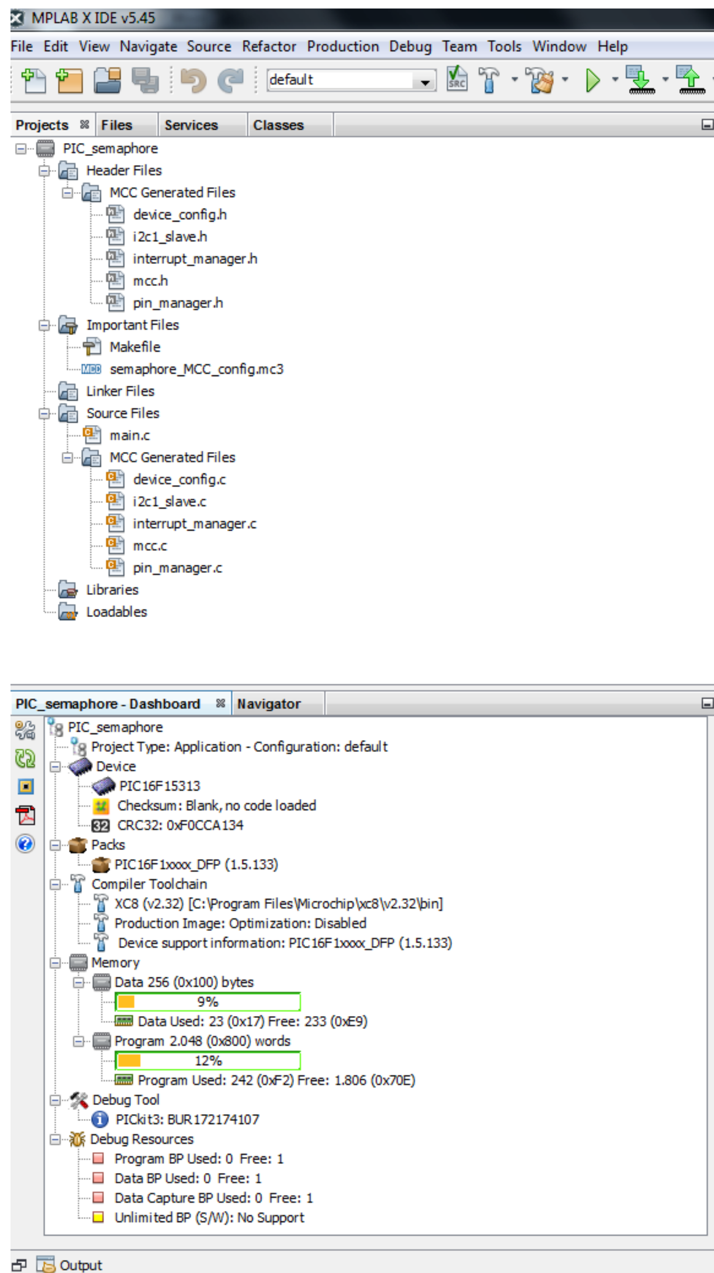


Figura 2.13: Árbol de ficheros y características del proyecto en MPLAB X IDE.

En el *dashboard* podemos ver información de la configuración proyecto, donde me gustaría destacar que ofrece la posibilidad de monitorear la memoria tanto de datos como de programa para el PIC escogido.

En la pestaña de *Navigator* tenemos un árbol con las diferentes estructuras y funciones que pudiera tener el fichero abierto en ese momento, como cualquier otra IDE de programación.

Por lo general, el camino a seguir para inicializar un PIC es común para todos los modelos, es decir: configurar e inicializar. La Figura 2.14 muestra parte del *main* del proyecto usado para el test de los PIC y consta de dos partes:

- *SYSTEM_Initialize*: es la función que asigna los registros de configuración principales del PIC (que a su vez, contiene funciones de algunos de los ficheros mencionados antes: *device_config*, *pin_manager*…).
- *INTERRUPT_Global/PerihperialInterruptEnable/Disable*: son las funciones que habilitan o deshabilitan las interrupciones de carácter global y/o provocadas por periféricos (por ejemplo, las provocadas por el I2C).

```
#include <xc.h>
#include <pic16f15313.h>
#include "mcc_generated_files/mcc.h"

/*
                         Main application
 */
void main(void)
{
    // initialize the device
    SYSTEM_Initialize();

    // When using interrupts, you need to set the Global and Peripheral Interrupt Enable bits
    // Use the following macros to:

    // Enable the Global Interrupts
    INTERRUPT_GlobalInterruptEnable();

    // Enable the Peripheral Interrupts
    INTERRUPT_PeripheralInterruptEnable();

    // Disable the Global Interrupts
    //INTERRUPT_GlobalInterruptDisable();

    // Disable the Peripheral Interrupts
    //INTERRUPT_PeripheralInterruptDisable();
```

Figura 2.14: Parte del *main* de un proyecto en MPLAB X IDE.

La función *SYSTEM_Initialize*, así como un ejemplo de las funciones a las que esta llama, tendría el aspecto que muestra la Figura 2.15. Podemos observar en la función *OSCILLATOR_Initialize* un ejemplo de a qué me refería con los términos de "asignar registros de configuración". Cada uno de los bits de estos registros de un byte es un parámetro de configuración del PIC, por lo que afecta directamente a su comportamiento.

Cabe mencionar que esto es una regla general ya que en la práctica no siempre se utilizan todos los bits, y de hecho, a menudo sucede que algunos registros tienen bits que el PIC ignora porque no tienen una funcionalidad implementada. Por ejemplo, el registro *ANSELA* establece qué pines se comportarán de forma analógica y que pines lo harán de forma digital. Sin embargo, el registro tiene más bits que pines disponibles tiene el PIC, por lo que habrá algunos bits cuyo valor será ignorado. Este comportamiento lo podemos ver en la Figura 2.16, donde los bits 3, 6 y 7 no son tenidos en cuenta.

```c
#include "mcc.h"


void SYSTEM_Initialize(void)
{
    PMD_Initialize();
    I2C1_Initialize();
    PIN_MANAGER_Initialize();
    OSCILLATOR_Initialize();
}

void OSCILLATOR_Initialize(void)
{
    // NOSC HFINTOSC; NDIV 4;
    OSCCON1 = 0x62;
    // CSWHOLD may proceed;
    OSCCON3 = 0x00;
    // MFOEN disabled; LFOEN disabled; ADOEN disabled; EXTOEN disabled; HFOEN disabled;
    OSCEN = 0x00;
    // HFFRQ 4_MHz;
    OSCFRQ = 0x02;
    // MFOR not ready;
    OSCSTAT = 0x00;
    // HFTUN 0;
    OSCTUNE = 0x00;
}
```

Figura 2.15: Función *SYSTEM_Initialize* y ejemplo de función de configuración.

REGISTER 14-4: ANSELA: PORTA ANALOG SELECT REGISTER

U-0	U-0	R/W-1/1	R/W-1/1	U-0	R/W-1/1	R/W-1/1	R/W-1/1
—	—	ANSA5	ANSA4	—	ANSA2	ANSA1	ANSA0
bit 7							bit 0

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
u = Bit is unchanged	x = Bit is unknown	-n/n = Value at POR and BOR/Value at all other Resets
'1' = Bit is set	'0' = Bit is cleared	

bit 7-6 **Unimplemented**: Read as '0'

bit 5-4 **ANSA<5:4>**: Analog Select between Analog or Digital Function on pins RA<5:4>, respectively
1 =Analog input. Pin is assigned as analog input(1). Digital input buffer disabled.
0 = Digital I/O. Pin is assigned to port or digital special function.

bit 3 **Unimplemented**: Read as '0'

bit 2-0 **ANSA<2:0>**: Analog Select between Analog or Digital Function on pins RA<2:0>, respectively
1 = Analog input. Pin is assigned as analog input(1). Digital input buffer disabled.
0 = Digital I/O. Pin is assigned to port or digital special function.

Note 1: When setting a pin to an analog input, the corresponding TRIS bit must be set to Input mode in order to allow external control of the voltage on the pin.

Figura 2.16: Estructura de los bits que componen el registro *ANSELA* [13].

Tras contemplar la inicialización del PIC, le sigue habilitar las interrupciones. Esto implica la creación de un gestor general que, en función de qué ha generado la interrupción, llame al *handler* adecuado. Este gestor tendría el aspecto que aparece en la Figura 2.17.

```
#include "interrupt_manager.h"
#include "mcc.h"

void __interrupt() INTERRUPT_InterruptManager (void)
{
    // interrupt handler
    if(INTCONbits.PEIE == 1)
    {
        if(PIE3bits.BCL1IE == 1 && PIR3bits.BCL1IF == 1)
        {
            MSSP1_InterruptHandler();
        }
        else if(PIE3bits.SSP1IE == 1 && PIR3bits.SSP1IF == 1)
        {
            MSSP1_InterruptHandler();
        }
        else
        {
            //Unhandled Interrupt
        }
    }
    else
    {
        //Unhandled Interrupt
    }
}
```

Figura 2.17: Gestor de interrupciones general.

Como en este momento solo nos interesa detectar la interrupción por I2C (recordemos, se encuentra en el módulo *MSSP*), nos basta con gestionar los *flags* que nos alertan de esa interrupción en concreto. En el caso de la Figura 2.17 detectamos dos *flags* diferentes, que son respectivamente: detección de colisión en el bus, y detección de petición de comunicación.

Si se genera una interrupción y el gestor detecta que es debido a esos *flags*, llamarán al *handler* en cuestión, que actuará en consecuencia y limpiará el *flag*. De esta forma el *flag* puede volver a activarse para alertar nuevamente de otra interrupción, tal como hemos descrito previamente en los pasos de una comunicación I2C típica.

Por último, la Figura 2.18 muestra un pequeño código de ejemplo para probar el comportamiento del PIC en el I2C. Los registros que aparecen hacen la siguiente función:

- *ANSELA*: define si el pin se comportará de forma analógica o digital.
- *TRISA*: define si el pin es de *input* u *output*.
- *PORTA*: asigna un voltaje al pin que determinará su valor (los voltajes son: *Voltage Input High* y *Voltage Input Low*, es decir 1 o 0 respectivamente).

```
ANSELA = 0x00;   // 1->analog / 0->digital
TRISA = 0x06;    // 1->input / 0->output
PORTA = 0x00;    // 1->VIH / 0->VIL

I2C1_Open();

while (1){                  // Deberia de tener este aspecto encendido:
    PORTA = read1Byte;      // 0b00110000, donde los bits a 1 son RA5 y RA4
}
```

Figura 2.18: Código para probar si el PIC se comunica por I2C [15].

Este código está constantemente actualizando los valores de salida de los pines RA4 y RA5, en función de la información que le llega por I2C. Estos pines están conectados a un LED cada uno, de forma que cuando su bit identificador en el registro *PORTA* es 1, el LED se enciende.

Cuando el *handler* recibe la interrupción, lee los datos que le han llegado y procesa la respuesta en *read1Byte*. Por ejemplo, como sale en el comentario de la Figura 2.18: si recibe por I2C que *read1Byte* toma el valor de 0b00110000 en binario, significa que el *PORTA* asignará un 1 a los pines RA4 y RA5, por lo que los LEDs se encenderán [18].

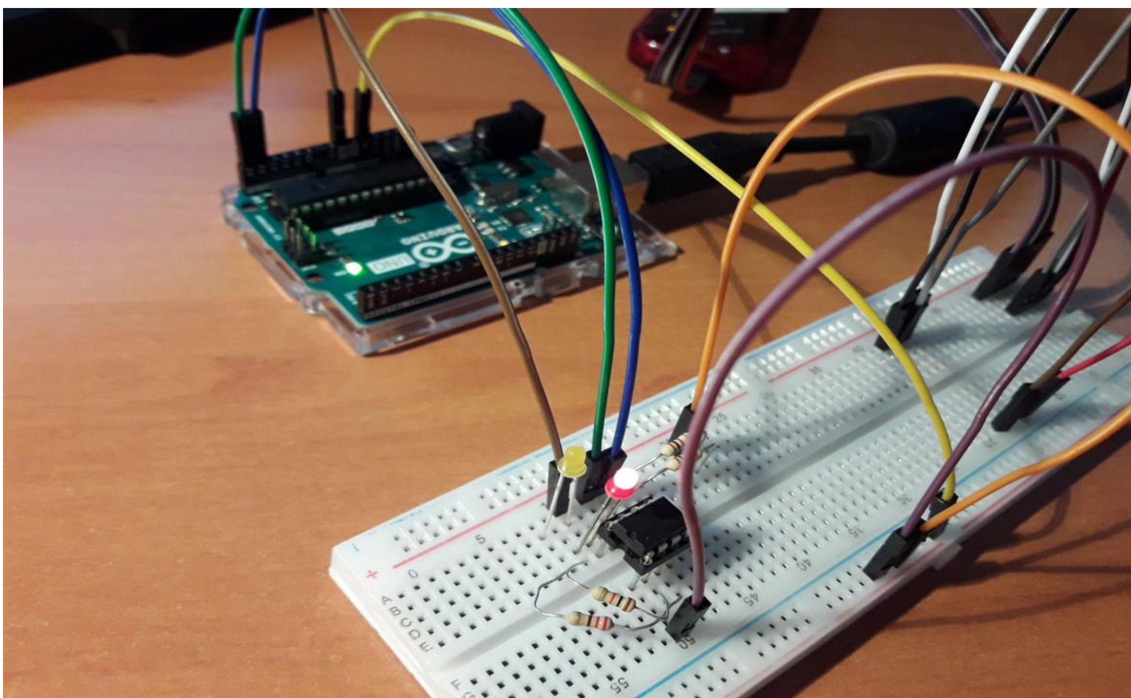


Figura 2.19: Test del código anterior: *read1Byte* con valor de 0b00100000 (RA5 = 1).

3 Diseño del sistema y los módulos

Ahora que tenemos unos PICs *slaves* funcionales, para poder probarlos en un escenario real tenemos que comenzar con el diseño del sistema y de los módulos que los PICs controlarán. A su vez, utilizaremos una placa Arduino Uno (Figura 3.1) que será el *master* del sistema [17]. Esta placa es ideal para esta función, debido al enorme soporte del que dispone, y la relativa sencillez que supone a la hora de trabajar con ella. Además, la capacidad de poder observar por pantalla las comunicaciones gracias al *monitor serie* facilitan razonablemente las tareas de *debug*. En este capítulo veremos el diseño del sistema y los módulos, así como las peculiaridades que pueda haber, tanto *hardware* como *software*.

Figura 3.1: Arduino Uno Rev. 3 [3].

Recordemos que contemplamos cuatro tipos de módulos: módulo de cambio de vía, módulo de control de semáforos, módulo de control de velocidad y módulo de detección de presencia. Dadas las limitaciones de la maqueta, supondremos que tendremos dos semáforos, dos cambios de vía, un tren que controlar, y un total de cinco sensores a repartir en el tramo de vía.

Por tanto, saldrán un total de seis circuitos: dos para los semáforos, dos para los cambios de vía, uno para el tren y otro más para los sensores de detección. En la Figura 3.2 podemos observar el boceto de un primer acercamiento a lo que queremos realizar. En el boceto solo aparece un módulo de cada, pero ha servido para hacerse una idea de la cantidad de conexiones que habrán. La colocación de estos será sobre la base, pero para facilitar la comprensión se colocaron fuera en el boceto, cuyas siglas significan:

- UNO: es la placa Arduino y estará conectada al resto de módulos, y aunque en el boceto no aparezca, también al bluetooth.
- TRAFO: es el transformador que suministra energía a la vía. Tiene una salida de corriente continua (DC) y otra de corriente alterna (AC).
- MCV: es el control de velocidad del tren, que debe dejar pasar corriente en función a la velocidad que se indique en la aplicación. Para ello se hará uso del *PWM* (siglas de *Pulse With Modulation*) para controlar la cantidad de energía que se manda a una carga. Estará conectado a la salida DC del TRAFO, ya que el tren funciona con DC.

- MCCV: es el cambio de vía, que estará directamente conectado a un solenoide que funciona con DC. Como no podemos usar la misma salida del TRAFO que usamos en el MCV, ya que afectaría directamente sobre el tren, este estará conectado a la salida AC del TRAFO. Esto complicará el diseño del circuito, pero esto lo veremos más adelante.
- MCS: es el control del semáforo, que ha de cambiar la luz en función de lo que se indique en la aplicación. La propia energía que da la Arduino es suficiente para hacerlo funcionar.
- MDP: es el detector de presencia, que debe detectar cuando pasa el tren por encima, y enviar esta respuesta. El hecho de que sea un solo módulo para cinco sensores es arbitrario, podría haberse realizado de igual forma con dos módulos con tres y dos sensores respectivamente. O incluso se podría añadir otro módulo entero con otros cinco y tener un total de diez sensores. La forma de proceder sería la misma, ya que cada módulo se identifica con una zona única.

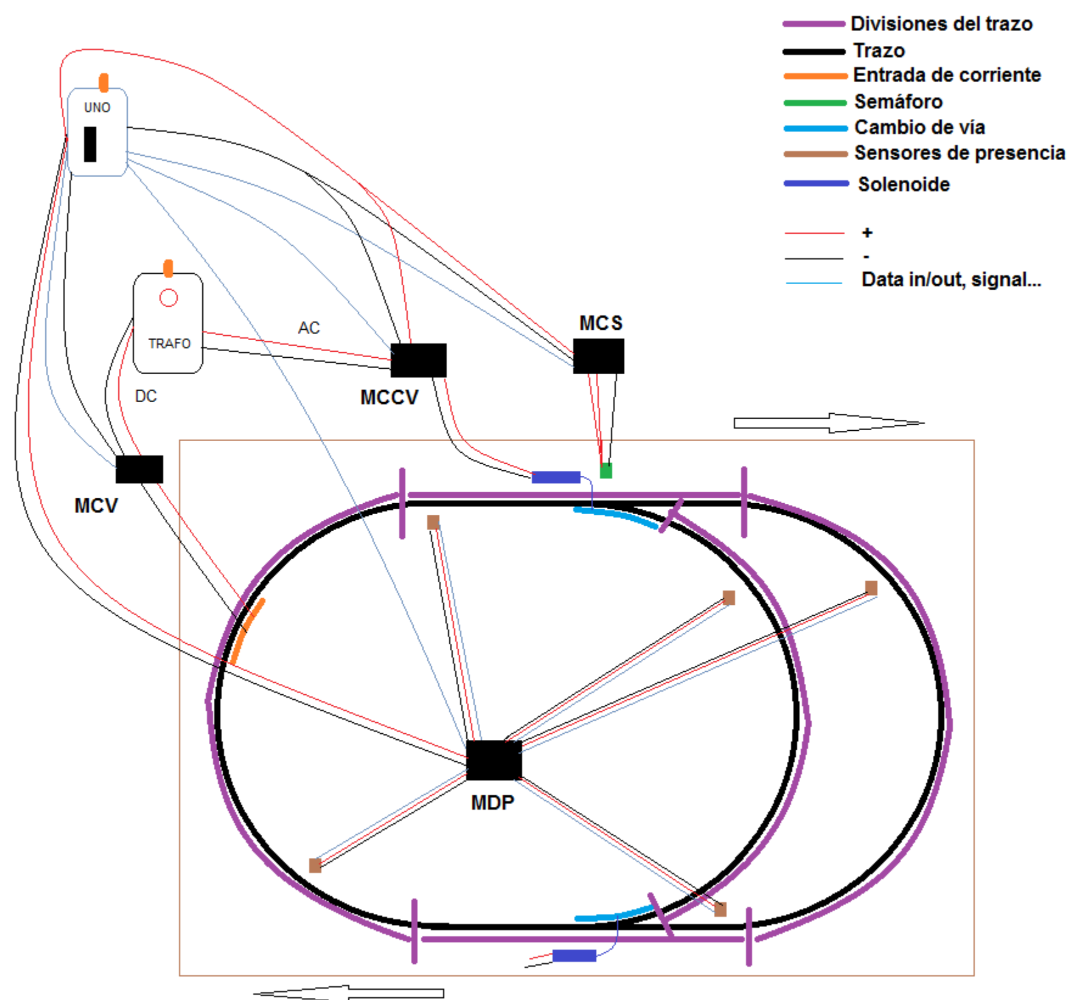


Figura 3.2: Boceto de la idea conceptual del sistema.

Omitiremos algunos pasos de la realización de la base ya que no tienen que ver con el objetivo de este proyecto, pero si que se añadirá alguna imagen del resultado. Los esquemáticos aportados en los siguientes apartados han sido realizados con *EAGLE*, que es ideal para los diseños de este tipo, ya que gracias a sus librerías podemos encontrar todo tipo de componentes y esquemáticos ya diseñados de muchos microcontroladores.

3.1 MCV – Módulo de control de velocidad

Como ya hemos dicho, este módulo nos ha de permitir variar la velocidad del tren por medio de *software*. Esto es, por medio de este módulo, tenemos que poder controlar a distancia la velocidad sin tocar el transformador para nada, únicamente mandando valores de velocidad por *software*. Para conseguir esto, hay que pensar en una forma de trasladar este concepto a *hardware*, y así poder hacer un circuito que cumpla con esa función. El concepto en el que se va a basar el circuito para llevar esto a cabo, es el uso de transistores *mosfet*.

El transistor *mosfet* debe su nombre al tipo de tecnología que usa y al tipo de transistor que es (*MOS* => *Metal-Oxide-Semidonductor* y *FET* => *Field-Effect-transistor*), y es utilizado para amplificar o conmutar señales electrónicas. Dicho de otra forma, en los *mosfet* se utiliza un campo eléctrico para controlar su conducción, y eso es justo lo que necesitamos [19]. En la Figura 3.3 podemos ver un esquemático de un circuito que cumpliría con los requisitos.

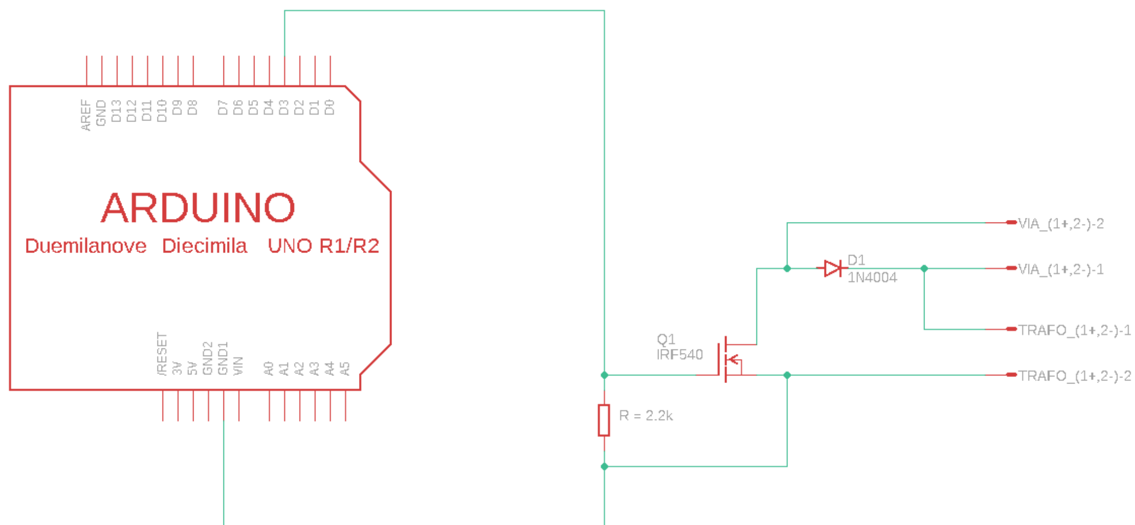


Figura 3.3: Esquemático del circuito para el control de velocidad.

Tenemos el *gate* de un transistor conectado al pin D3 de la Arduino para controlarlo. Esta decisión no ha sido arbitraria ya que no todos los pines de la Arduino soportan el *PWM*, D3 sin embargo si que es capaz de hacer uso de esta característica. El transistor que he utilizado en el diseño final es el modelo *P24NF10* (Figura 3.4), que en lugar de ser un *mosfet* es un *power mosfet* [21]. Este transistor ha sido diseñado para minimizar la resistencia interna, por lo que es adecuado para interruptores de alta frecuencia, así como aplicaciones con bajos requisitos para la *gate*.

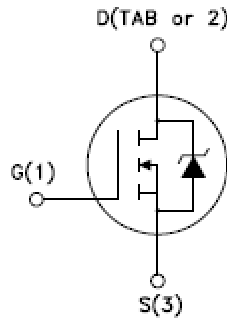


Figura 3.4: Esquemático interno del transistor P24NF10 [21].

En el esquemático del circuito vemos también la resistencia que está colocada ahí para la señal que manda la Arduino al transistor, así como un diodo para protegerlo del transformador.

Como hemos dicho, el *PWM* es lo que utilizaremos para mandar la señal al *gate* [16]. Esta señal requiere que escribamos un valor analógico en el pin D3 de la Arduino, tal como vemos en la Figura 3.5, donde *SPEED* es un *define* al pin D3.

```
analogWrite(SPEED, PWM_spd);
PWM_spd += incremento;
if (PWM_spd > 254) incremento*(-1);
if (PWM_spd < 1) incremento*(-1);
delay(10);
```

Figura 3.5: Ejemplo de uso del *PWM* [16].

Es un código muy simple, que va dentro de un bucle, que va acumulando *PWM_spd* con *incremento* y hace *analogWrite* por el pin D3. Cuando el valor llega 255 o a 0, invierte el incremento. El rango de valores que puede tomar el *PWM* va de 0 a 255, es decir, un byte. Y el funcionamiento es muy simple:

- Si *PWM_spd* = 0: el transistor actúa como un interruptor abierto y no circula corriente por el circuito.
- Si *PWM_spd* = 255: el transistor actúa como un interruptor cerrado y la corriente circula por el circuito.
- Si *PWM_spd* está comprendido entre 0 y 255: la corriente circulará de forma proporcional al valor de la señal. En otras palabras, en función de esta señal, el tren irá más rápido o menos.

El código anterior hace que el tren alcance progresivamente su velocidad máxima, y tras alanzarla, la reduzca progresivamente. Cuando llega a cero y se detiene, el ciclo comienza de nuevo. Este código es solo un ejemplo; a la hora de insertar el módulo en el sistema, lo que en el código era *PWM_spd* ahora será el valor que le digamos desde la aplicación del móvil, y el cálculo de *incremento* desaparecerá.

3.2 MCCV – Módulo de control de cambio de vía

Para el módulo del cambio de vía tenemos una complicación añadida: aunque la idea detrás del circuito es similar a la idea del circuito para el control de velocidad, el solenoide funciona con DC, mientras que nuestro transformador nos da AC. El objetivo es controlar el cambio de vía por *software*, y para ello también haremos uso de transistores *mosfet* (el mismo modelo), solo que esta vez no habrá termino medio: o el transistor deja pasar corriente al 100% (circuito cerrado), o no deja pasar corriente (circuito abierto).

Para solucionar el problema de la AC, tendremos que hacer uso del concepto de rectificación de onda completa [20], que podemos ver en la Figura 3.6.

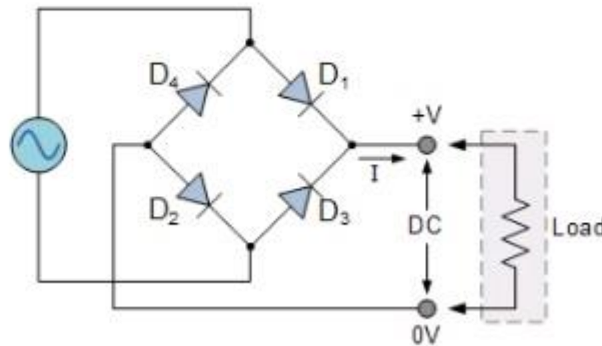


Figura 3.6: Circuito con un puente rectificador de onda completa [20].

Este circuito nos permite convertir una onda AC en DC. Los cuatro diodos etiquetados D1, D2, D3 y D 4 están dispuestos en “pares en serie” con solo dos diodos que conducen corriente cada medio ciclo. Durante el semiciclo positivo del suministro, los diodos D1 y D2 se conducen en serie, mientras que los diodos D3 y D4 tienen polarización inversa y la corriente fluye a través de la carga (*load*) [20]. En la Figura 3.7 y 3.8 se muestran los semiciclos positivo y negativo.

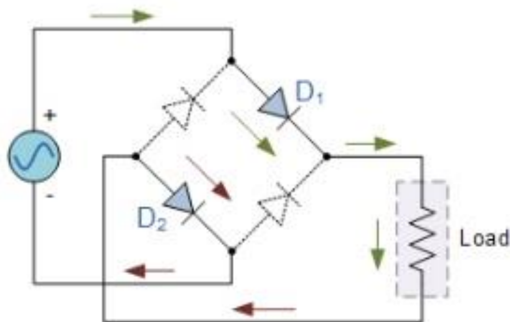


Figura 3.7: Semiciclo positivo [20].

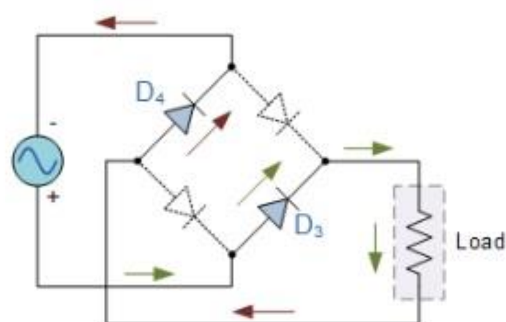


Figura 3.8: Semiciclo negativo [20].

Sin embargo, como vemos en la Figura 3.9, la salida que obtenemos del puente rectificador es ondulada y no sirve para alimentar un solenoide que va con DC.

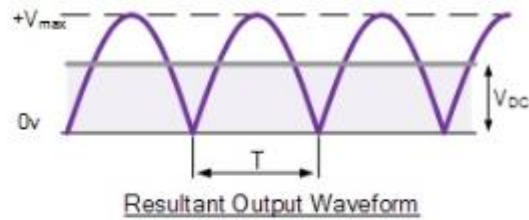


Figura 3.9: Onda de salida resultante rectificada [20].

Para poder aprovechar esta onda rectificada, necesitaremos hacer uso de un condensador de suavizado, que convertirá la salida ondulada de onda completa del rectificador en un voltaje de salida DC más suave [20]. En la Figura 3.10 tendríamos el circuito de rectificación completo, así como la onda de salida rectificada esperada.

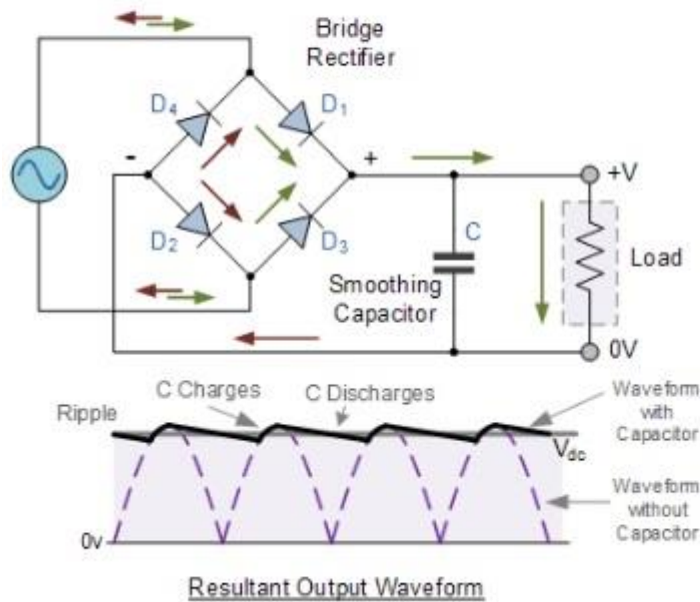


Figura 3.10: Rectificador de onda completa con condensador de suavizado [20].

Este es parte del circuito que necesitamos para el módulo de cambio de vía. Ahora falta añadir un transistor que controle esa corriente de salida, y que la deje pasar cuando nosotros lo pedimos por *software*. En la Figura 3.11 podemos ver un esquemático de un circuito que cumpliría con los requisitos.

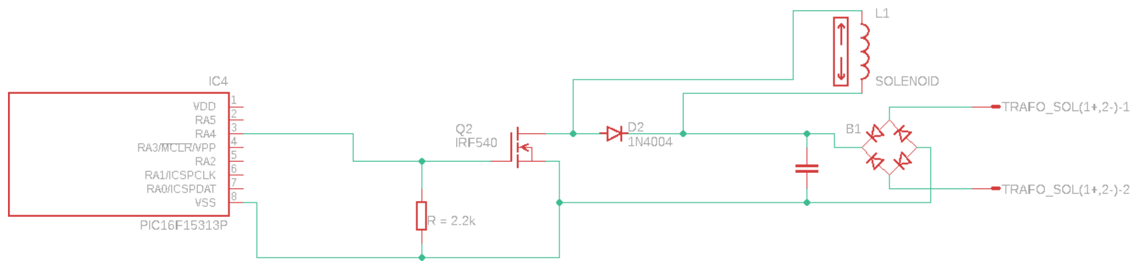


Figura 3.11: Esquemático del circuito para el control de cambio de vía.

Esta vez es el PIC quien controla el *gate* del transistor, que de nuevo está protegido del transformador por un diodo. Si hay que activar el cambio de vía, el PIC pondrá el pin RA4 a *VIH*, y el transistor cerrará el circuito permitiendo la circulación de la corriente y activando el solenoide.

3.3 MCS – Módulo de control de semáforos

Este es el módulo más sencillo de fabricar, dado que no requiere ni el uso de transistores ni el uso del transformador, ya que el PIC es capaz de alimentarlo con la corriente de sus pines. El objetivo es poder controlar las luces del semáforo a distancia, mediante señales que le mandaremos al PIC desde la aplicación. En la Figura 3.12 podemos ver un esquemático de un circuito que cumpliría con los requisitos.

Figura 3.12: Esquemático del circuito para el control de semáforos.

Y de hecho, el test sobre protoboard que aparece en la Figura 2.19 es exactamente este diseño con las conexiones I2C añadidas. Aunque en el circuito final se usará un semáforo en miniatura de verdad, y no los LEDs que aparecen en ese test.

Figura 3.13: Semáforo utilizado en la maqueta, controlado por el módulo.

3.4 MDP – Módulo de detección de posición

Por último, el circuito para la detección de posición, que será el más grande de todos. A el se conectarán los cinco sensores infrarrojos que estarán repartidos por el tramo de vía. Este módulo nos debe de indicar cual de los cinco sensores es el que se ha activado, significando así que el tren ha pasado por encima de este. Este es el único módulo que en lugar de recibir información, la transmite: el PIC recogerá la señal y la enviará por I2C.

Para asegurarnos de que los valores que el circuito lee son correctos, y evitar falsas señales generadas por el *ruido* de los circuitos electrónicos, será necesario añadir resistencias de *pull-down*. Estas resistencias no tienen nada de especial, pero están colocadas de una forma determinada en el circuito. Su función es establecer un estado lógico en un pin de entrada, o entrada de un circuito lógico, cuando se encuentra en estado de reposo [22].

En este caso, una resistencia de *pull-down* establece un estado *LOW* cuando el pin se encuentra en reposo (justo a la inversa de una resistencia *pull-up*) [22]. Y esto es justo lo que necesitamos, ya que establecerá un estado *LOW* cuando el sensor no detecte nada (en reposo), y así evitaremos dar falsos positivos al PIC. En la Figura 3.14 podemos ver un esquemático de un circuito que cumpliría con los requisitos.

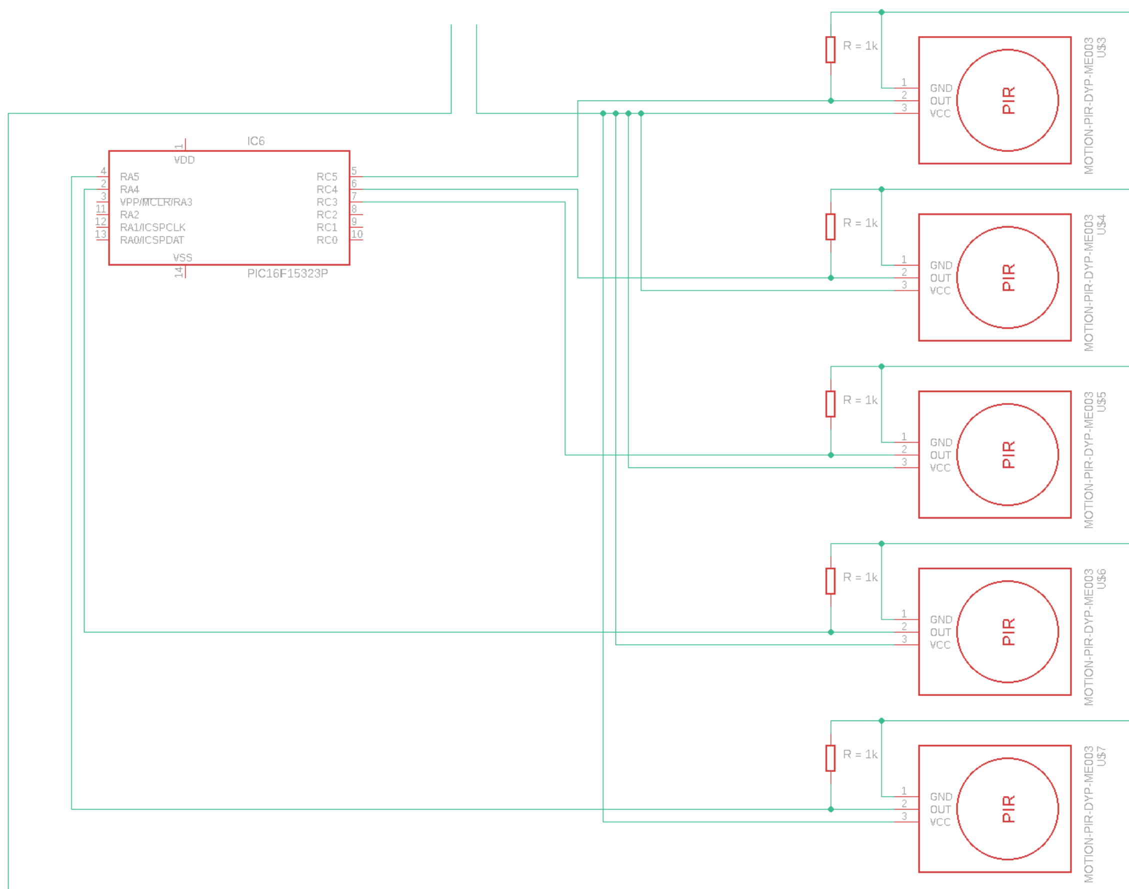


Figura 3.14: Esquemático del circuito para la detección de presencia.

Y es para este módulo que hemos usado el PIC16F15323, para poder tener suficientes pines para los sensores. En el esquemático anterior podemos ver las resistencias de *pull-down* que mencionábamos antes, conectando la salida a tierra. Esta vez es la Arduino la que alimenta los sensores gracias a la salida de 5V que da. En la Figura 3.15 aparece un test del concepto del circuito, con un sensor haciendo uso de la resistencia de *pull-down* y un LED para tener una alerta visual de cuando el sensor detecta presencia.

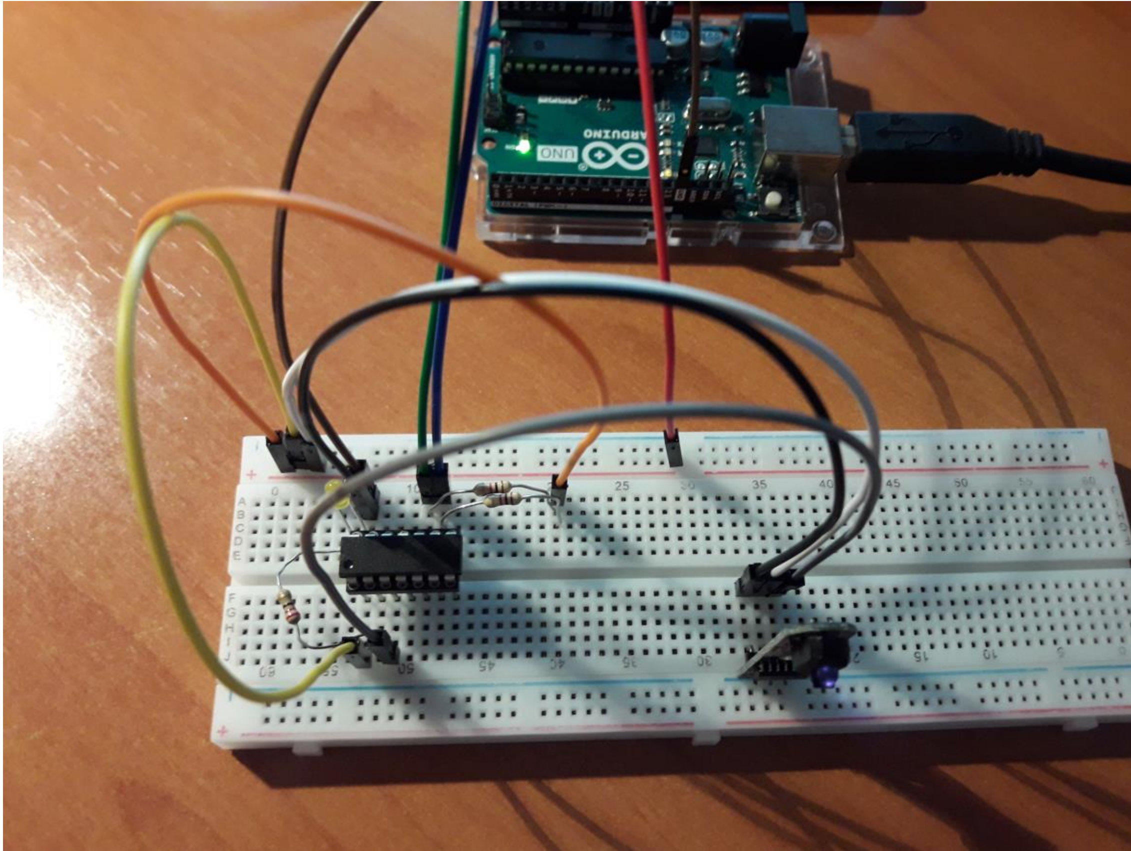


Figura 3.15: Test del sensor de presencia con la resistencia de *pull-down*.

Aunque el LED está ligeramente tapado, podemos ver que está apagado, que es el comportamiento esperado para cuando el sensor no detecta presencia. En este test además, se estaba probando que el PIC comunicase el resultado por I2C (la pareja de cables verde-azul que hacen uso de dos resistencias *pull-up*).

3.5 Sistema con todos los módulos juntos

Ahora que ya tenemos decidido el diseño de los circuitos para cada módulo, hay que integrarlo todo en un mismo sistema. Este diseño será el que usaremos en la demostración, y es el que podemos ver en la Figura 3.16.

Nótese que en el esquemático solo aparece un circuito de cada módulo. Esta decisión es intencionada, ya que añadir más circuitería replicada no aporta información y reduciría la calidad de la imagen. Añadir otro módulo del semáforo, por ejemplo, sería añadir otro circuito idéntico al existente al bus del I2C.

En el esquemático definitivo vemos algunos añadidos más además de los circuitos mencionados en los apartados anteriores. En la parte superior tenemos conectado el módulo de bluetooth (modelo *HC-05*) para la comunicación con la aplicación móvil. Además, ahora todos los módulos ya tienen sus conexiones para energía, así como para tierra, que no estaban presentes en los esquemáticos anteriores.

Por último tenemos los buses de I2C que salen de los pines A4 y A5 de la placa Arduino y conectan todos los circuitos para que se puedan comunicar con la placa. Nótense las dos resistencias de *pull-up* de 1kΩ al inicio del bus, pues son necesarias, ya que la comunicación por I2C no tendría lugar si se omitiesen: Los PICs no podrían generar la condición de *start* ni tampoco transmitir la dirección de con quien se quieren comunicar [23].

Ahora bien, en el caso de usar el bus I2C con una placa Arduino como master, si que es posible omitir estas dos resistencias de 1kΩ, aunque no es recomendable. Esto es debido a que el microcontrolador de la placa Arduino (*ATmega328p*) posee resistencias internas de *pull-up* de 20kΩ que pueden ser habilitadas vía *software*. Si el bus es lo suficientemente pequeño, físicamente hablando, y la comunicación es lo suficientemente lenta, estas resistencias pueden servir [23]. Sin embargo, no es un diseño confiable, por lo que hemos optado por añadir las resistencias discretas de 1kΩ que vemos en el esquemático, para asegurar el correcto funcionamiento.

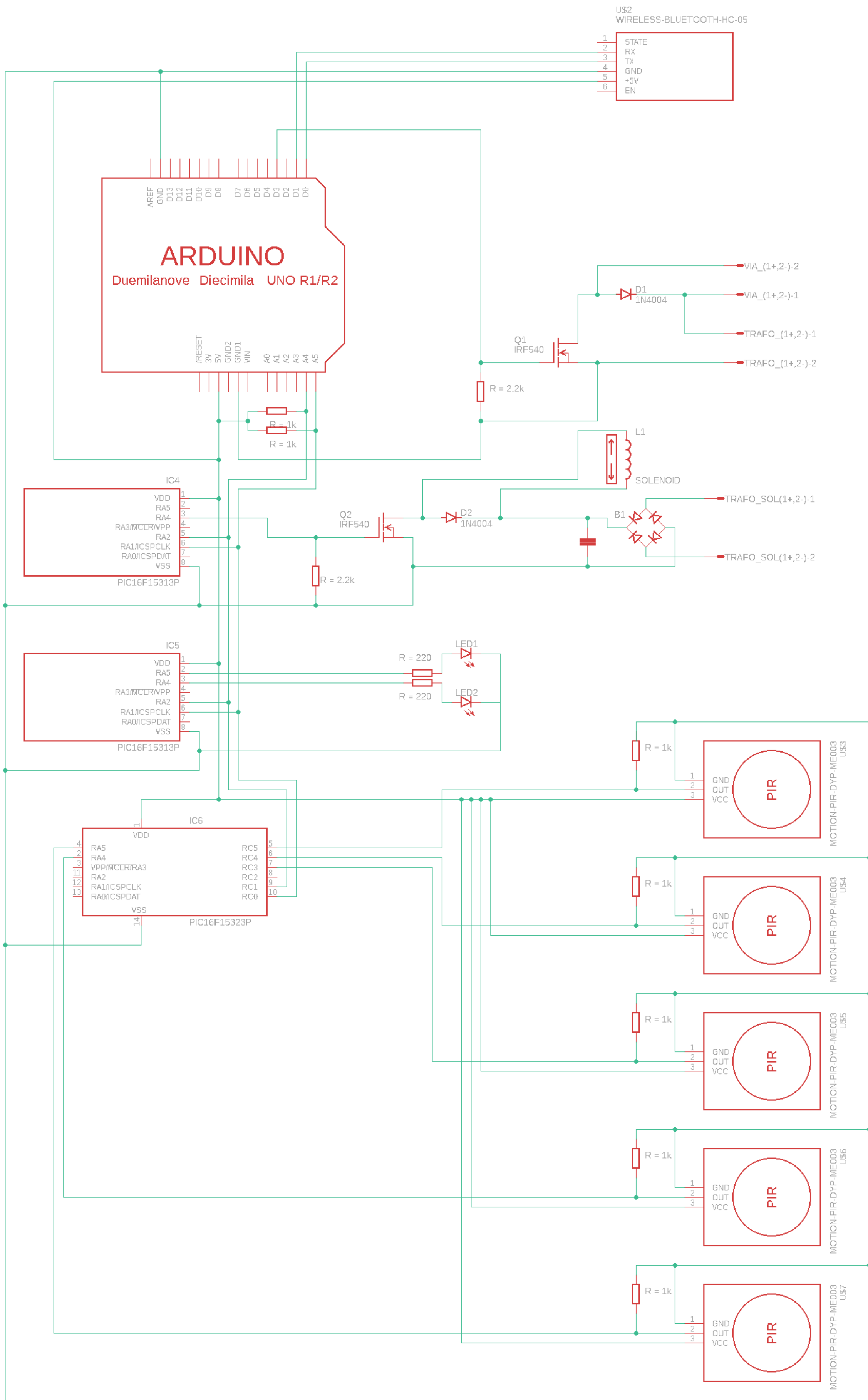


Figura 3.16: Esquemático definitivo del sistema con los módulos integrados.

A continuación se muestran los circuitos resultantes de los esquemáticos anteriores, ya soldados y testeados. En la Figura 3.17 tenemos el circuito del módulo de control de velocidad, donde los cables gris y blanco superiores son el positivo y negativo del transformador, respectivamente. Mientras que los *wires* blanco y gris que aparecen ligeramente a la izquierda son la señal del *PWM* y el *GND* de la Arduino, respectivamente. Los otros dos cables rojo y negro son los bornes positivo y negativo que se conectarán a la vía.

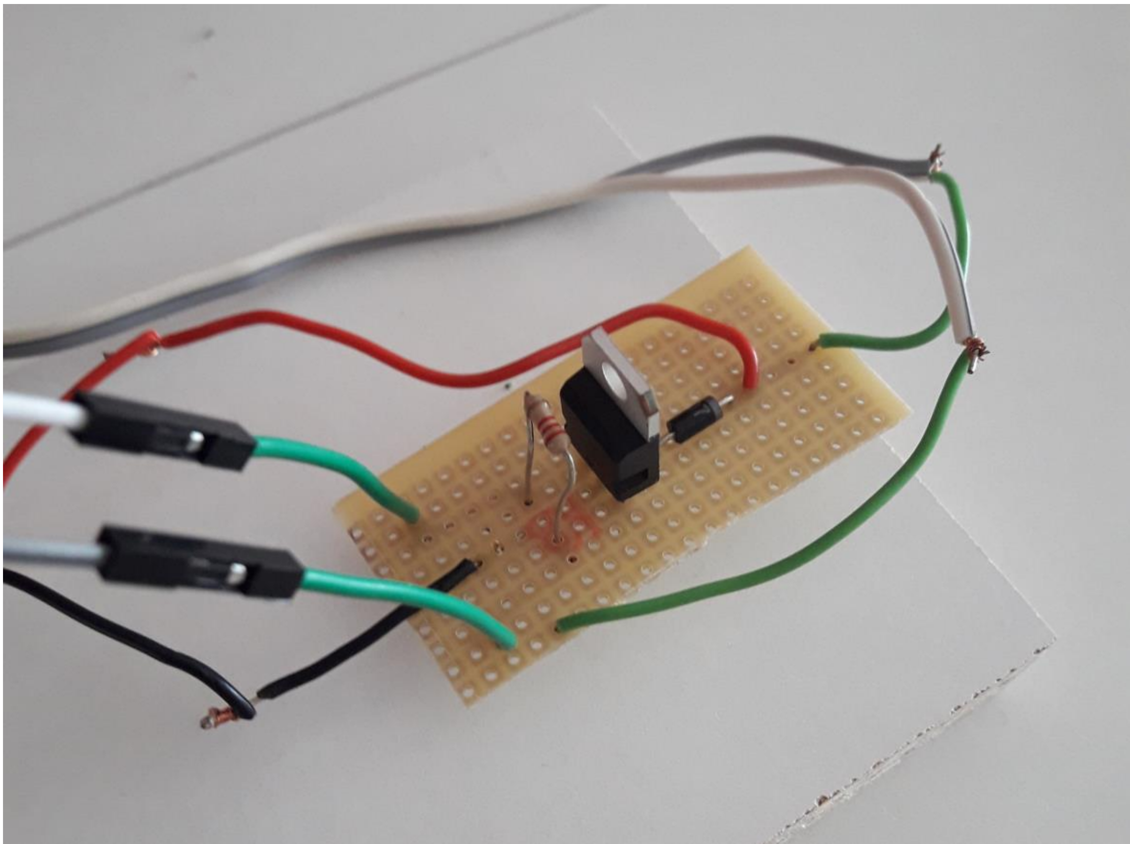


Figura 3.17: Módulo de control de velocidad.

En la Figura 3.18 tenemos uno de los módulos de control de cambio de vía, donde podemos ver el puente rectificador y el condensador de suavizado que mencionamos en el apartado 3.2. Las cuatro conexiones de la izquierda son:

- Negro: 5V
- Naranja y rojo: I2C
- Marrón: *GND*

En el momento de tomar la imagen estaba experimentando con la adición de disipadores al transistor, de ahí que no aparezca el solenoide conectado ni la conexión AC del transformador. No obstante, para facilitar la comprensión, he editado donde irían estas conexiones.

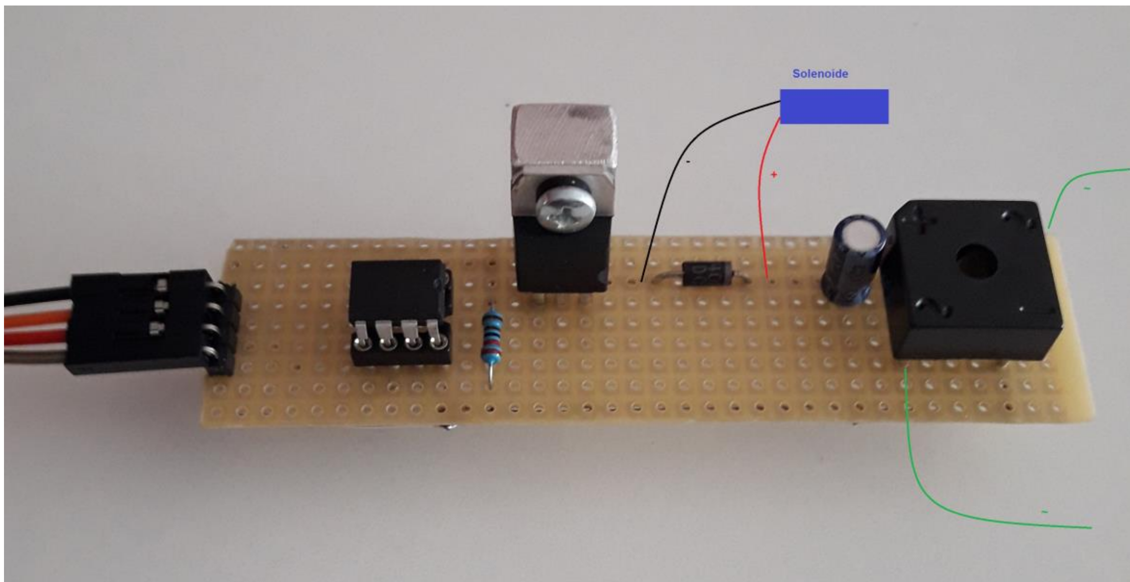


Figura 3.18: Módulo de control de cambio de vía.

Seguidamente, en la Figura 3.19 tenemos uno de los módulos de control de semáforo, donde las conexiones de la derecha son:

- Morado: 5V
- Amarillo: *GND*
- Azul y verde: I2C

Y las conexiones de la parte superior son, de izquierda a derecha: semáforo en verde, semáforo en rojo, y *GND*.

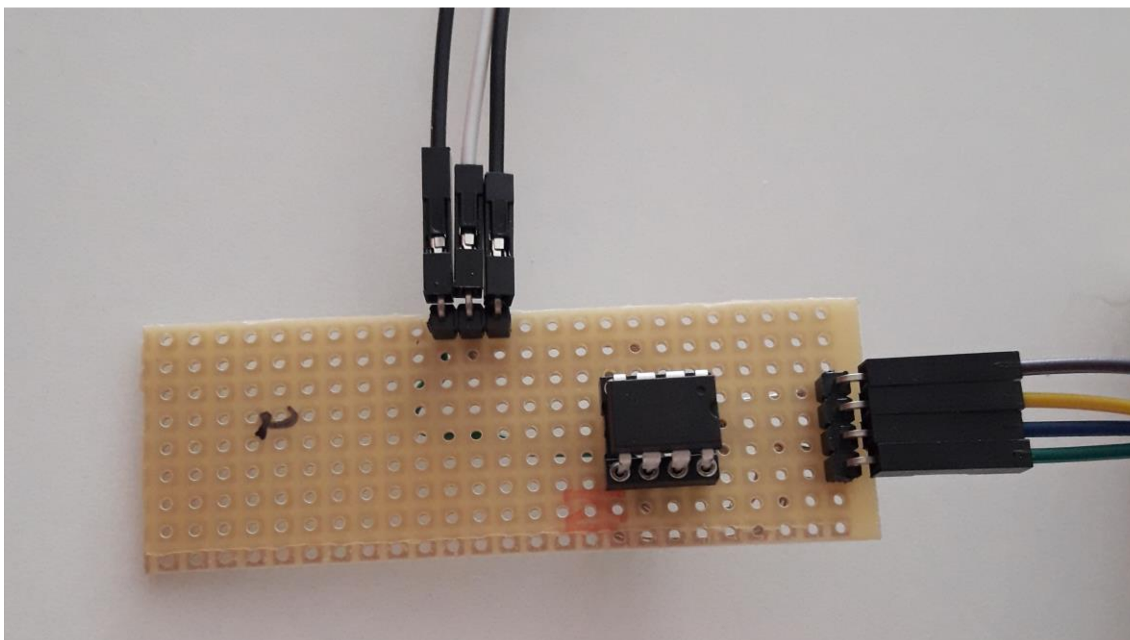


Figura 3.19: Módulo de control del semáforo.

Por último, en la Figura 3.20 tenemos el módulo de detección de posición, el más grande de todos, donde las conexiones de la izquierda (Figura 3.20.a) son:

- Verde y azul: I2C
- Morado: *GND*
- Gris: 5V

Y las conexiones de la parte inferior pertenecen a los 5 sensores, y son todas idénticas entre sí, de izquierda a derecha: señal de respuesta, 5V, y *GND*. Las conexiones a 5V y *GND* de los sensores aparecen con cableado rojo y negro respectivamente en la vista inferior del circuito. El circuito fue diseñado para que requiriese la menor cantidad posible de cableado, pero fue necesario el uso de estos cables con revestimiento (resisten el calor de aplicar la soldadura) para evitar puentes indeseados entre pistas.

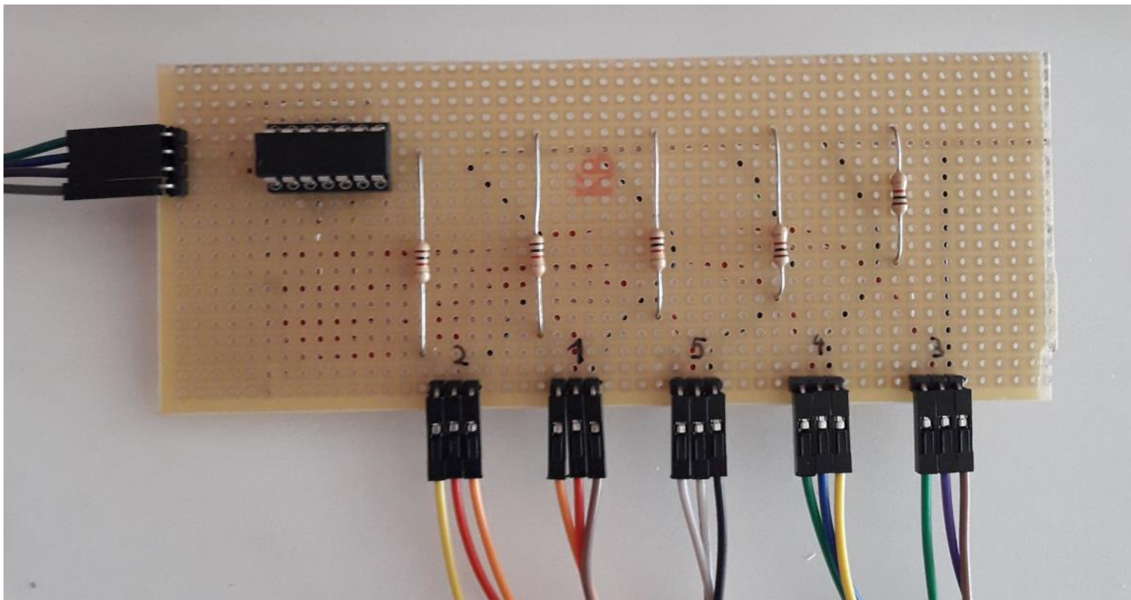


Figura 3.20.a: Módulo de detección de posición (vista superior).

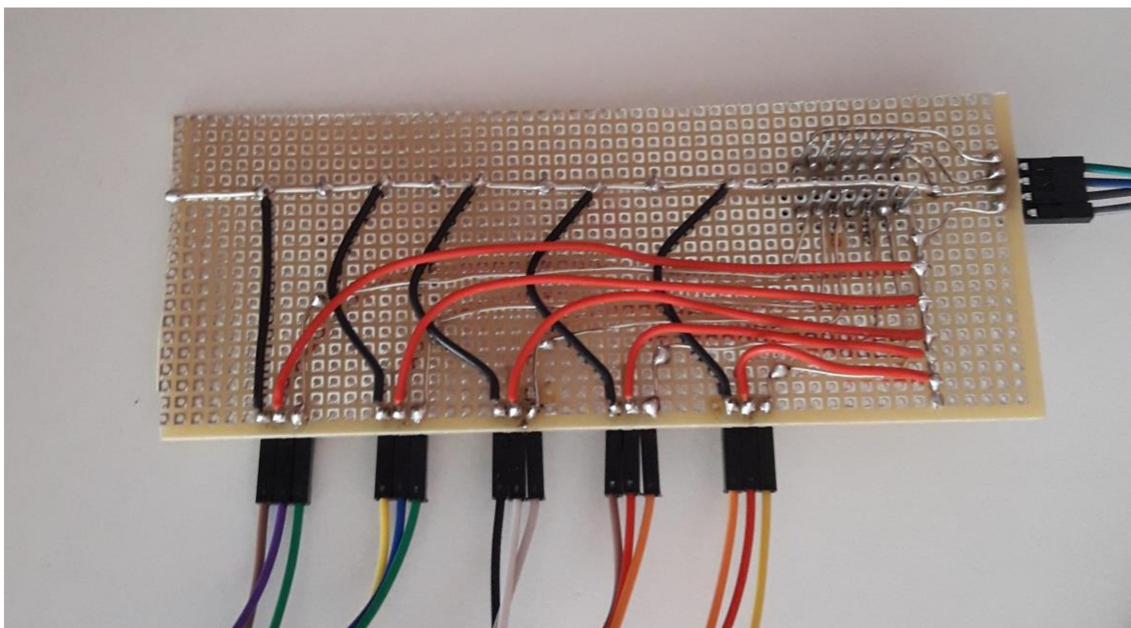


Figura 3.20.b: Módulo de detección de posición (vista inferior).

4 Diseño de la aplicación móvil

Una vez tenemos los circuitos construidos con los PICs programados, testeado su funcionamiento, y todas las conexiones realizadas, pasamos a crear una aplicación para poder ver y controlar todo el sistema desde el móvil.

La aplicación la crearemos en *App Inventor*, que es un entorno de desarrollo de *software* para dispositivos *Android* creado por *Google Labs*. Este entorno es un servicio web basado en la nube que permite almacenar y realizar el seguimiento de proyectos que creemos, y solo requiere un navegador web para utilizarlo [24]. Este entorno trabaja con dos herramientas para crear aplicaciones, que son las siguientes:

- Diseñador: es la herramienta que permite crear la interfaz de usuario, es decir, la parte visible de la aplicación (por ejemplo: los botones o las imágenes).
- Editor de bloques: es la herramienta que define la lógica de los componentes de la aplicación utilizados en el Diseñador.

Figura 4.1: Interfaz de usuario de la aplicación en el Diseñador [24].

En la Figura 4.1 podemos ver el aspecto de la interfaz de usuario de la aplicación, en la que distinguimos varios apartados:

- Botones Conectar y Desconectar: para poder controlar la maqueta, antes tenemos que conectarnos a ella. Y esto es lo que hace el botón Conectar: abre un menú con una lista de los dispositivos bluetooth disponibles a los que podemos conectarnos. Aquí aparecerá nuestro sistema, al que debemos pulsar para conectarnos. El botón Desconectar nos desconecta del bluetooth de la maqueta, para cuando hemos acabado de utilizar la aplicación.
- Deslizador *Speed*: esta es la opción que nos permitirá controlar la velocidad. Es un deslizador que va de 0 a 255, que recordemos, es el rango de valores con el que controlamos la velocidad mediante el *PWM*.
- Interruptores Sem 1 y Sem 2: estos interruptores nos permiten controlar el estado del semáforo, y por defecto comienzan en verde (GO). Si pulsamos el interruptor, su color verde se volverá rojo, y la palabra <GO> cambiará por <STOP>. Se utilizan los mismos colores que los LEDs del semáforo, para así relacionarlos fácilmente. La Figura 4.2 muestra un ejemplo de uso.
- Interruptor Switch Track 1: este interruptor activa el tramo de vía alternativo del circuito, activando los dos solenoides a la vez. En general, cuando un tramo de vía se divide en dos en un punto, estos acaban convergiendo de nuevo en otro punto (véase el dibujo del circuito que aparece en la Figura 4.1), a menos que sea una vía muerta (algo poco común en maquetas pequeñas). De modo que he considerado oportuno diseñar la aplicación con esta idea de activar los dos a la vez, ya que activar solo uno hará que el tren descarrile en el siguiente.
- *Section Map*: este último apartado no se ha de pulsar ni nada, ya que es un mapa donde se van iluminando las secciones por las que el tren va pasando, y se van apagando conforme el tren las deja atrás. Solo puede haber una sección iluminada a la vez, puesto que solo hay un tren que detectar y este no puede estar en dos sitios a la vez. Como hemos utilizado cinco sensores, este circuito está dividido en 5 secciones, que son las tres curvas y las dos rectas que vemos en el dibujo del circuito que aparece en la Figura 4.1.

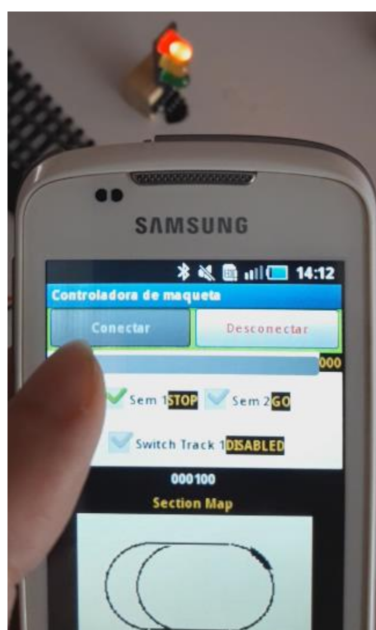


Figura 4.2: Test de la aplicación para controlar el semáforo 1 y ponerlo en rojo (STOP).

En cuanto a la lógica de la aplicación, tenemos que movernos a la herramienta del Editor de bloques. La idea del funcionamiento del Editor de bloques es bastante simple: escogemos a qué elemento le queremos aplicar una lógica, y luego escogemos que lógica le asignamos. Por ejemplo, para el caso del interruptor de cambio de vía: nos interesa que cuando pulsamos el interruptor para cambiar a otra vía, enviemos un valor; mientras que si volvemos a pulsar para volver a dejar la vía como estaba, enviemos otro valor diferente. Esta lógica la podemos ver plasmada en la Figura 4.3 haciendo uso del Editor de bloques.

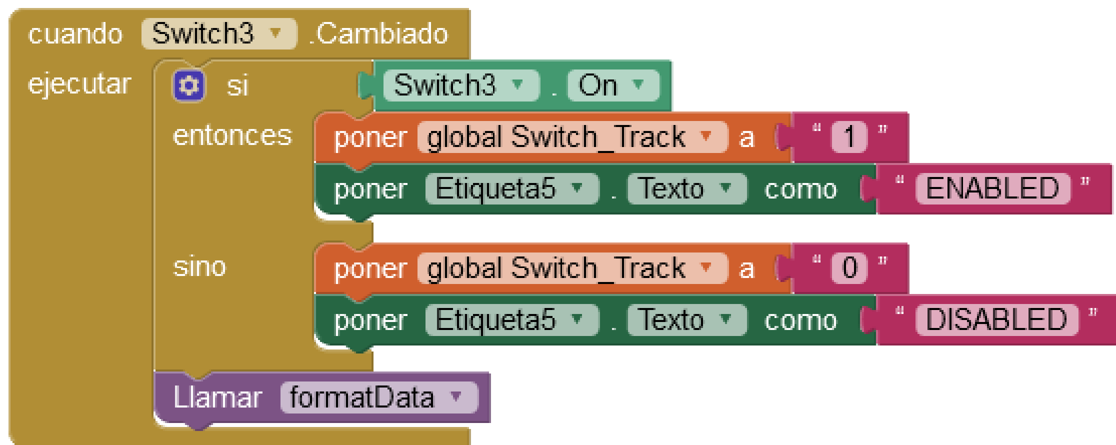


Figura 4.3: Lógica del interruptor de cambio de vía [24].

Esta función se ejecutará en la aplicación, y tiene el comportamiento habitual de una interrupción: se activa cuando detecta un cambio en el interruptor. A continuación mira el estado del interruptor: si ha cambiado a "activado", preparamos un "1" y lo informamos por pantalla mediante "ENABLED"; por el contrario, si el interruptor ha cambiado a "desactivado", preparamos un "0" y lo informamos nuevamente por pantalla mediante "DISABLED". A continuación se llama a la función *formatData*.

La función *formatData* sirve para dar un formato para toda la información gestionada en la aplicación. Esto es de gran utilidad porque de esta forma la placa Arduino puede saber qué es lo que le va a llegar por bluetooth. En la Figura 4.4 podemos ver una versión simple de lo que hace esta función.

Figura 4.4: Función *formatData* [24].

Esta función une en una misma variable (*global_data*) todas las variables esenciales que la aplicación utiliza. En términos de código, está haciendo una cadena de variables de tipo *char* que juntas forman un *string*, y lo hace en ese orden. Posteriormente esta variable *global_data* se enviará por bluetooth. De esta forma, la placa Arduino sabe siempre qué es lo que le está llegando: sabe que los primeros *char* hacen referencia a la velocidad, que a continuación vienen los estados de los semáforos 1 y 2, y que por último llega el estado del cambio de vía. De esta forma, es más sencillo programar a qué módulo enviarle qué por I2C.

La aplicación tiene un *timer* que salta cada 200 milisegundos, y cada vez que salta, se ejecuta una rutina de forma similar a como lo hacen las interrupciones. Esta rutina la podemos ver en la Figura 4.5.

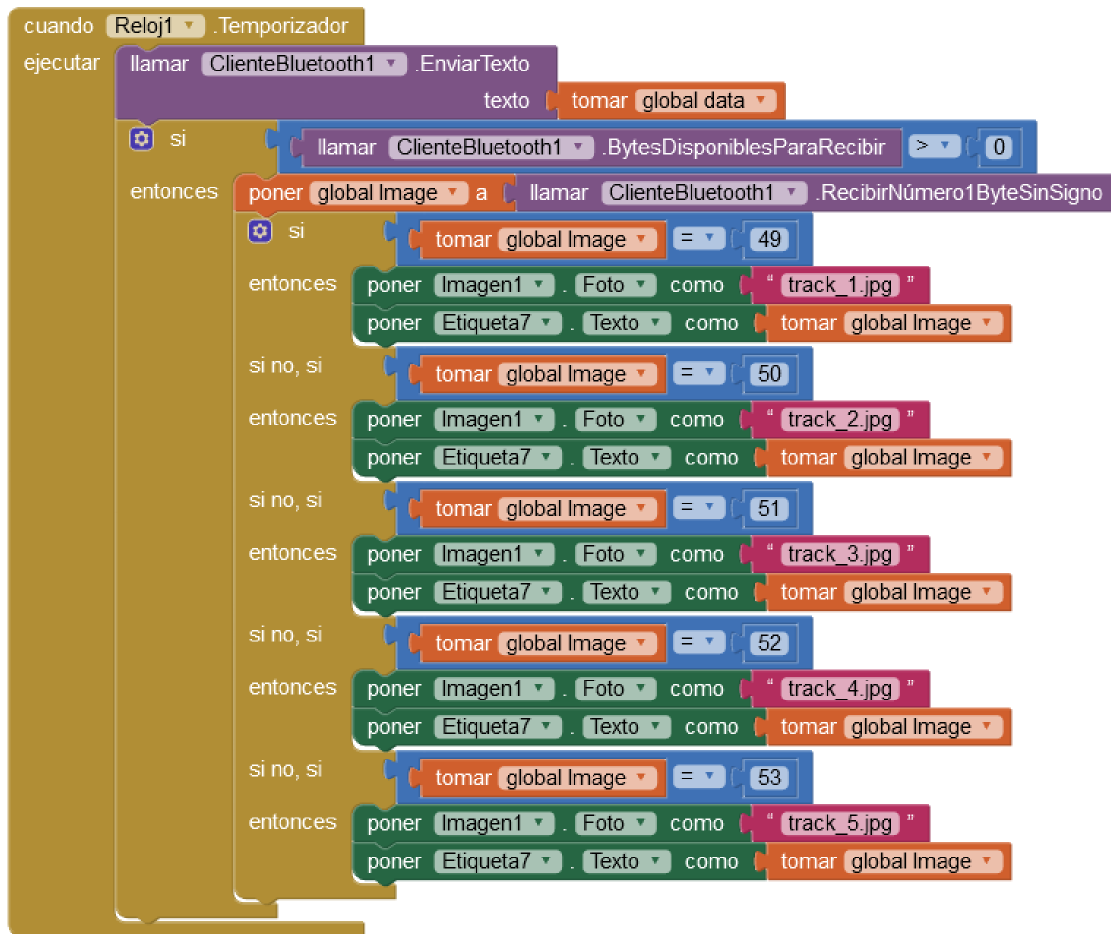


Figura 4.5: Rutina de la interrupción del *timer* [24].

Cuando se ejecuta la rutina, lo primero que hace es enviar por bluetooth los datos previamente generados con la función *formatData* anterior. A continuación comprueba si tiene datos pendientes de recibir y, si los tiene, extrae un byte *unsigned* de información. Este byte es el que le ha enviado la placa Arduino con la información sobre qué sensor ha sido el último en detectar el tren. Con un byte es suficiente para codificar esta información, por eso leemos de uno en uno. Además, la placa Arduino solo envía este byte si la información que contiene es diferente al último byte enviado, de esta forma evitamos saturar el *buffer* del bluetooth. La lógica restante compara el byte con valores de la tabla *ASCII* para decidir qué sección iluminar en la aplicación.

En la Figura 4.6 tenemos un ejemplo del código para que la Arduino solo envíe el byte de datos si es diferente al anterior.

```
Wire.requestFrom(0x31, 1);              // request 1 bytes from slave device 0x31

while (Wire.available()) {              // slave may send less than requested
  uint8_t result = Wire.read();         // receive a byte as character
  if (result != lastResult) {
    Serial.println(result);             // send the byte to the APP
    lastResult = result;                // save the last valid result known
  }
}
```

Figura 4.6: Código para enviar un byte de datos a la aplicación desde la Arduino.

Por último, para terminar de integrar la aplicación en todo el sistema, falta añadir el módulo de bluetooth y añadir el código necesario al programa principal para que la placa Arduino pueda hacer uso de esta funcionalidad. En la Figura 4.7 aparece una versión simplificada de la recepción de datos procedentes de la aplicación.

```
while (Serial.available() and count < 6) {
  delay(50);                       // Requesting 6 bytes from the App
  if (Serial.available() > 0) {
    char c = Serial.read();
    chunk_data += c;
    ++count;
  }
}

if (chunk_data.length() == 6) {
  train_speed = chunk_data.substring(0,3).toInt();
  sem_1 = chunk_data.substring(3,4).toInt();
  sem_2 = chunk_data.substring(4,5).toInt();
  switch_track = chunk_data.substring(5,6).toInt();
  chunk_data = "";
  count = 0;
}
```

Figura 4.7: Código para recibir seis bytes de datos de la aplicación a la Arduino.

Nótese que hay que hacer un [casting] para pasar los datos a una variable de tipo entero, ya que recordemos, lo que la función *formatData* hacía era generar un *string*. Además, si prestamos atención, notaremos que el orden en que los leemos y le hacemos el [casting], es el mismo orden en el cual los uníamos (Figura 4.4). Por este motivo la función nos facilitar el trabajo, porque ya sabemos que nos va a llegar. Toda la información extraída se guarda en variables de tipo *uint8_t*.

Con la información ya disponible, ahora solo queda enviarla a los módulos y a la aplicación. En la Figura 4.8 tenemos este proceso de forma simplificada.

```
Serial.print(track_segment);
delay(10);                    // Sending position to the App through bluetooth

analogWrite(SPEED, train_speed);

Wire.beginTransmission(semaphore_slave_1);
Wire.write(sem_1);
Wire.endTransmission();
delay(10);                    // Sending signal to Semaphore PIC 1 through I2C

Wire.beginTransmission(semaphore_slave_2);
Wire.write(sem_2);
Wire.endTransmission();
delay(10);                    // Sending signal to Semaphore PIC 2 through I2C

Wire.beginTransmission(track_switch_slave_1);
Wire.write(switch_track);
Wire.endTransmission();
delay(10);                    // Sending signal to Track switch PIC 1 through I2C

delay(100);                   // Safety delay
```

Figura 4.8: Código para enviar la información tratada hacia los módulos y la aplicación.

Distinguimos:

- Variable *track_segment*: similar a la Figura 4.6, contiene el último sector en detectar el tren. Este byte se recibe de un módulo y se envía a la aplicación.
- Variable *train_speed*: se recibe de la aplicación y se envía al transistor haciendo uso del PWM.
- Variables *sem_1*, *sem_2* y *switch_track*: se reciben de la aplicación y se envían a sus respectivos módulos.

Los *delay* que aparecen son por seguridad, de que la comunicación se realiza con el tiempo suficiente para no provocar colisiones. No afecta al rendimiento del sistema puesto que sucede más rápido que la velocidad máxima que alcanza el tren.

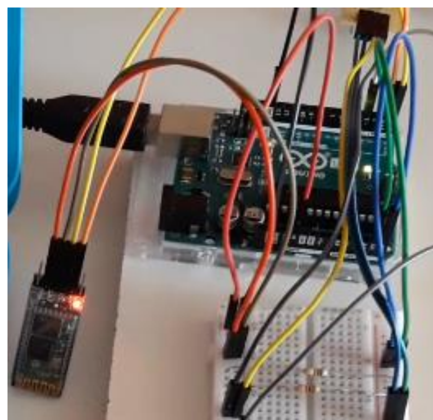


Figura 4.9: Módulo de bluetooth añadido al sistema.

5 Evaluación de la funcionalidad del proyecto

Cuando todos los test y pruebas de los circuitos en la protoboard parecen funcionar correctamente, y con una versión inicial de la aplicación ya operativa, es el momento de probar todo el sistema. Para ello hay que montar todas las piezas que lo componen sobre la base. Hay que montar todo el cableado necesario previamente, ya que una vez fijada la maqueta sobre la base, retirarla no es trivial. La mejor forma de proceder es hacer pasar todo el cableado por la cara inferior de la base, para que no obstruya, y dejar un extremo de los cables en la cara superior mediante un orificio. Este trabajo de cableado puede verse en la Figura 5.1.

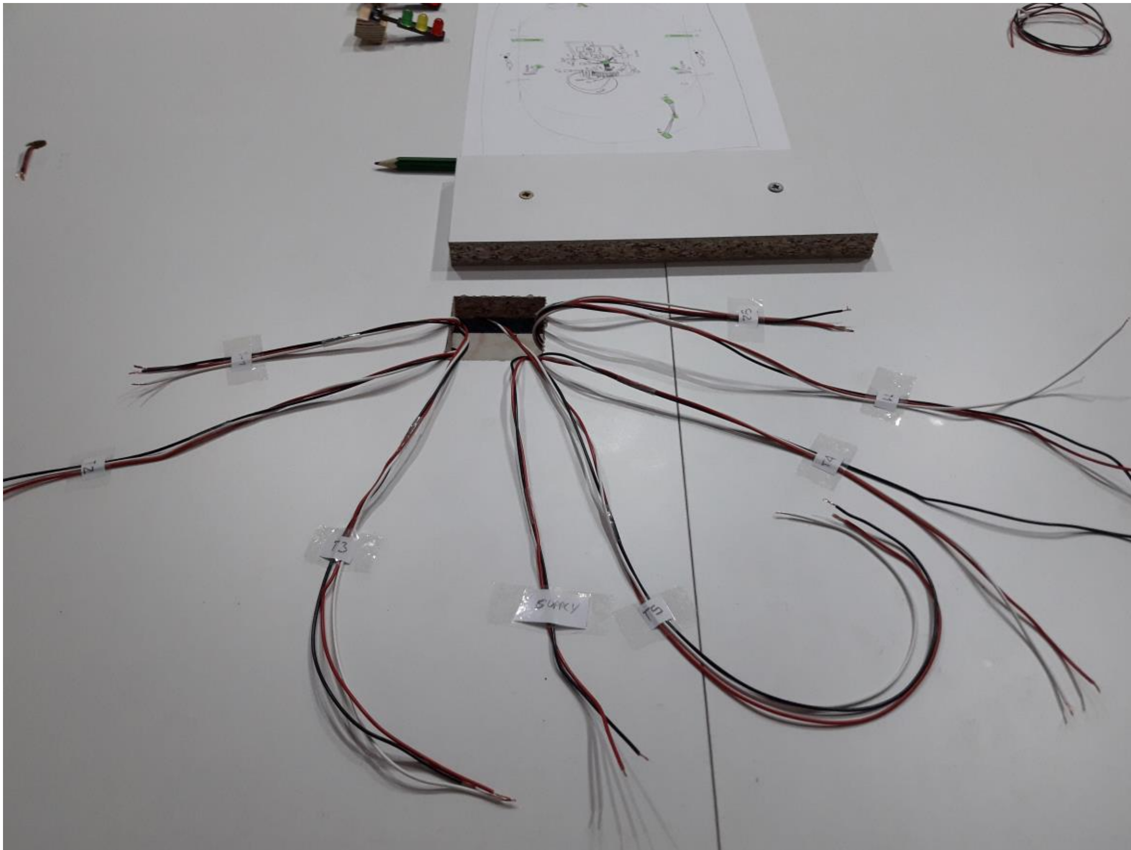


Figura 5.1: Estructura del cableado del sistema en la base.

Es en esta zona de la base donde colocaremos los circuitos y el sistema con la placa Arduino. Con el cableado ya fijado, ahora hay que colocar los cinco sensores de presencia. Este paso no es trivial, puesto que para que el sensor no quede obstaculizado, tanto el emisor de infrarrojos como el receptor de infrarrojos han de coincidir entre las *traviesas* de la vía. En la Figura 5.2 podemos observar uno de los sensores funcionando, y podemos apreciar que está encajado entre dos *traviesas*, ya que podemos ver la luz del emisor de infrarrojos (el ojo humano no es capaz de verlo, pero el objetivo de una cámara sí que puede).

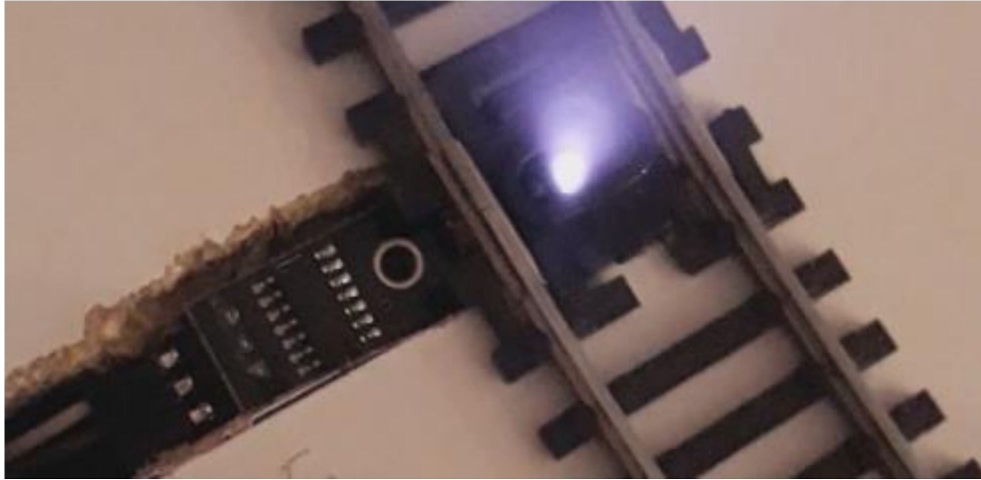


Figura 5.2: Ejemplo de sensor encajado con éxito entre dos *traviesas*.

5.1 Montaje de la maqueta

Con los sensores ya colocados, ya podemos colocar y fijar la maqueta, así como el resto de accesorios (semáforos y solenoides). La Figura 5.3 muestra la base prácticamente completada, con los semáforos, los sensores y la vía ya colocados.

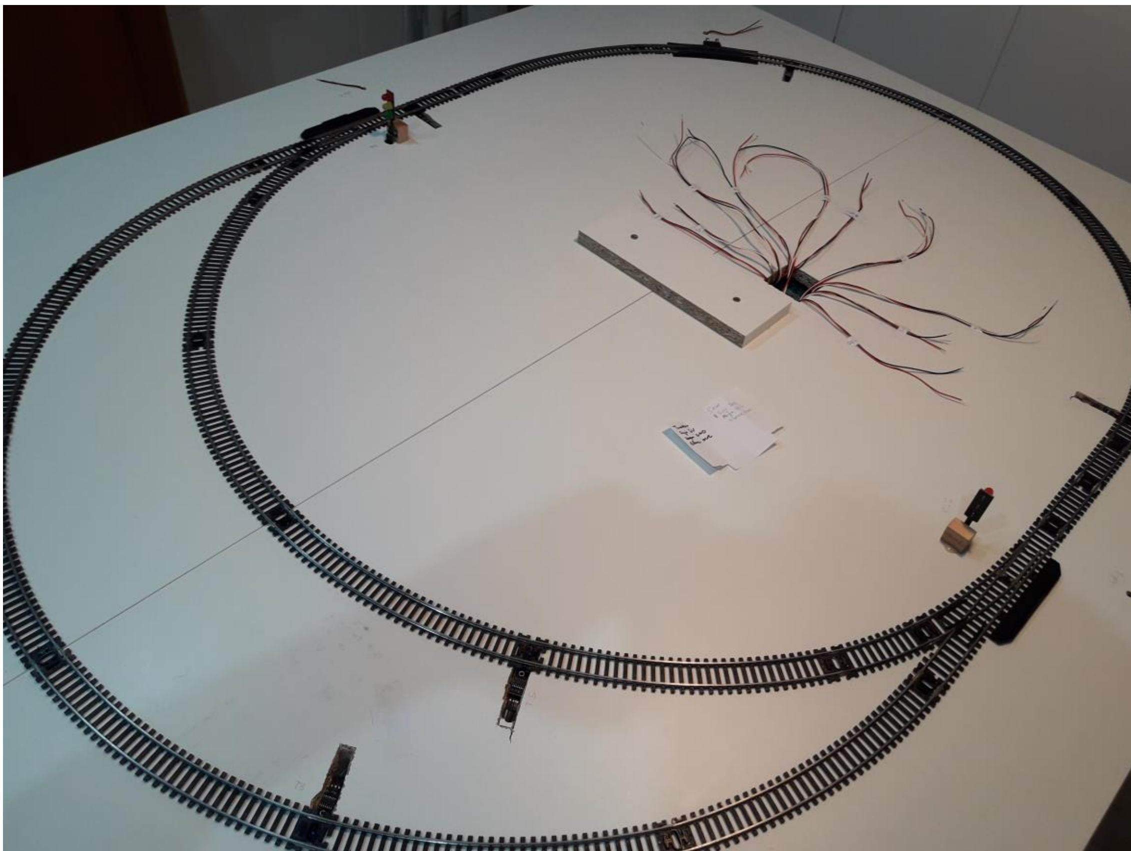


Figura 5.3: Proceso de construcción de la base.

Con la base ya lista, podemos empezar a colocar la placa Arduino y los circuitos, y comenzar a probar su funcionamiento. En la Figura 5.4 se muestra el módulo de detección de posición ya instalado, así como el móvil de pruebas para probar la aplicación.

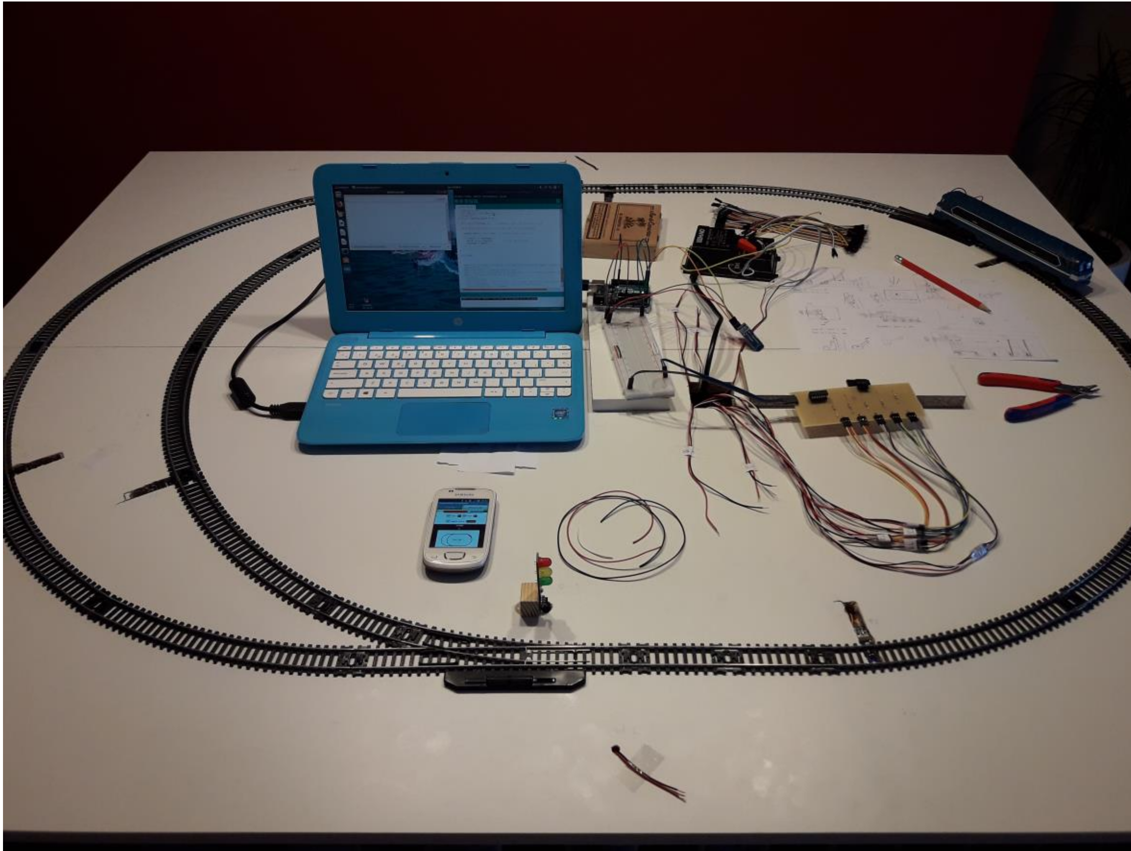


Figura 5.4: Proceso de construcción de la base: módulo de control de posición.

Se detallará más adelante, pero la protoboard se ha utilizado como un *hub* con el que poder conectar todos los módulos tanto a la alimentación como al bus I2C, y así poder hacer pruebas. Cada módulo que se fue instalando, se fue probando que cumpliera con su función. Es decir, se fue probando el sistema de forma incremental.

A continuación se instalaron los módulos de control de semáforos, que podemos apreciarlo en la Figura 5.5. Especial atención a que ya puede verse que los semáforos están funcionando (color rojo).

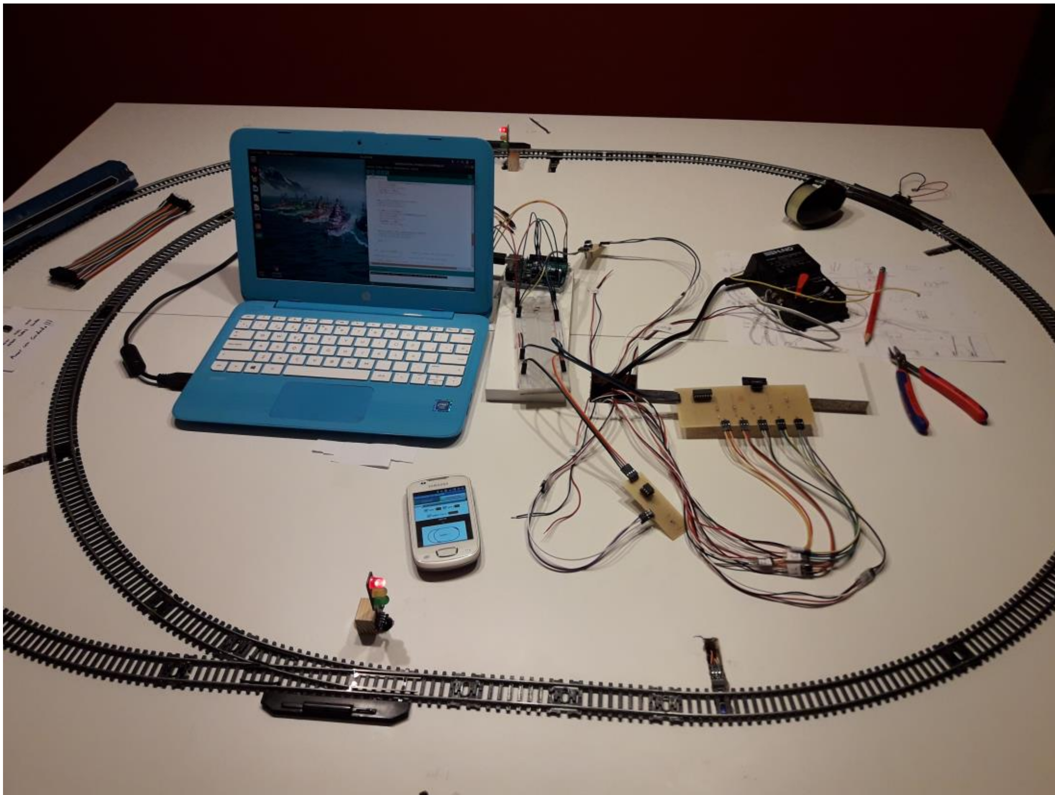


Figura 5.5: Proceso de construcción de la base: módulo de control de semáforo.

Y posteriormente se añadieron los módulos restantes. Nótese que, además del resto de módulos, en la Figura 5.6 ya están instalados los solenoides.

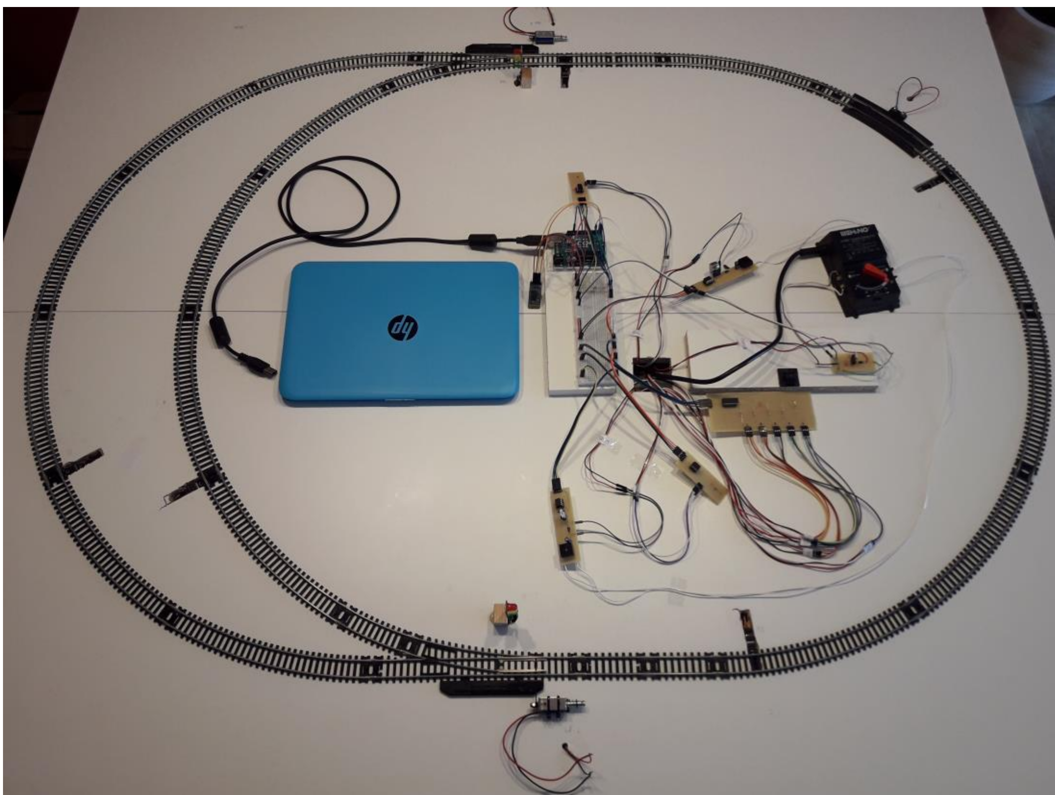


Figura 5.6: Proceso de construcción de la base: resto de módulos instalados.

5.2 Conexiones

Como hemos dicho previamente, la protoboard sirve de *hub* para las conexiones de los módulos, ya que la malla que la forma es ideal para lo que necesitamos. Una vista más de cerca de la protoboard, así como un esquema de sus conexiones, se muestra en la Figura 5.7.

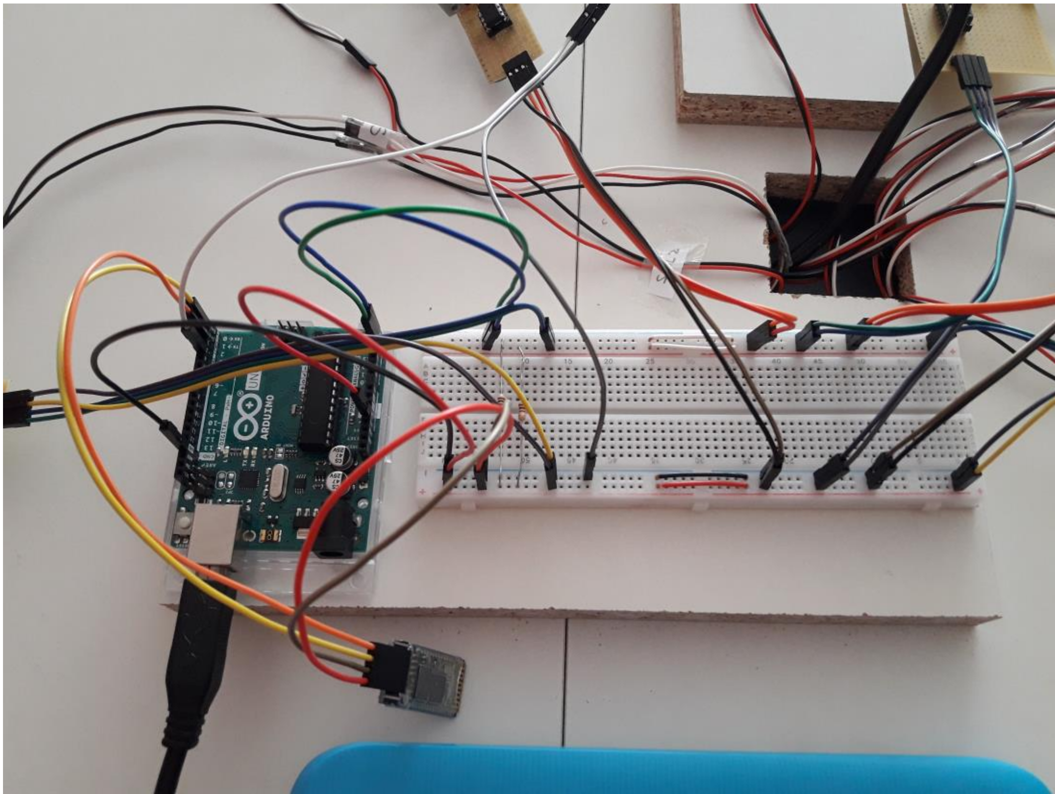


Figura 5.7.a: Conexiones en la protoboard.

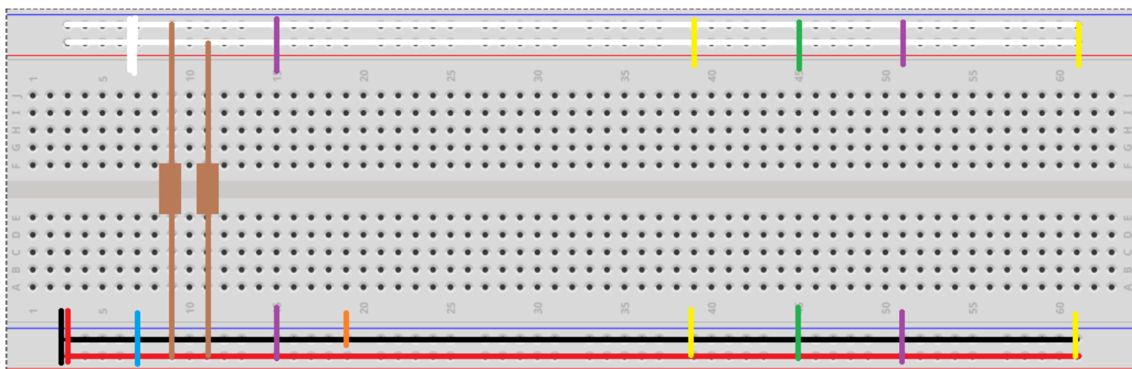


Figura 5.7.b: Esquema de las conexiones de la protoboard.

Donde los colores significan:

- Rojo y negro: los 5V y el *GND* de la placa Arduino respectivamente.
- Blanco: *SCL* y *SDA* del I2C.
- Azul: Bluetooth.
- Amarillo, morado, verde y naranja: los módulos de cambio de vía, control de semáforo, detección de posición, y control de velocidad respectivamente.
- Marrón: Resistencias de *pull-up* del bus I2C.

5.3 Test

Por último queda probar el funcionamiento del sistema, y para ello iremos probando todos los módulos con la aplicación del móvil. Para empezar tenemos que cargar el programa, y hay que tener cuidado de desconectar el módulo de bluetooth, porque produce colisión con el programador de la placa Arduino. Con el programa listo, volvemos a conectar el bluetooth, y pasamos a probar el módulo de cambio de vía.

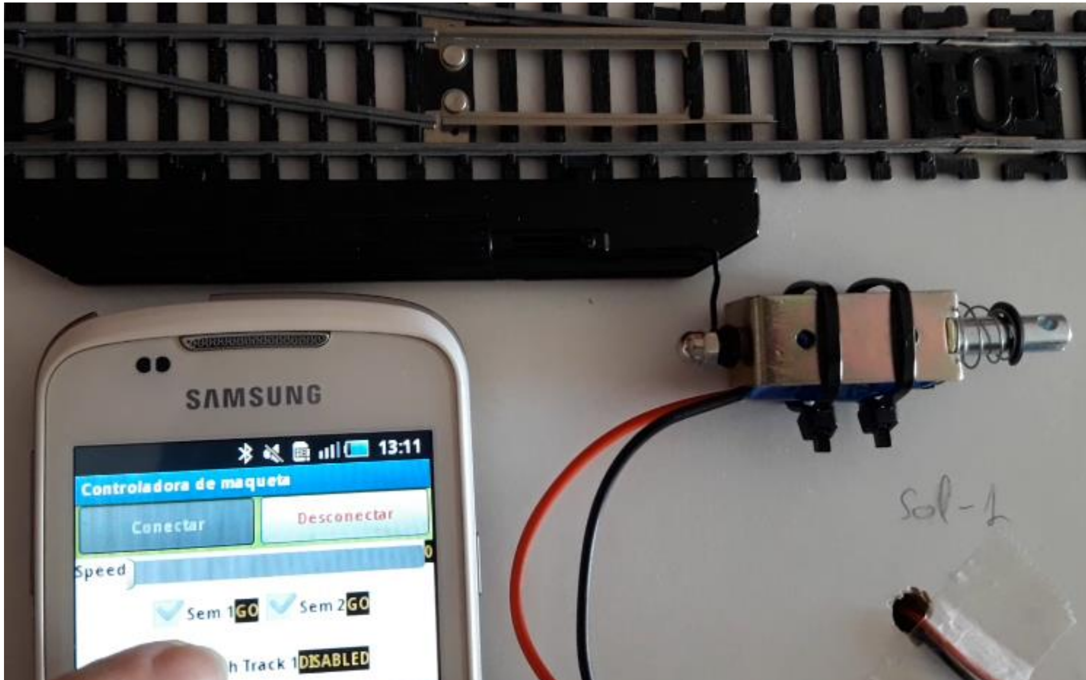


Figura 5.8.a: Controlando el solenoide desde la aplicación (DISABLED).

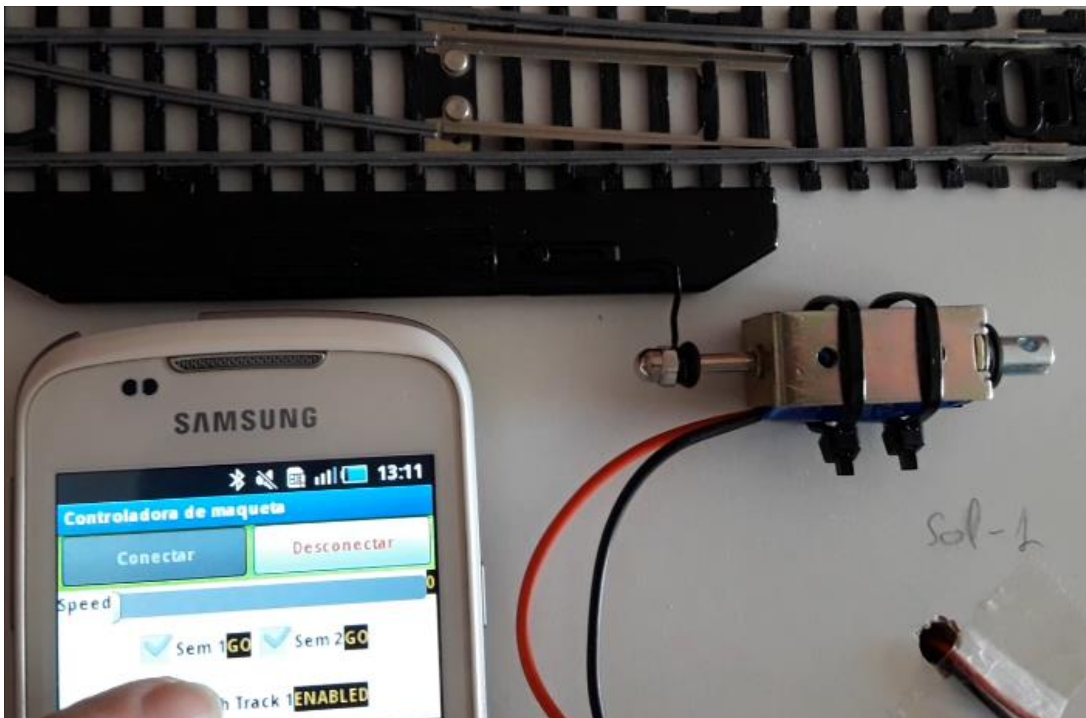


Figura 5.8.b: Controlando el solenoide desde la aplicación (ENABLED).

El funcionamiento del módulo, así como la instalación del solenoide es un éxito (Figura 5.8). No solo funciona el módulo y su interacción con el sistema y la aplicación, si no que el circuito además rectifica la onda de la manera esperada. Tal como vemos en las imágenes anteriores, hemos conseguido que el solenoide (que recordemos, funciona con DC) se controle desde la aplicación, a pesar de que el transformador da AC. Nótese como el brazo que empuja el solenoide cambia la vía.

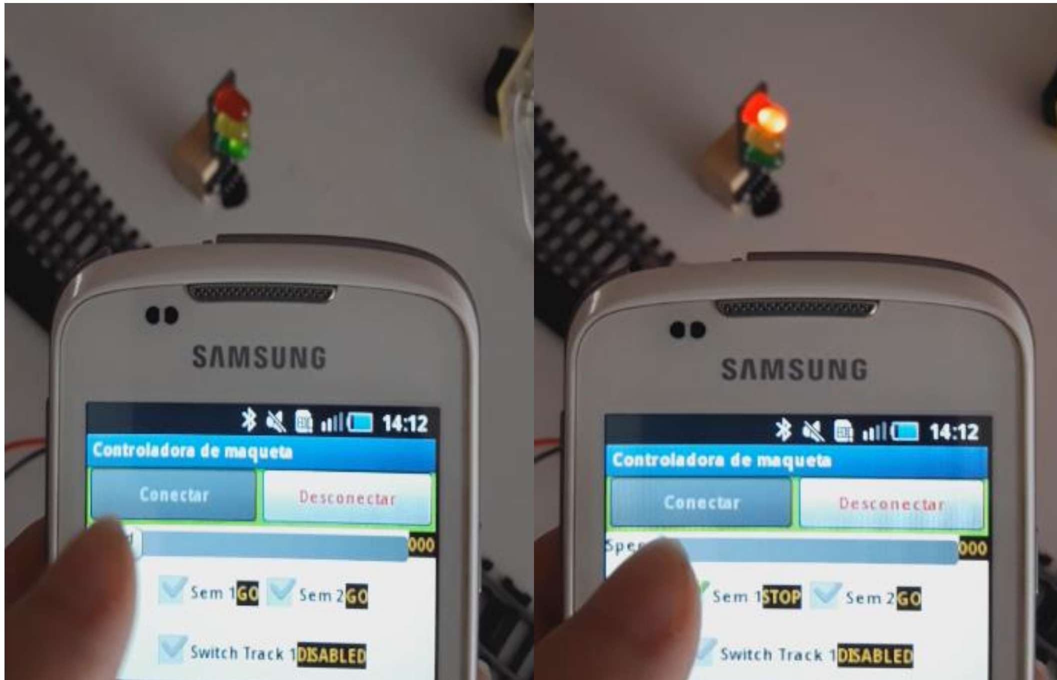


Figura 5.9.a: Semáforo en verde (GO). Figura 5.9.b: Semáforo en rojo (STOP).

En la Figura 5.9 vemos el resultado de probar el módulo del semáforo, y en este caso vemos que podemos controlar el comportamiento de este desde la aplicación. Especial atención en que el móvil nos dice el estado del semáforo por medio de GO (verde) y STOP (rojo).

A continuación probamos el módulo de detector de posición. Para ello nos acercaremos a un sensor con el móvil cerca y provocaremos que el sensor detecte algo. En la Figura 5.10 podemos ver que el sensor ha detectado una presencia (lo sabemos porque cuando esto sucede, un LED rojo del sensor ilumina la cavidad donde este se encuentra, y esto puede apreciarse en la imagen), y la aplicación nos dice que se trata de la sección 2. Aunque en la imagen no se aprecia bien, hay dos maneras de corroborarlo:

- La primera es que el sensor está marcado a lápiz con el acrónimo *T2* (parte superior derecha de la imagen). *T2* es una nomenclatura que he usado y significa *Tracker_2*, que es el encargado de detectar en la sección 2.
- La segunda es que el número que aparece en la aplicación (sobre el mapa) hace referencia al sensor que ha detectado algo por última vez, que en este caso es 50. Y si recordamos la rutina *timer* (Figura 4.5) y lo que dijimos sobre como tratamos los datos, veremos que ese valor es interpretado como un *char*. Y 50 en la tabla *ASCII* es, efectivamente, 2.

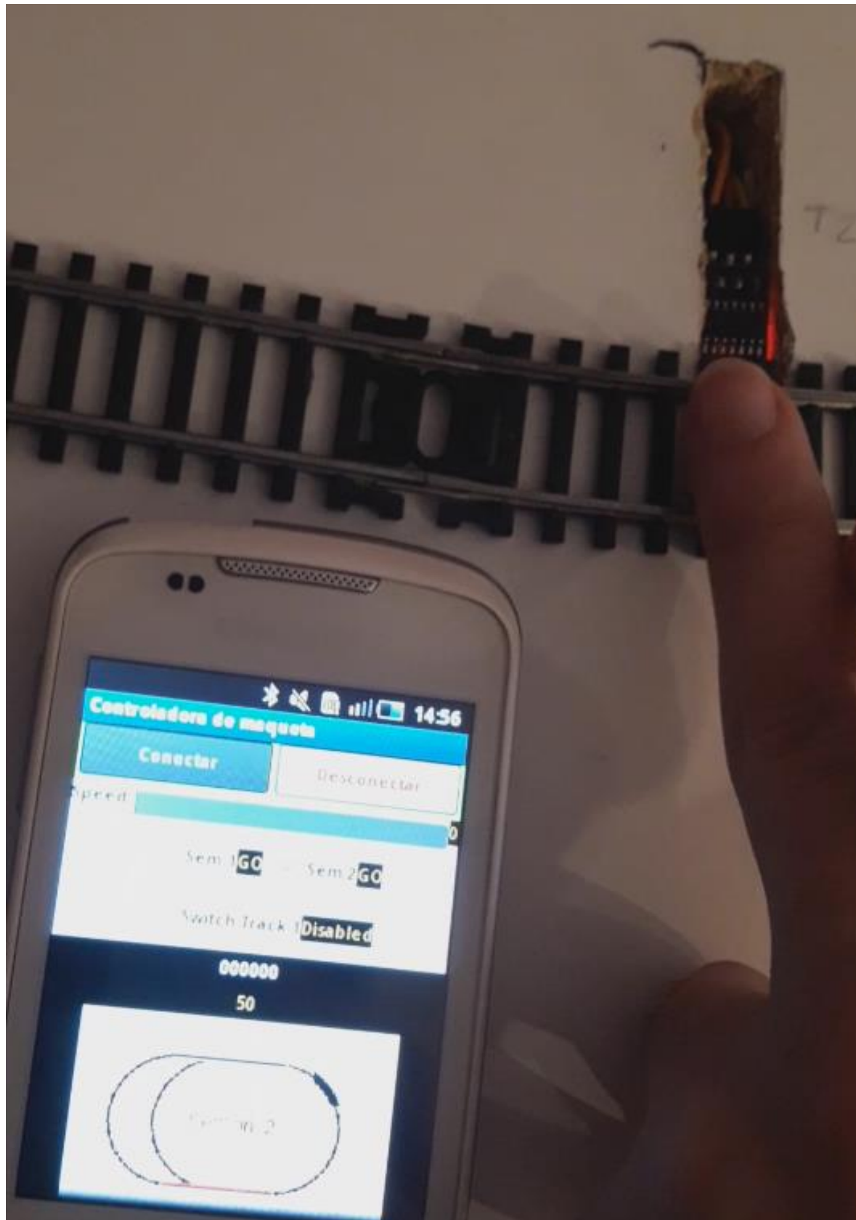


Figura 5.10: Comprobando el funcionamiento del módulo detector de posición.

Por último, resta probar el módulo de control de velocidad, y para ello hay que usar el tren. De forma colateral, en esta prueba se verá nuevamente el funcionamiento del módulo detector de posición, ya que el tren pasará por encima de los sensores. Aunque en una imagen no puede apreciarse el movimiento del tren, en la Figura 5.11 se observa dicha prueba.

Si miramos la barra de velocidad, vemos que el tren está alrededor del 80% de potencia. Y además el tren vuelve a encontrarse en la sección 2 (valor *ASCII* 50, como en la Figura 5.9). Esto lo sabemos por dos motivos, de forma similar a la prueba del módulo anterior:

- El primero es que número que aparece de color blanco (211000) es un número de *debug* para poder saber en todo momento qué valores estoy introduciendo en la aplicación. Los tres primeros dígitos hacen referencia a la velocidad (211 en este caso). Y si recordamos, los valores que puede tomar el *PWM* están comprendidos entre 0 y 255, luego el 211 es un valor esperado que además corresponde con la posición de la barra de velocidad de la aplicación. Nótese por ejemplo, que en la Figura 5.10, el mismo dígito aparece como 000000, y que la barra de velocidad está al 0%.
- El segundo es que cuando el tren está en movimiento, una luz en la cabina se enciende, y esta se puede apreciar en la imagen (a la misma altura que la luz verde del semáforo).

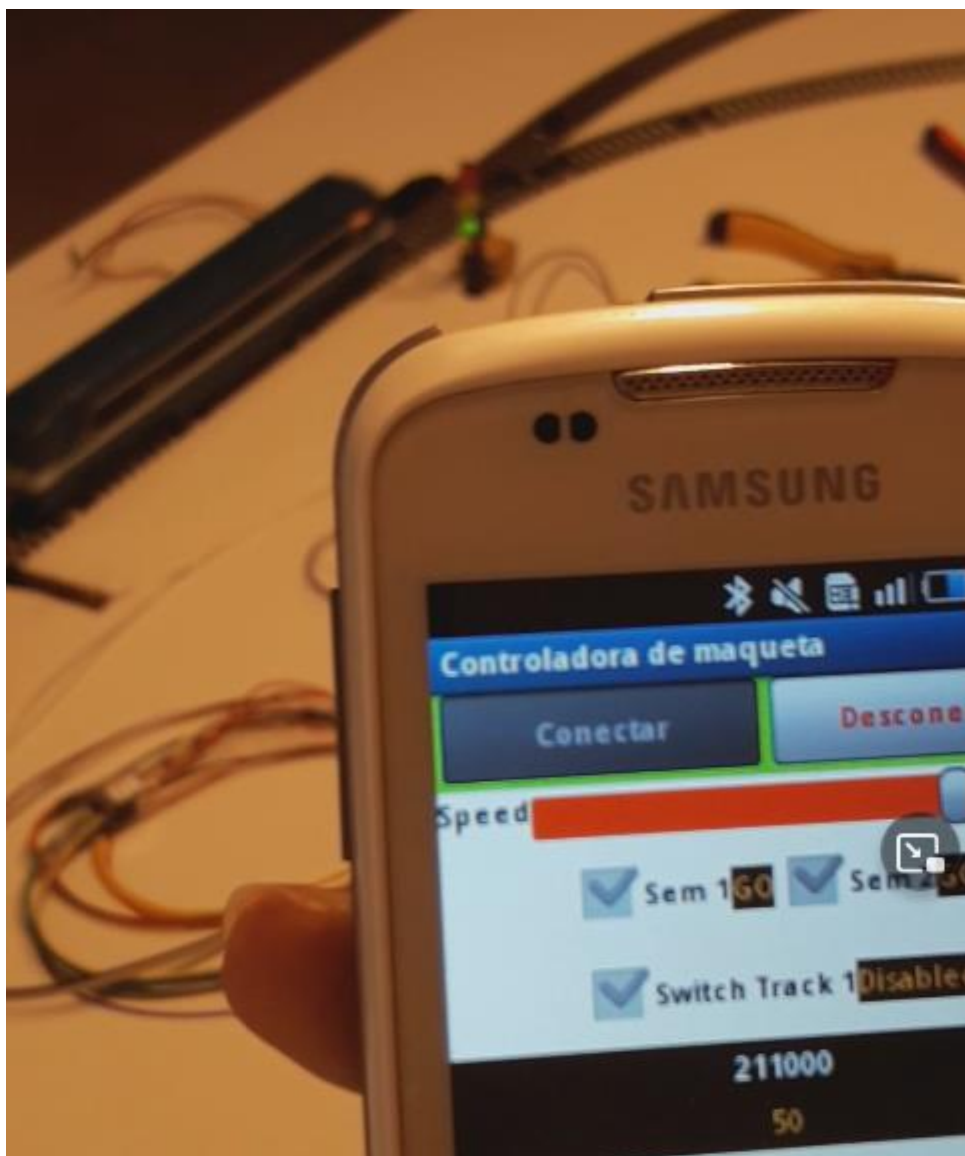


Figura 5.11: Comprobando el funcionamiento del módulo de control de velocidad.

5.4 Análisis de los resultados

Una vez acabados los tests, vamos a repasar de nuevo los requisitos que planteamos en el apartado 1.3.2, y justificarlos:

- Todos los módulos deben funcionar, y no deben de provocar colisiones unos con otros.
    - Tras realizar los tests anteriores, podemos afirmar que los módulos efectivamente funcionan. En cuanto a las colisiones, es difícil de demostrar mediante imágenes, pero si prestamos atención al test del módulo de control de velocidad, veremos que este módulo estaba siendo probado mientras el módulo de detección de posición estaba en funcionamiento. Además, tal como mencionamos en el apartado 5.1, los módulos se fueron probando de forma incremental, por lo que al instalar y probar el último módulo de forma exitosa, podemos afirmar que los módulos funcionan a la vez si provocar colisiones entre ellos.

- Los módulos deben de poder gobernarse con el programa principal.
    - Todos los tests realizados en el capítulo 4 fueron desde el programa principal, sin hacer uso de la aplicación, para verificar que no había un error de diseño. O para corregirlo de ser así. Por lo que podemos afirmar que los módulos pueden ser controlados mediante el programa principal, independientemente de donde vengan los datos.

- La aplicación debe de poder comunicarse con el programa principal.
    - Los tests realizados en este capítulo 5 así lo demuestran: la aplicación puede comunicarse con el programa principal de la placa Arduino, que a su vez controla los módulos. El hecho de cumplir este requisito se puede traducir a que, efectivamente, nuestro sistema soporta con éxito las comunicaciones bidireccionales:
        - Aplicación => Arduino => Módulo
        - Módulo => Arduino => Aplicación

- El sistema tiene que poder controlarse a distancia.
    - La integración y correcta configuración del bluetooth permite que el sistema pueda ser controlado a tanta distancia como el módulo de bluetooth así lo permita. Por tanto podemos afirmar que cumplimos este último requisito.

Analizados nuevamente los requisitos, concluimos que la realización de este proyecto ha sido un éxito. Ya que hemos logrado cumplir todos los requisitos propuestos y creado una primera versión del sistema que cumple con todas las expectativas dentro del tiempo estimado y sin mayores contratiempos.

6 Conclusiones

Para este proyecto, creo que he conseguido todos los objetivos inicialmente propuestos por tal de conseguir diseñar e implementar un sistema modular basado en microcontroladores. El diseño de los módulos y de la circuitería de todo el sistema está prácticamente completado para esta versión inicial. Y ahora ya estoy pensando en como lo podría mejorar en relación al *software*, ya que creo que tiene potencial para abarcar más funcionalidades aun.

He de decir que aunque ya partía con una base de conocimientos que eran necesarios para la realización de este proyecto, considero que los he podido aumentar en gran medida. Desde el boceto de cómo podría ser el sistema, a los diseños sobre *EAGLE*, y también de la parte electrónica en relación al comportamiento de los microcontroladores. Por ejemplo, no sabía que habían tanto la programación a bajo voltaje, como a alto voltaje; supongo que queda como anécdota que en uno de los test que hice en el circuito de cambio de vía, una mala conexión tuvo como consecuencia que friese un PIC… Pero no hubo que lamentar nada más. También he desarrollado más capacidad para sobreponerme a los imprevistos y a reaccionar con más rapidez a ellos.

Al comienzo del proyecto, tenía el objetivo de crear este sistema como una solución más simple a las opciones ya existentes para las maquetas que sean más simples. Demostrar que puede realizarse sin hacer uso de los elementos más caros (como los cambios de vía automáticos). No diré que he desarrollado la solución perfecta y definitiva, puesto que soy una persona que considera que nada nunca es perfecto, ya que siempre hay margen de mejora. Por ejemplo, soy consciente que la solución con solenoides no es la más eficiente, puesto que a fin de cuentas es una bobina y es puro consumo eléctrico. No se imaginan lo que daría por saber qué hay dentro de los cambios de vía automáticos (lo he buscado, pero sin éxito). Pero creo que he conseguido acercarme a ese objetivo, y quien sabe, tal vez este proyecto ayude a otra persona a perfilar más ese objetivo.

A lo largo del proyecto he aprendido a extraer y plasmar en un diseño la concepción de una idea. Es decir, desde que se tiene la idea hasta conseguir plasmarlo en papel y lápiz, para luego pasar al diseño por ordenador. He aprendido también a cómo decidir y en qué me debo de fijar a la hora de escoger los materiales para crear lo que sea que quiera crear. En este caso a decidir qué transistores serían los más adecuados para el propósito que tenía, por ejemplo. O cómo analizar un problema y encontrarle una solución adecuada, que es lo que hemos visto con los solenoides que funcionan con DC, y el transformador que funciona con AC. Quizá no tan relevante, pero he desarrollado algunas habilidades y buenas praxis a la hora de soldar; y como una vez me dijo un profesor: *"Todo ingeniero que se dedique o tenga que trabajar con hardware, tiene que saber soldar*". También he aprendido a hacer uso de *MPLAB X IDE*, la herramienta por excelencia para programar PICs, pero aun me queda mucho por aprender ya que es una herramienta muy potente. En cuanto a trabajar con la IDE de Arduino y *EAGLE*, pues ya tenía bastantes conocimientos, pero nunca está de más aprender cosas nuevas; por ejemplo, dijimos en un apartado anterior que las placas Arduino tienen resistencias de *pull-up* internas, pero no son muy confiables para I2C.

Gracias a la asimilación y aprendizaje de todos estos conocimientos, fue posible construir este sistema. Y tras la realización de tests y pruebas sobre protoboard, fue posible hallar errores y repararlos. Con todas las pruebas realizadas, ya se pudo hacer un ensayo real con la maqueta y el tren sobre la vía. Y aunque también surgieron otros errores, como es de esperar, me alegra decir no fue complicado solucionarlos. He logrado tener un sistema básico que cumple con lo esperado, y además el tren ha sobrevivido a pesar de los años que tiene.

De cara al futuro me gustaría poder añadir más funcionalidades de software, ya que como dije, tiene potencial para abarcar más. Y como quiero que este trabajo pueda ayudar a otras personas que también estén interesadas en este mundo, me gustaría pensar que alguien lo puede tomar como base para desarrollarlo todavía más.

Una de las ideas que lancé así al aire cuando estaba diseñando la aplicación, fue la de crear una opción que fuese "piloto automático" y que fuese introduciendo valores aleatorios (pero razonables) a la aplicación. Por ejemplo, cuando el tren ha dado tres vueltas al circuito, se active el cambio de vía por 3 vueltas. O que varíe la velocidad de forma automática cada cierto tiempo. Y por supuesto, hallar otra forma de realizar el cambio de vía que sea más eficiente que la actual.

También me habría gustado probar el sistema con dos trenes a la vez, pero no dispongo ni de otro tren, ni de otro transformador como tal. No obstante, el sistema debería ser capaz de adaptarse a este cambio con facilidad, ya que ha sido diseñado con módulos precisamente para esto.

Para este proyecto, se han contemplado las siguientes competencias técnicas:

CEC1.1: Diseñar un sistema basado en microprocesadores/microcontroladores [en profundidad]: todo el proyecto se basa en la creación de un sistema basado en módulos controlados por microcontroladores, y lo vemos a lo largo de toda la memoria.

CEC2.1: Analizar, evaluar, seleccionar y configurar plataformas *hardware* para el desarrollo y ejecución de aplicaciones y servicios informáticos [en profundidad]: la mayoría de los módulos del capítulo 3 han sido realizados bajo microcontroladores PIC que funcionan bajo arquitecturas de instrucción *RISC*, para los cuales se les ha dedicado el capítulo 2 (donde se comentan, entre otras, las ventajas de su uso). Así como el programa principal que se asienta sobre la placa Arduino en el capítulo 3 y 4.

CEC2.3: Desarrollar y analizar *software* para sistemas basados en microprocesadores y sus interfaces con usuarios y otros dispositivos [en profundidad]: a lo largo del capítulo 2 hemos podido ver la programación y el testeo del *software* de los módulos que componen el sistema, así como la comunicación por I2C en los capítulos 2 y 3.

CEC3.1: Analizar, evaluar y seleccionar las plataformas *hardware* y *software* más apropiadas para el soporte de aplicaciones encastadas y de tiempo real [en profundidad]: la aplicación ha sido desarrollara en el entorno App Inventor para la elaboración de aplicaciones para el sistema operativo *Android*, que podemos ver en el capítulo 4. Su evaluación supone la evaluación de todo el sistema, que tiene lugar en el capítulo 5.

Apéndices

A Gestión de riesgos: planes alternativos y obstáculos

Como hemos visto en la sección de obstáculos y riesgos, la duración estimada de las tareas puede verse afectada a causa de ellos. El peor de los casos sería que nos notificaran un retraso en el envío de materiales, ya que sin los módulos, no podemos empezar a codificar y probar el programa principal.

Afortunadamente, son de mi conocimiento algunos establecimientos especializados cercanos, que pueden proveerme (aunque a un precio más elevado, por tanto sería un coste imprevisto) de materiales específicos similares en poco tiempo (lapso de días). Si se diera el caso de que los materiales llegaran a medias (de ahí que los pida todos a la vez), este proyecto está diseñado para ser modular. Por lo que mientras espero que unos materiales lleguen, puedo empezar a ensamblar, codificar, y testear parte del programa en función de los materiales que vayan llegando.

El caso de la pandemia, desgraciadamente no tiene solución. En cuanto a la trata de elementos con corriente eléctrica, basta con seguir unas normas de seguridad y utilizar el equipo apropiado. Otro aspecto a mencionar es que en el diagrama de Gantt ya he tenido en cuenta los posibles retrasos que las funcionalidades de comunicación puedan causar, asignado unas cuantas horas más de las previstas a esas tareas.

B Presupuesto

B.1 Identificación de costes

Para identificar los costes de proyecto, tenemos que tener en cuenta varios recursos. Para empezar tenemos los recursos humanos, que se hacen patentes en la realización de las tareas del proyecto, para los que he decidido crear perfiles. Haciendo esta distinción, nos será más sencillo estimar mejor cuanto costará cada tarea. A continuación describo las responsabilidades de cada perfil, a las que les serán asignadas tareas del proyecto.

- **Perfil para el escritor de documentos técnicos (EDT):** la persona de este perfil tiene la responsabilidad de la documentación y el manejo de la terminología técnica que el proyecto requiera. Las tareas GEP1 – GEP4 y DP1 – DP3 serían asignadas a personas con este perfil, ya que requieren de un conocimiento técnico, característico de este proyecto.

- **Perfil para el ingeniero *hardware* (IH):** gran parte del proyecto recae en personas con este perfil, ya que se encargarán del diseño y ensamblaje de los módulos *hardware* de este proyecto. Las tareas PM1 – PM3, MCV1 – MCV5, MCCV1 – MCCV6, MCS1 – MCS5 y MDP1 – MDP6 serían asignadas a personas con este perfil.

- **Perfil para el programador *software* (PS):** otra gran parte del proyecto recae en personas con este perfil, pues serán las encargadas de la codificación y programación de los módulos y las aplicaciones. Las tareas PP1 – PP2 y AM1 – AM3 serían asignadas a personas con este perfil.

Seguidamente tenemos los recursos materiales genéricos, tales como *software* y *hardware*, los cuales tendremos que amortizar. Todo el *software* utilizado es libre y/o es suficiente con la versión limitada, sin embargo el *hardware* será la parte más costosa de los costes materiales.

También se incluyen en esa categoría los costes de electricidad e Internet, y otros gastos energéticos. Debido a la pandemia, no hay costes de movilidad, ni de espacio (alquiler o mobiliario).

Para acabar, están los costes de contingencia para cubrir obstáculos no previstos (que he decidido estimar en 10%), y los costes para los imprevistos que pudieran surgir, detallados en el apartado de obstáculos y riesgos anterior.

El presupuesto en su totalidad se identifican a continuación (Tabla b.1), desglosado en: costes por actividad, costes generales, costes de contingencia y los costes para imprevistos. En los costes por actividad, los perfiles se identifican por sus siglas en la columna de *comentarios*.

Actividad	Importe (€)	Comentarios
GP1 - Contexto y alcance del proyecto	198,50	EDT 10h
GP2 - Planificación del proyecto	198,50	EDT 10h
GP3 - Presupuesto y sostenibilidad	198,50	EDT 10h
GP4 - Integración en el documento final	99,25	EDT 5h
GP5 - Reuniones periódicas	653,25	IH y PS 15h
PM1 - Mediciones y determinación del tamaño de la base	259,04	IH 10h
PM2 - Diseño estructural de la maqueta	388,56	IH 15h
PM3 - Instalación de la maqueta y test eléctrico	259,04	IH 10h
MCV1 - Diseñar la implementación del módulo	207,23	IH 8h
MCV2 - Generar un BOM	51,81	IH 2h
MCV3 - Comprar lo decidido en el BOM	25,90	IH 1h y 1 semana de envio (coste 0)
MCV4 - Ensamblaje del módulo	155,42	IH 6h
MCV5 - Test eléctrico del módulo	51,81	IH 2h
MCCV1 - Diseñar la implementación del módulo	207,23	IH 8h
MCCV2 - Generar un BOM	51,81	IH 2h
MCCV3 - Comprar lo decidido en el BOM	25,90	IH 1h y 1 semana de envio (coste 0)
MCCV4 - Mecanización del cambio de vía	155,42	IH 6h
MCCV5 - Ensamblaje del módulo	155,42	IH 6h
MCCV6 - Test eléctrico del módulo y del cambio de vía	77,71	IH 3h
MCS1 - Diseñar la implementación del módulo	207,23	IH 8h
MCS2 - Generar un BOM	51,81	IH 2h
MCS3 - Comprar lo decidido en el BOM	25,90	IH 1h y 1 semana de envio (coste 0)
MCS4 - Ensamblaje del módulo	155,42	IH 6h
MCS5 - Test eléctrico del módulo	51,81	IH 2h
MDP1 - Determinal la precisión de la detección	129,52	IH 5h
MDP2 - Diseñar la implementación del módulo	207,23	IH 8h
MDP3 - Generar un BOM	51,81	IH 2h
MDP4 - Comprar lo decidido en el BOM	25,90	IH 1h y 1 semana de envio (coste 0)
MDP5 - Ensamblaje del módulo	155,42	IH 6h
MDP6 - Test eléctrico del módulo y de los sensores	77,71	IH 3h
PP1 - Codificación del programa	705,84	PS 40h
PP2 - Testeo del programa	352,92	PS 20h
AM1 - Codificación de la aplicación	529,38	PS 30h
AM2 - Añadir funcionalidades de comunicación al programa principal	352,92	PS 20h
AM3 - Testeo final	352,92	PS 20h
DP1 - Anotación de eventos	198,50	EDT 10h
DP2 - Revisión de eventos	198,50	EDT 10h
DP3 - Escribir documentación	794,00	EDT 40h
CPA total	8.045,07	
Hawrdware		La amortización del hardware es a 4 años
PC sobremesa	52,91	364h. Precio 1000€
Monitor	6,35	364h. Precio 120€
Teclado	0,63	364h. Precio 12€
Ratón	0,42	364h. Precio 8€
Arduino	0,47	130h. Precio 25€
Protoboard	0,09	130h. Precio 5€
1º Compra: sensores, PICs, transistores, solenoides…	1,23	130h. Precio 65€
2º Compra: bluetooth, conexiones acodadas, cableado…	0,66	130h. Precio 35€
Osciloscópio	0,67	20h. Precio 230€
Tester	0,06	20h. Precio 20€
Soldador	0,03	24h. Precio 10€
Software		El software libre no añade costes
LibreOffice	0,00	Libre
GanttProject	0,00	Libre
MPLAB X IDE	0,00	Gratuito
Arduino IDE	0,00	Libre
EAGLE	0,00	Gratuito
Otros gastos		
Electricidad	240,00	60€ al mes y el proyecto dura unos 4 meses
Internet	159,80	39,95€ al mes y el proyecto dura unos 4 meses
CG total	463,33	
Costes totales (CPA total + CG total)	8.508,40	
Contingencia	850,84	Margen de contingencia: 10%
Costes totales (CPA total + CG total + contingencia)	9.359,24	
Retraso en las funcionalidades de comunicación (1 semana)	123,52	Coste: PS 20h, riesgo: 35%
Retraso en el ensamblaje debido al envio de materiales (1 semana)	77,71	Coste: IH 20h, riesgo: 15%
Costes imprevistos	201,23	
TOTAL	9.560,48	

Tabla b.1: Presupuesto.

B.2 Estimación de costes

Tal como hemos dicho, el presupuesto estimado se divide en costes por actividad (CPA), costes generales (CG), costes de contingencia y los costes para imprevistos.

Para calcular el CPA, he asignado a cada tarea (actividad en la Tabla b.1) uno de los perfiles mencionados en el apartado anterior. Para cada tarea, el coste total es el coste de un perfil por hora, multiplicado por las horas totales de la tarea. En la siguiente Tabla b.2 se muestran los salarios anuales de cada perfil, incluido el coste de la seguridad social (que es el 30% del salario anual).

Perfil	Salario anual (€)	Salario anual + seguridad social (€)	Precio/hora (€)	Horas totales	Coste total perfil (€)
EDT	26.263,00	34.141,90	19,85	95	1.885,74
IH	34.273,00	44.554,90	25,90	104	2.694,02
PS	23.347,00	30.351,10	17,65	145	2.558,67

Tabla b.2: Salario anual de los perfiles involucrados.

El precio por hora es calculado teniendo en cuenta un contrato de 1720 horas anuales, según estipula el [10] BOE. Y los salarios surgen de la página web [11] *GlassDoor*, que muestra el salario medio de personas trabajado en un perfil concreto. El coste CPA total es de 8.045,07€.

En cuanto al CG, hay que tener en cuenta el material usado, *hardware* y *software* en este caso, y otros gastos como electricidad o Internet. El *software* usado en este proyecto es libre, por lo que solo el *hardware* implicará costes. El *hardware* lo amortizaremos mediante la siguiente fórmula: ((coste total del producto * total de horas usado en el proyecto) / (total de horas en 4 años)). Estas amortizaciones pueden observarse en la Tabla 2. En cuanto a otros gastos como electricidad o Internet, también se aporta una estimación suponiendo que el proyecto se desarrolle según lo previsto, lo que implica que tenga una duración de 4 meses. El coste CG total es de 463,33€.

Calculados el CAP y el CG, tenemos un coste total de 8.508,40€. A este coste total, le he añadido un 10% como margen de contingencia. Con este margen, el coste total asciende a 9.359,24€

Para acabar, he añadido costes imprevistos para el proyecto. Como hemos visto en las secciones previas, este proyecto no está exento de imprevistos, y el diagrama de Gantt ya tiene en cuenta algún posible retraso menor. Pero si alguna tarea del proyecto sufre un retraso que excede lo previsto, no queda más remedio que pagar salario extra al ingeniero de *hardware* (IH) y/o al programador de *software* (PS), en especial a este último, que es el que tiene mayor probabilidad de retrasar el proyecto. Estos costes imprevistos y su probabilidad pueden observarse en la Tabla 2. El coste total de imprevistos es de 201,23€, y el coste total del proyecto es de 9.560,48€.

B.3 Control de gestión

En esta sección, se describen los procedimientos para controlar el presupuesto, así como indicadores de control que ayudarán a supervisar las desviaciones de coste durante la ejecución del proyecto.

Una manera de medir y evitar desviaciones es obteniendo la diferencia entre los recursos reales consumidos y los recursos estimados consumidos. De esta forma podemos saber la desviación de costes en los recursos humanos (CPA) en términos de eficiencia por cada tarea descrita en el diagrama de Gantt. Para ello, tenemos la siguiente fórmula:

*Desviación de recursos humanos = (Coste estimado por hora – Coste real por hora) * Total de horas invertidas*

De forma similar, también podemos aplicar el mismo concepto al caso de los costes generales (CG). Como se ha mencionado, el *software* que utiliza este proyecto es libre, aunque la siguiente fórmula para la desviación de la amortización, sirve tanto para *software* como para *hardware*:

*Desviación de amortización = (Estimación de horas de uso – Horas reales de uso) * Precio por hora*

En cuanto a otros costes generales, la tarifa de Internet no acostumbra a cambar a menos que suceda un cambio en las políticas de la proveedora del servicio. Sin embargo el subministro de electricidad si que puede sufrir una desviación:

*Desviación del coste de electricidad = (Estimación de horas de uso – Horas reales de uso) * Precio por hora*

A continuación tenemos los costes debidos a imprevistos, cuya desviación tiende a ser imprevisible si el proyecto va cumpliendo los términos establecidos:

*Desviación del coste de imprevistos = (Estimación de horas de imprevistos – Horas reales de imprevistos) * Total de horas de imprevistos*

Para acabar, hay que sumar todas las desviaciones para obtener la desviación total:

Desviación total = Desviación de recursos humanos + Desviación de amortización + Desviación del coste de electricidad + Desviación del coste de imprevistos

Con toda esta información podemos saber en qué parte del proyecto recae la mayor desviación. Para tener una visión más general del presupuesto del proyecto, basta con comparar si el coste total del proyecto es mayor que el coste total estimado, y ver si los costes de contingencia fueron calculados adecuadamente.

C Sostenibilidad

A través de una encuesta facilitada por la UPC, me doy cuenta de que tengo un nivel intermedio de conocimiento sobre sostenibilidad en sus tres campos: ambiental, económico y social. Debido a que en algunos temas tengo ciertas nociones, pero en otros tal vez flaqueo, ya que dentro del campo de la ingeniería informática hay cantidad de aspectos a tener en cuenta en relación a la sostenibilidad.

Puedo concretar lo siguiente para cada campo de la sostenibilidad:

- **Ambiental:** En el campo ambiental tengo bastante conocimiento, debido a la cantidad de competencias que he realizado a lo largo de la carrera, y a las conferencias a las que he asistido. Una de las más notables a las que asistí fue a la de [12] "*The E-waste Tragedy*" dirigida por Cosima Dannoritzer. Este proyecto intentará que la huella ambiental sea lo más pequeña posible.

- **Económico:** En el campo económico es quizá donde más flaqueo, ya que es la primera vez que hago un proyecto de estas dimensiones. Sin embargo, esto ha sido una oportunidad para reforzar ese campo. Ya que he tenido que planificar un presupuesto para saber los costes del proyecto, así como la distribución de tareas, por lo que ahora creo entender un poco mejor como funcionan los aspectos económicos de un proyecto a lo largo de su vida útil.

- **Social:** En el campo social creo me hago una idea de cómo un proyecto como el mío puede afectar, directa o indirectamente, a la sociedad. Así como también creo saber evaluar si un proyecto contribuye a mejorar el bien común de la sociedad. Aunque supongo que este campo es el más difícil de asimilar, ya que las sociedades están en constante cambio.

C.1 Dimensión ambiental

Respecto a la PPP: ¿Has cuantificado el impacto ambiental de la realización del proyecto? ¿Qué medidas has tomado para reducir el impacto? ¿Has cuantificado esta reducción?

Aunque no lo parezca, cualquier proyecto que acabe tratando con circuitería (como es este), acaba teniendo un impacto ambiental. Y por su naturaleza, es difícil de medir (la fabricación de componentes con circuitería es muy contaminante por ejemplo, pero es difícil de medir). Habría sido una buena praxis intentar reducirlo, como el caso de los solenoides, que mencioné que no es una solución eficiente.

Respecto a la PPP: Si hicieras de nuevo el proyecto, ¿podrías realizarlo con menos recursos?

Posiblemente. Con el conocimiento y la experiencia que he adquirido, es de suponer que podría ahorrar materiales, y sobretodo tiempo. Aunque no lo parezca a simple vista, ahorrar en tiempo supone un ahorro de energía, ya que es menos tiempo que los dispositivos requieren estar encendidos.

Respecto a la vida útil: ¿Qué recursos estimas que se usarán durante la vida útil del proyecto? ¿Cuál será el impacto ambiental de estos recursos?

Prácticamente ninguno. Si se quiere ampliar el sistema, ciertamente harán falta más recursos para fabricar más módulos, pero el impacto será difícil de calcular por los mismos motivos que la pregunta anterior. En cuanto a mejorar el sistema en la parte de *software*, pocos recursos serán necesarios, ya que solo requiere de una terminal en funcionamiento y horas de dedicación.

Respecto a la vida útil: ¿El proyecto permitirá reducir el uso de otros recursos? ¿Globalmente, el uso del proyecto mejorará o empeorará la huella ecológica?

Depende. Es cierto que al ser un sistema cuya idea es que sirva para maquetas simples, los componentes que lo formen sean más simples (y por tanto, más baratos) que aquellas soluciones para el modelismo profesional. Sin embargo, al final ambas opciones requieres el mismo tipo de componentes. La diferencia está en la cantidad. Aunque si lo observamos de esta forma: aquellas personas que quieran una opción más simple, como la que hemos diseñado, implica que podrían no decantarse por una opción más compleja (que seguramente requiera un uso de recursos más exhaustivo).

Respecto a riesgos: ¿Podrían producirse escenarios que hiciesen aumentar la huella ecológica del proyecto?

No realmente. A lo largo del proyecto he recalcado que es la parte *software* la que tiene potencial para abarcar más funcionalidades, y esta es precisamente la que requiere de menos recursos para su desarrollo.

C.2 Dimensión económica

Respecto a la PPP: ¿Has cuantificado el coste (recursos humanos y materiales) de la realización del proyecto? ¿Qué decisiones has tomado para reducir el coste? ¿Has cuantificado este ahorro?

Así es. En el apéndice B.1 hay un presupuesto que tiene en cuenta tanto los costes materiales como los costes humanos necesarios para llevar a cabo el proyecto. Realmente ha sido un presupuesto bastante acertado desde el primer momento (quizá algún componente a media realización, pero con un impacto mínimo en el presupuesto final debido al margen de contingencia). De hecho, si ha habido un ahorro, es gracias a que prácticamente no hemos necesitado tocar el margen de contingencia.

Respecto a la PPP: ¿Se ha ajustado el coste previsto al coste final? ¿Has justificado las diferencias (lecciones aprendidas)?

Si, ha sido un presupuesto bastante ajustado a lo previsto.

Respecto a la vida útil: ¿Qué coste estimas que tendrá el proyecto durante su vida útil? ¿Se podría reducir este coste para hacerlo más viable?

Principalmente suministro eléctrico y recursos humanos, para el mantenimiento y la actualización del *software*, por tal de añadirle mejoras y más funcionalidades. De ser así, más tests serán necesarios, que implicarán más costes. No podría reducirse el coste más aun para mantenerlo, el mencionado es el mínimo necesario.

Respecto a la vida útil: ¿Se ha tenido en cuenta el coste de los ajustes/ actualizaciones/reparaciones durante la vida útil del proyecto?

En el apartado B.1 no tenemos en cuenta dicho coste de forma explicita, sin embargo si que aparece de forma estimada la amortización de todos los materiales usados durante la vida útil del proyecto.

Respecto a riesgos: ¿Podrían producirse escenarios que perjudicasen la viabilidad del proyecto?

Claro. Si alguien aparece con una idea que es implementada de mejor forma que la de este proyecto, y consigue llevarla a cabo de forma exitosa, perjudicará la viabilidad del proyecto. Sin embargo, ya concluí que este proyecto no es perfecto, y considero que es un primer acercamiento para este tipo de sistemas y que puede ser mejorado.

C.3 Dimensión social

Respecto a la PPP: ¿La realización de este proyecto ha implicado reflexiones significativas a nivel personal, profesional o ético de las personas que han intervenido?

Este proyecto sirve para demostrarme que tengo las cualidades necesarias para valerme por mismo. Como futuro ingeniero, este proyecto me sirve para acostumbrarme a la siguiente máxima: "si surge un problema, le encuentro una solución", ya que eso es lo que hacemos los ingenieros. He tenido ocasión de diseñar algo por mi cuenta y esto puede ayudarme en un futuro cuando tenga que decidir qué componentes seleccionar en una situación similar. He podido poner en práctica conocimientos aprendidos en la carrera, y ver su uso más allá de las clases, como la importancia de realizar bien un *BOM*, o buscarse la vida con un multímetro y un osciloscopio. Y por supuesto, que las soluciones llegan por ensayo y error.

Respecto a la vida útil: ¿Quién se beneficiará del uso del proyecto? ¿Hay algún colectivo que puede verse perjudicado por el proyecto? ¿En qué medida?

En la sección 1.1.4 comentamos quienes pueden ser las partes interesadas que podrían beneficiarse de este proyecto. Y como está orientado a un público concreto, no creo que pueda haber algún colectivo que pueda verse perjudicado por este proyecto.

Respecto a la vida útil: ¿En qué medida soluciona el proyecto el problema planteado inicialmente?

Este proyecto aporta una solución, con la previsión de que se tienen unas nociones básicas de los temas tratados, a esas personas que les gustaría hacer algo más con su maqueta ferroviaria. Pero que no necesitan una central digital (Figura 1.1) porque no necesitan una solución de carácter profesional.

Respecto a riesgos: ¿Podrían producirse escenarios que hiciesen que el proyecto fuese perjudicial para algún segmento particular de la población?

No. Este proyecto está orientado a esas personas que disfrutan montando maquetas ferroviarias y que quieren hacerlas mejor. No puedo pensar de algún colectivo de la población que pueda verse ofendido por esto.

Respecto a riesgos: ¿Podría crear el proyecto algún tipo de dependencia que dejase a los usuarios en posición de debilidad?

No. Este tipo de situaciones acostumbran a salir cuando una entidad crea un problema, para más tarde venderte la solución (los *smartphones* son un claro ejemplo de esta praxis). Este proyecto solo intenta facilitar y mejorar la calidad de vida del usuario que tiene dedicación al modelismo ferroviario.

Glosario

***Firmware*:** es un pequeño programa que hace que un dispositivo funcione tal y como el fabricante lo ha diseñado. Sin el, la mayoría de equipos electrónicos no funcionaría. Por ejemplo, un televisor hace uso de un *firmware*, y contiene las instrucciones que permiten a sus componentes trabajar junto al sistema operativo. Es el intermediario entre el *hardware* y el sistema operativo. Se almacena generalmente en memorias *ROM* (siglas de *Read-Only Memory*) [25].

***EEPROM*:** es una memoria programable de sólo lectura que puede borrarse al aplicar electricidad de forma adecuada. Retiene su contenido aun sin energía. Acostumbra a requerir de un mayor voltaje para practicarle un borrado [26].

Arquitectura *Harvard*: a granes rasgos, es una configuración de la computadora en la que los datos y las instrucciones de un programa se encuentran en memorias físicamente diferentes, y que se pueden acceder de forma independiente. Los primeros sistemas informáticos (donde las instrucciones del programa podían estar en otro medio, como tarjetas perforadas) son un ejemplo de esta arquitectura. En la actualidad, productos de procesamiento de video y audio utilizan este concepto de arquitectura, así como otros productos basados en chips electrónicos. Es la otra gran arquitectura además de la *von Neumann* [27].

***RISC*:** en la arquitectura computacional, las máquinas *RISC* (siglas de *Reduced Instruction Set Computer*) son microprocesadores/microcontroladores que tienen instrucciones de tamaño dijo y presentan un número reducido de formato, y además solo las instrucciones de carga y almacenamiento acceden a la memoria de datos. El objetivo de estas máquinas es posibilitar la segmentación y el paralelismo en la ejecución de instrucciones, y reducir los accesos a memoria [28].

PCB: es una placa de circuito impreso (siglas de *Printed Circuit Board*) donde se instalan componentes electrónicos y se interconectan entre ellos. Estos componentes pueden ser chips, condensadores, diodos, resistencias, conectores… Para conectar cada elemento en una PCB, se utilizan una serie de pistas conductoras de cobre extremadamente finas. En los circuitos más sencillos solamente tenemos pistas en una o ambas caras de la PCB, pero en otros circuitos más complejos pueden haber múltiples niveles dentro de la PCB, cada uno con sus pistas y componentes apilados [29].

CPI: en la arquitectura de computadores, CPI hace referencia a los ciclos por instrucción, y son una medida del rendimiento de un procesador. Es la inversa del IPC o instrucciones por ciclo [30].

Flag: en términos de interrupciones, un *flag* es un bit que se activa cuando se dan una serie de condiciones (provocadas en *hardware*), y sirve para avisar de un estado del *hardware* concreto. Consultar un *flag* permite saber si ese estado concreto ha tenido lugar, para poder así actuar en consecuencia.

Handler: relacionado con el *flag*, es la rutina que se ejecuta si una interrupción ha tenido lugar. En esta rutina se comparan los *flags*, y si se encuentra uno activado, el *handler* llamará a la función relacionada con dicho *flag* para solventar la interrupción que lo ha provocado.

PWM: es un tipo de señal de tensión usada en electrónica (de las siglas *Pulse Width Modulation*). El *PWM* está formado por una onda cuadrada que no siempre tiene el mismo ciclo de trabajo, ya que depende del tiempo que está en VIH y del tiempo que está en VIL (y la suma de ambos, es el periodo). Variando el tiempo del VIH y VIL conseguimos variar el ciclo de trabajo. Variar el ciclo de trabajo de una señal *PWM* hace que varíe su tensión media, y esta variación provoca que cambie el comportamiento de los componentes que esta señal atraviese [31].

Casting: es un procedimiento para transformar una variable primitiva de un tipo a otro. También se utiliza para transformar un objeto de una clase a otra sierre y cuando haya una relación de herencia entre ambas [32].

Traviesa de la vía: en las vías férreas, son los elementos transversales al eje de la vía, que sirven para mantener unidos y a una misma distancia los dos carriles (rieles) que conforman la vía [33].

Bibliografía

[1] admin. *Modelismo Ferroviario con Arduino.* Dec. 2012. URL: http://www.afergodella.es/modelismo-ferroviario-con-arduino/

[2] Sherlin.xBot. *¿Qué es un microcontrolador?* URL: http://sherlin.xbot.es/microcontroladores/introduccion-a-los-microcontroladores/que-es-un-microcontrolador

[3] Arduino. URL: https://www.arduino.cc/

[4] Gnu. *Licencias - Proyecto GNU - Free Software Foundation.* URL: https://www.gnu.org/licenses/licenses.es.html

[5] Ionos. *¿Qué es Bluetooth? Toda la información sobre el estándar inalámbrico.* July 2020. URL: https://www.ionos.es/digitalguide/servidores/know-how/que-es-bluetooth/

[6] Marklin-spain. *Explora el fascinante mundo de los trenes a escala.* URL: http://www.marklin-spain.com/

[7] Saúl García. *Cómo conectar un Solenoide a Arduino.* June 2020. URL: https://blog.330ohms.com/2020/06/16/como-conectar-un-solenoide-a-arduino/

[8] Microchip. *Microcontrollers and Microprocessors.* URL: https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors

[9] Google Meet – Free video meetings. URL: https://apps.google.com/meet/

[10] BOE. *BOE-A-2020-1626*. Feb. 2020. URL: https://www.boe.es/diario_boe/txt.php?id=BOE-A-2020-1626

[11] Glassdoor. URL: https://www.glassdoor.es/index.htm

[12] Cosima Dannoritzer. *The E-waste Tragedy*. May 2014. URL: https://www.imdb.com/title/tt3804476/

[13] Microchip. *PIC16F15313/23 datasheet*. Sept 2020. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/PIC16_L_F15313_23_Data_Sheet_40001897C.pdf

[14] Microcontroladoressesv. *Microcontroladores PIC y sus variedades*. Dec 2012. URL: https://microcontroladoressesv.wordpress.com/microcontroladores-pic-y-sus-variedades/

[15] Ligo George. *I2C Communication with PIC Microcontroller – MPLAB XC8*. Aug 2019. URL: https://electrosome.com/i2c-pic-microcontroller-mplab-xc8/

[16] Enrico Simonetti. *How to drive a small motor at different speeds using an Arduino*. March 2015. URL: https://enricosimonetti.com/control-a-motor-speed-with-arduino/

[17] Luis Llamas. *El bus I2C en Arduino*. May 2016. URL: https://www.luisllamas.es/arduino-i2c/

[18] Ligo George. *Getting Started with MPLAB XC8 Compiler – LED Blinking*. Dec 2017. URL: https://electrosome.com/led-pic-microcontroller-mplab-xc8/

[19] Vicente García. *El transistor MOSFET*. Nov 2012. URL: https://www.diarioelectronicohoy.com/blog/el-transistor-mosfet

[20] Electronics-tutorials. *Full wave rectifier*. April 2020. URL: https://www.electronics-tutorials.ws/diode/diode_6.html

[21] Digchip. *P24NF10 datasheet*. Aug 2006. URL: https://www.digchip.com/datasheets/parts/datasheet/456/P24NF10.php

[22] Luis del Valle Hernández. *Resistencia pull-up y pull-down*. March 2021. URL: https://programarfacil.com/blog/arduino-blog/resistencia-pull-up-y-pull-down/

[23] Ricardo. *What happens if I omit the pull-up resistors on I2C lines?* Dec 2015. URL: https://electronics.stackexchange.com/questions/102611/what-happens-if-i-omit-the-pullup-resistors-on-i2c-lines

[24] Appinventor. *Explore MIT App Inventor*. URL: https://appinventor.mit.edu/

[25] Dobleclick. *¿Qué es el firmware? y mejor aún, ¿para qué sirve?* Nov 2018. URL: https://dobleclick.eu/que-es-el-firmware-y-para-que-sirve/

[26] Ingeniería y mecánica automotriz. *¿Qué es una memoria EEPROM y cómo funciona?* May 2020. URL: https://www.ingenieriaymecanicaautomotriz.com/que-es-una-memoria-eeprom-y-como-funciona/

[27] Helmut Sy Corvo. *Arquitectura Harvard: origen, modelo, cómo funciona.* Oct 2019. URL: https://www.lifeder.com/arquitectura-harvard/

[28] Arquicom. *Arquitectura CISC vs RISC.* Aug 2016. URL: https://is603arquicom2016.wordpress.com/arquitectura-cisc-vs-risc/

[29] José Antonio Castillo. *¿Qué es una PCB o Placa de Circuito Impreso? Uso, cómo se fabrica.* Feb 2019. URL: https://www.profesionalreview.com/2019/02/11/pcb-que-es/

[30] Wikipedia. *Ciclos por instrucción.* Oct 2020. URL: https://es.wikipedia.org/wiki/Ciclos_por_instrucci%C3%B3n

[31] Enrique Gómez. *¿Qué es PWM y para qué sirve?* Dec 2017. URL: https://www.rinconingenieril.es/que-es-pwm-y-para-que-sirve/

[32] Miguel Bayón Alonso. *Conversión entre tipos primitivos (casting).* April 2013. URL: https://sites.google.com/site/pro012iessanandres/java/conversion-entre-tipos-primitivos-casting

[33] Wikipedia. *Traviesa.* April 2021. URL: https://es.wikipedia.org/wiki/Traviesa