ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

INGENIERO ELECTROMECÁNICO

CONCEPCIÓN DE LA ELECTRÓNICA DE A BORDO DEL VEHÍCULO MOBEE'CITY

Autor: Laura Martínez Fúster
Director: Patrick Perrard

Madrid
Julio 2018

ESTE PROYECTO CONTIENE LOS SIGUIENTES DOCUMENTOS

DOCUMENTO Nº1, MEMORIA

1.1 Memoria pág. 10 a 69: 60 páginas

1.2. Código Fuente pág. 70 a 100: 31 páginas

1.4 Anexos pág. 102 a 111: 10 páginas

DOCUMENTO Nº2, PLANOS

2.1 Lista de planos pág. 1: 1 página

2.2 Planos pág. 2 a 6: 5 páginas

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor D. Laura Martínez Fúster DECLARA ser el titular de los derechos de propiedad intelectual de la obra: CONCEPCIÓN DE LA ELECTRÓNICA DE A BORDO DEL VEHÍCULO MOBEE'CITY, que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente).*

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que

pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.

d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

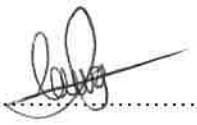
6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 10 de 07 de 2018

ACEPTA



Fdo.....

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Concepción de la electrónica de a bordo del vehículo Mobee'City

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2017-2018 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Laura Martínez Fúster Fecha: 10/ 07/ 2018

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Patrick Perrard Fecha: 15 / 07 / 2018

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

INGENIERO ELECTROMECÁNICO

CONCEPCIÓN DE LA ELECTRÓNICA DE A BORDO DEL VEHÍCULO MOBEE'CITY

Autor: Laura Martínez Fúster
Director: Patrick Perrard

Madrid
Julio 2018

CONCEPCIÓN DE LA ELECTRÓNICA DE A BORDO DEL VEHÍCULO MOBEE'CITY

Autor: Martínez Fúster, Laura.

Director: Perrard, Patrick.

Entidad Colaboradora: École Centrale de Lyon.

RESUMEN DEL PROYECTO

1. Introducción

A lo largo de los últimos años se ha observado un crecimiento en la utilización de vehículos eléctricos. El uso de este tipo de vehículos contribuye a reducir las emisiones de gases contaminantes y a disminuir la dependencia de las energías fósiles. El mundo está experimentando una transición energética, y los fabricantes de automóviles apuestan cada vez más por la tecnología de los coches eléctricos.

Esta tecnología ha sido adoptada por las plataformas de carsharing (vehículos de uso temporal), como por ejemplo Car2Go, que ofrecen una alternativa a los habituales vehículos particulares y a los saturados servicios de transporte público. El éxito que están teniendo estas plataformas es un hecho muy importante en el desarrollo de las ciudades.

El proyecto universitario Mobee'City se une a este movimiento creando su propio vehículo eléctrico, destinado al alquiler, en pequeños entornos urbanos: campus universitarios, ciudades financieras, distritos empresariales etc. El proyecto nació en 2012 en el seno de la antigua Clase de Emprendedores de la École Centrale de Lyon, y se trata de un pequeño coche urbano de motorización eléctrica, 4kW de propulsión, una velocidad máxima de 45 km/h y una autonomía de alrededor de 100km.

Figura 1: Diseño del vehículo Mobee'City

Planteamiento del problema

En los vehículos eléctricos, el sistema electrónico toma un papel esencial, ya que controla los dos elementos más importantes para su conducción: la velocidad y la dirección. La programación del microcontrolador integrado en el vehículo sirve para determinar el ángulo de giro de la rueda directriz y la velocidad de las ruedas motrices garantizando en todo momento la estabilidad del vehículo, evitando pérdidas de tracción.

Para optimizar el control del vehículo, es necesario poner especial atención en la programación del **diferencial electrónico**, un dispositivo que regula la velocidad de las ruedas, permitiendo que puedan girar a distinta velocidad. Esto requiere realizar un estudio matemático que defina la relación entre las velocidades de giro de las dos ruedas motrices y construir un modelo para realizar una simulación del comportamiento del coche.

Mobee'City es un proyecto que se va desarrollando año tras año por distintos alumnos de la École Centrale de Lyon. Este curso 2017-2018 el objetivo es diseñar la electrónica de a bordo del vehículo. Para contribuir realmente al proyecto Mobee'City facilitando su comprensión y continuación, es preciso programar una electrónica que sirva de base para el futuro desarrollo del proyecto.

El vehículo necesita un sistema que transmita las órdenes del conductor a los distintos elementos del vehículo (velocidad, dirección, luces, intermitentes etc.), una interfaz gráfica que informe al conductor del estado de ciertos parámetros (luces encendidas, velocidad en km/h, nivel de batería) y una herramienta que facilite el servicio de mantenimiento del sistema electrónico

desarrollado (un menú gráfico en el que el técnico pueda comprobar el estado de las señales electrónicas).

Estado de la técnica – Arduino

Hoy en día, existe una gran variedad de microcontroladores en el mercado. Hasta el momento, los motores de Mobee'City se controlaban con un micocontrolador PIC de Microchip, presentaba fuertes limitaciones en términos de compacidad y de posibilidades de mantenimiento y evolución. Por ello, Monsieur Perrard, director del proyecto, requería para este curso 2017-2018 un cambio de soporte para la electrónica de a bordo, sustituyéndola por la plataforma Arduino®.

Arduino® es una plataforma de hardware libre y ampliable, basada en una placa con un microcontrolador (Atmega de Atmel) y un Entorno de Desarrollo Integrado basado en *Wiring* y en *Processing*, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares. El IDE y el conocido como *Arduino Programming Language* (C/C++ simplificado) es simple e intuitivo, pero a la vez flexible para usuarios avanzados. Sus ventajas dotan a Arduino® de numerosas aplicaciones industriales (prototipado, domótica, monitorización, adquisición de datos…). Los bajos precios de sus placas y accesorios (shields) y su accesibilidad a todo el mundo, la convierten en la tecnología ideal para utilizar en el desarrollo de Mobee'City. [1]

Objeto del proyecto

El objeto principal es desarrollar un código Arduino® que sirva de base para la continuación del proyecto Mobee'city.

Inicialmente, el objetivo final era diseñar y programar la electrónica de a bordo del vehículo utilizando la plataforma Arduino®. Sin embargo, la École Centrale de Lyon no pudo facilitar el acceso al vehículo real ni al "banco de ensayo" con los motores y demás elementos del coche. Por esta razón, el proyecto adquirió un nuevo enfoque, ya que no se ha podido implantar la electrónica en el modelo real del coche Mobee'City o programar el control de los motores reales.

El proyecto se ha dividido en varias etapas. En primer lugar, se realizó un análisis del vehículo Mobee'City, que incluye un estudio cinemático y la modelización del diferencial electrónico. Después se analizaron con los dispositivos electrónicos necesarios para diseñar la electrónica del vehículo. Posteriormente, se desarrolló un código con Arduino® para controlar los diferentes elementos del coche, la pantalla del puesto de mando y el menú de servicio de mantenimiento. Por último, se realizaron dos ensayos: visualización con una pantalla LCD[1] y la construcción de un "prototipo" que permitió comprobar el funcionamiento de algunas de las "salidas" programadas en la placa Arduino®.

2. Metodología

Análisis – Modelización del diferencial electrónico

El objetivo es modelizar un sistema que reduzca las inestabilidades producidas por pérdidas de tracción que se producen cuando un vehículo traza una curva, ya que la rueda interior recorre una distancia menor que la rueda exterior. Para evitar el deslizamiento de los neumáticos, la velocidad de rotación de cada rueda debe ser diferente, dependiendo del ángulo de giro del coche.

Este proyecto describe el funcionamiento de un diferencial electrónico, programable con Arduino, que permita controlar de forma independiente los dos motores asociados a las ruedas motrices de Mobee'City.

Analizando el modelo cinemático del vehículo Mobee'City, a partir de relaciones geométricas, de las hipótesis del modelo dinámico de Ackerman [2], y de alguna simplificación adicional (se desprecia el ángulo de deslizamiento), se han obtenido las expresiones de las consignas de los dos motores en función de la velocidad global del vehículo (V) y del ángulo de giro (δ) (Ecuaciones 1 y 2), y la expresión de las velocidades angulares de las ruedas (Figura 2).

[1]Liquid Cristal Display / Pantalla de Cristal Líquido

$$\omega_{m_d} = \frac{V}{k_r.R_r}\left(1 - \frac{1}{2}\frac{d}{L}\tan(\delta)\right) \quad (1)$$

$$\omega_{m_g} = \frac{V}{k_r.R_r}\left(1 + \frac{1}{2}\frac{d}{L}\tan(\delta)\right) \quad (2)$$

Donde:

L es la distancia entre las ruedas delanteras y la rueda trasera,

R_r es el radio de las ruedas y

k_r el factor de reducción a la salida del motor.

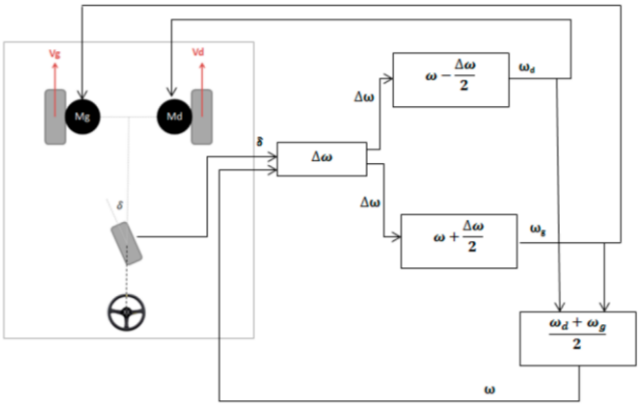


Figura 2: Diagrama ilustrativo del funcionamiento del diferencial

Los dos motores de las ruedas motrices se han modelizado como motores de corriente continua. Utilizando las ecuaciones electromecánicas, eléctricas y mecánicas de un motor CC, se obtiene la siguiente función de transferencia relacionando la tensión de inducido (U) con la velocidad angular (Ω_m):

$$\frac{\Omega_m(s)}{U(s)} = \frac{k_c}{(Js+f)(L_0 s + R_0) + k_e.k_c} \quad (3)$$

Donde:
K_c es la constante del par;
K_e constante de la fuerza contraelectromotriz;
J la inercia del sistema;
f el coeficiente de rozamiento;
R_0 y L_0 la resistencia y la inductancia del inducido

Para poder comparar las velocidades de consigna generadas por el diferencial electrónico con las velocidades angulares reales y obtener el mínimo error posible (diferencia entre la velocidad de consigna y la velocidad obtenida), se introduce un corrector Proporcional Integral Derivativo.

El sistema completo se ha modelizado con Matlab Simulink, y lo componen tres bloques distintos: el bloque del diferencial electrónico y los dos bloques "motores". El primero genera las consigas de velocidad para los motores a partir de la velocidad y el ángulo de giro del coche. Con estas consignas, los dos bloques "motores" calculan las velocidades reales de cada rueda y del coche, que se compara con la consigna de velocidad inicial.

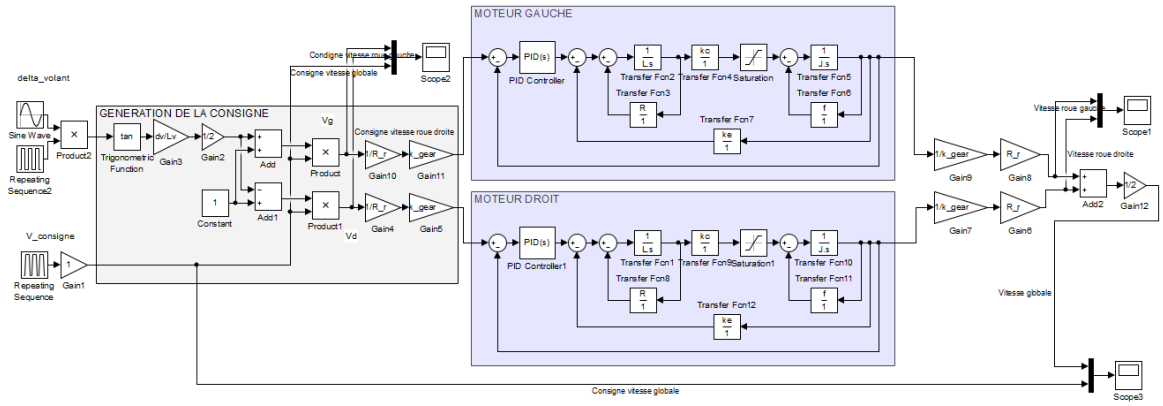


Figura 3: Modelo Simulink completo del control de velocidad de las ruedas

Electrónica

Para diseñar la electrónica del vehículo, es necesario tener en cuenta las características de la placa Arduino® escogida: **Arduino Mega2560** (escogida frente a otros modelos en parte por su elevado número de pines).

- Tensión de alimentación 7 - 12 V, por lo que puede alimentarse con una de las baterías plomo/ácido de Mobee'City (baterías de 12 V en serie).
- 54 pines Entrada/Salida digitales. 14 de estos pines permiten generar señales analógicas con el procedimiento PWM[1] con valores de 0 (0V) a 255 (5V). Para controlar las salidas digitales

[3]Pulse-Width Modulation / Modulación por ancho de pulsos

se utilizan relés, que funcionan como interruptores, que activan el paso de corriente si reciben una señal HIGH (5V) de la salida digital del Arduino y permite aislar eléctricamente el circuito de control (Arduino®) del de potencia (actuadores que utilizan la tensión de las baterías 12 V).

- 16 entradas analógicas, que utilizan un ADC[2], de 10 bits de precisión. La tensión máxima de entrada es de 5 V, mientras que la batería genera hasta 48V. Para medir el nivel de batería, es necesario realizar un divisor de tensión, con tensión máxima de salida inferior a 5V. Por otro lado, se ha impuesto la utilización de optoacopladores en las entradas analógicas, que permiten aislar galvánicamente dos circuitos manteniendo una comunicación entre ellos.

Programación

Se ha desarrollado un código en lenguaje Arduino®. El bucle principal del programa está constituido por una serie de funciones que permiten: detectar el valor de las entradas de la placa (órdenes del conductor), tratar esos valores y almacenarlos en un vector (vector *param)*, y por último controlar las salidas de la placa Arduino en función de esos valores almacenados. Las salidas del Arduino controlan los motores y los accesorios del vehículo (ej.: luces o claxon).

Para mostrar visualmente el estado de ciertos elementos del vehículo como la velocidad en km/h, el nivel de batería o las luces e intermitentes, se ha conectado al Arduino una pantalla LCD 128x64 pixeles. Su programación se ha realizado gracias a la librería de Arduino *U8glib.h*, estableciendo la aparición de un objeto/motivo en un lugar determinado de la pantalla.

Para desarrollar el menú del servicio de mantenimiento, se ha generado un código con el que la placa Arduino comunica el LCD con el pequeño dispositivo de 5 interruptores que el técnico conectará al coche cuando necesite comprobar el estado de alguna de las señales electrónicas (a la entrada o a la salida del microcontrolador). Este código muestra en la pantalla un menú en forma de lista desplegable por el que el técnico puede navegar obteniendo la información que necesita, localizando así fallos en el sistema.

Prototipado rápido

Una gran parte del proyecto se dedicó a la construcción de un pequeño coche teledirigido por Bluetooth desde una aplicación móvil Android. Esto permite mostrar parte de los resultados obtenidos con la programación de una manera muy visual, comprobando que el código controla correctamente algunos de los elementos del vehículo: luces, intermitentes, claxon, diferencial electrónico adaptado a las características del prototipo (motores 6Vcc, batería LiPo 9V).

El desarrollo del prototipo ha supuesto un reto muy interesante, que ha permitido comprobar la utilidad de las herramientas de prototipado rápido (cortadora láser e impresora 3D y softwares como CATIA o Adobe Illustrator), y ha supuesto una investigación en profundidad del funcionamiento de lenguajes de programación como Java (para desarrollar la aplicación en el IDE Android Studio) y la comprensión de la comunicación Bluetooth entre un smartphone y Arduino. Para la programación del prototipo, se utilizó un Motor Shield de Arduino® y sus correspondientes librerías para controlar los dos motores CC y un pequeño servomotor, así como un módulo Bluetooth HC-05. Entre las funciones de la aplicación, destaca la posibilidad de controlar la dirección del vehículo inclinando el teléfono (haciendo uso del acelerómetro).

3. Resultados

Se han realizado simulaciones con el modelo Simulink para una situación concreta (el vehículo gira a la derecha y posteriormente a la izquierda a velocidad constante), y los resultados son satisfactorios. En el ejemplo utilizado el diferencial funciona correctamente, pues la consigna de giro no ha influido en la velocidad media del vehículo y ha aumentado la velocidad de la rueda exterior y disminuido la de la interior (ver Figura 4).

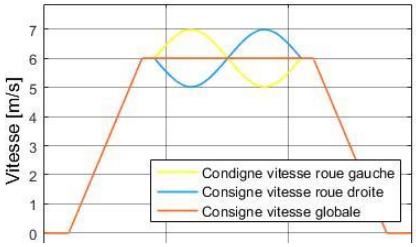


Figura 4: Resultado de la simulación, velocidad de las ruedas y velocidad total

[3]Analog-to-Digital Converter / Conversor Analógico Digital

Por tanto, las ecuaciones 1 y 2 se han utilizado en la programación Arduino para generar las consignas de velocidad de los motores.

Por otro lado, se ha podido comprobar el buen funcionamiento de la programación de la pantalla LCD y del menú de servicio de mantenimiento (Figuras 5 y 6). El diseño de los dibujos ha sido posible gracias al software *TheDotFactory*.

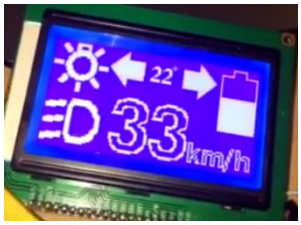


Figura 5 : Resultado LCD, pantala principal

Figura 6 : Resultado LCD, pantalla de inicio

Por último, se desarrolló un prototipo sencillo y una aplicación Android ergonómica e intuitiva, pero con un diseño cuidado, que han permitido comprobar el funcionamiento del código Arduino® desarrollado para controlar algunos de los elementos del vehículo.

Figura 7 : Interfaz de Inicio Mobee'City

Figura 8 : Interfaz de Conducción Mobee'City

Figura 9 : Prototipo

4. Conclusiones

El proyecto cumple el objetivo principal: desarrollo con Arduino® de la electrónica de a bordo del vehículo, que sirva de base para el desarrollo del proyecto y facilite su continuidad y mejora.

La simulación del funcionamiento del diferencial electrónico se ha realizado a partir de un modelo simplificado, suficiente para una programación de base. Sin embargo, el modelo no tiene en cuenta efectos aerodinámicos, ángulos de deslizamiento de los neumáticos, irregularidades en la carretera o resistencia sobre las ruedas. En un futuro, se debe completar esta modelización para programar un sistema que controle la velocidad de las ruedas incluso cuando existan factores externos que modifiquen el comportamiento del vehículo.

La continuación del proyecto requiere profundizar en el estudio del control de los variadores y los motores reales del vehículo (de la rueda motriz y la rueda directriz). Esto será posible una vez se haya instalado el "banco de ensayo", con los elementos reales del vehículo y con un panel de control (formado por interruptores y joysticks) que ha sido diseñado por Monsieur Perrard a lo largo del curso 2017-2018. La adaptación del código y las pruebas realizadas permitirán la posterior implementación en el vehículo Mobee'City.

Por último, con este proyecto se han aplicado conocimientos técnicos y capacidades adquiridas en la formación de grado en ingeniería electromecánica: conocimientos de electrónica, modelización de sistemas, fabricación, capacidad de profundizar en un nuevo entorno de programación (Aplicaciones Android) y utilización de herramientas de organización, de gestión y presentación de proyectos.

5. Referencias

[1] Aprendiendo Arduino [Internet] *Qué es Arduino*. 25 Septiembre 2016. Disponible en: https://aprendiendoarduino.wordpress.com/2016/09/25/que-es-arduino/

[2] Kada Hartani, Mohamed Bourahla, Yahia Miloud, Mohamed Sekour. *Electronic Differential with Direct Torque Fuzzy Control for Vehicle Propulsion System.* 2009

DESIGN OF THE ON-BOARD ELECTRONICS OF THE VEHICLE MOBEE'CITY

Author: Martínez Fúster, Laura.

Director: Perrard, Patrick.

Collaborating Entity: École Centrale de Lyon.

ABSTRACT OF THE PROJECT

1. Introduction

In recent years there has been a significant growth in the use of electric vehicles. Using this kind of vehicles leads to a reduction of contaminating gas emissions and decreases fossil fuel dependency. The world is experiencing an energy transition, and automobile manufacturers increasingly focus on electric vehicles technology.

Carsharing platforms (short-term rental cars) have chosen this technology, such as Car2go, which offers an alternative to ordinary private vehicles and to saturated public transport. The success of these platforms is a very important fact in the development of cities.

The university project Mobee'City joins this movement by creating its own electric vehicle, for rental, in urban environments: university campus, financial cities, business districts etc. The project emerged in 2012 within the former "Classe Entrepreneuriale de l'École Centrale de Lyon", and consists on a small urban car completely electric, of 4kW, a maximum speed of 45 km/h and about 100 km autonomy.

Figure 1: Design of Mobee'City vehicle

Problem statement

The electronic system takes a very important role in electric vehicles, as it controls the two most important elements for driving it: speed and direction. The programming of the microcontroller integrated in the car, determines the rotation angle of the steering Wheel and the speed of each driving wheel guaranteeing stability and preventing loss of traction.

To optimize the control of the vehicle, it is necessary to pay special attention to the programming of the **electronic differential** system, which adjusts the speed of the wheels, allowing them to turn at different speeds. This requires the execution of a mathematical analysis defining the relation between the angular speed of both wheels and to build a model that enables to carry out a simulation of the vehicle's behavior (progress of the speed of each wheel when making a turn).

Mobee'City is a project that is developed year after year by different students from École Centrale de Lyon. This year, the objective is to design the on-board electronics of the vehicle. To really make a contribution to the project, making its comprehension and continuation easier, it is required to program the electronics to serve as a basis for the future development of the project.

The vehicle needs a system that transfers operational commands from the driver to the vehicle elements (speed, direction, headlights, turn signals etc.), a graphic interface that informs the driver of the state of different parameters (headlights on, speed in km/h, battery level) and a tool that helps maintenance service of the developed electronic system (a drop-down menu where the technician can check the state of electronic signals).

State of the art – Arduino

Today, there is a wide range of microcontrollers in the market. So far, Mobee'City motors were controlled by a Microchip PIC microcontroller, which presented compactness, maintenance and

development limitations. Because of this, Monsieur Perrard, project director, required for this year 2017-2018 a change in the support of the on-board electronics, using Arduino® platform instead.

Arduino® uses an easily extended open-source hardware, which consists of a programable board circuit with a microcontroller (Atmega de Atmel) and an Integrated Development Environment based on *Wiring* and *Processing*, design for facilitating the use of electronics in multidisciplinary projects. Its IDE and the *Arduino Programming Language* (simplified C/C++) is simple and intuitive, but also flexible for advanced user. Its advantages give Arduino® many industrial applications (prototyping, automation, monitorization, data acquisition…). The low costs of its boards and shields and its accessibility for all, make it the ideal technology for the development of Mobee'City. [1]

Objectives

The main purpose of the project is to develop a code with Arduino® that serves as a basis for the continuation of Mobee'city project.

Originally, the final goa lof the project was to design and program the on-board electronics of the vehicle using Arduino® platform. However, École Centrale de Lyon could not give access to the real vehicle, the "test bench" with the motors and the rest of the elements. For this reason, the project was given a new approach, as it has not been able to introduce the electronic system in the real vehicle or programming the control of the real motors.

The project was divided in various stages. In the first place, an analysis of Mobee'city vehicle was made, including a cinematics study and the modelling of an electronic differential. After that, the needed electronic devices for the design of the electronic system were chosen. Then, an Arduino® code was developed for controlling the car devices, the command post screen and the maintenance service menu. Finally, two tests were carried out: visualization with a LCD[1] screen and the construction of a "prototype" that allowed to check the operation of some of the "outputs" programmed in the Arduino® board.

2. Methodology

Analysis – Modelling of the Electronic Differential System

The electronic system of the vehicle takes a fundamental role in its stability regulation. The purpose is to model a system that reduces instabilities produced by the loss of traction when the vehicle makes a turn, because the outer wheel travels a bigger distance that the inner wheel. For avoiding slipping of the tyres, the rotational speed of each wheel must be different, depending on the rotation angle of the car.

This project describes the operation of an electronic differential system, programmable with Arduino® that allows the independent control of the motors associated to each driving wheel.

Analyzing Mobee'city vehicle, using geometric relations, Ackerman dynamic model hypothesis [2], and some additional simplifications (no slip angle), it was possible to obtain the following expressions of the commands for the motors depending on the global speed of the vehicle (V) and the turning angle (δ) (Equations 1 & 2), and the expression for the rotational speed of the wheels (Figure 2).

[1]Liquid Cristal Display

$$\omega_{m_d} = \frac{V}{k_r.R_r}\left(1 - \frac{1}{2}\frac{d}{L}\tan(\delta)\right) \quad (1)$$

$$\omega_{m_g} = \frac{V}{k_r.R_r}\left(1 + \frac{1}{2}\frac{d}{L}\tan(\delta)\right) \quad (2)$$

Where:

L is the distance between the fron wheel and the back wheel,

R_r the wheel radious

k_r reduction factor of the motors

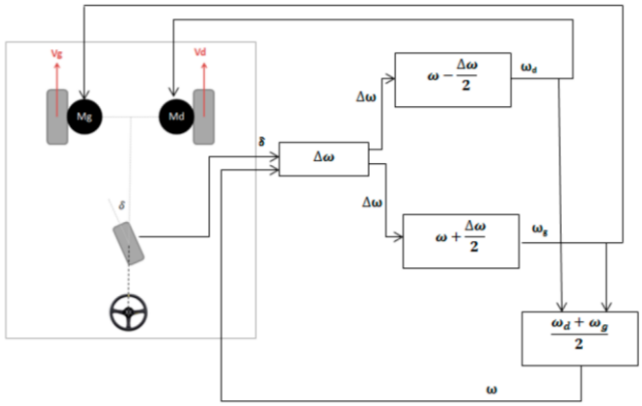


Figure 2: Block diagram show use of the electronic differential

The motors of the driving wheels have been modelled as Direct Current motors. Using electromechanical, electric and mechanical equations for a DC motor, the following transfert fonction is obtained relating the induced voltage (U) and the rotational speed of the motor (Ω_m):

$$\frac{\Omega_m(s)}{U(s)} = \frac{k_c}{(Js+f)(L_0 s + R_0) + k_e.k_c} \quad (3)$$

Where:
K_c: Torque constant;
K_e Back-emf constant;
J Motor Inertia;
f friction coefficient;
R_0 y L_0 motor resistance and motor inductance

To compare the setpoint speed to the speed generated with the electronic differential and minimizing the error (difference between setpoint and generated speed) a PID controller is introduced.

The complete system has been modelled using Matlab Simulink and consists of three blocks: the electronic differential block and two "motor" blocks. The first one generates the setpoint speed for each motor using the driver commands for speed and turning angle. With these speeds, the two "Motor" blocks calculate the real speed for each wheel and the total speed, which is compared to the speed command.

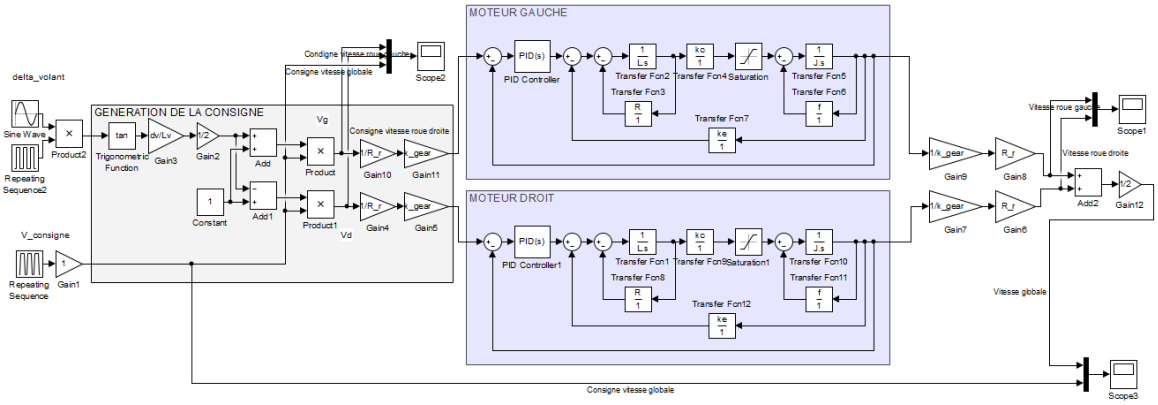


Figure 3 : Complete Simulink model of the control of the speed of the wheels.

Electronics

Designing the electronics of the vehicle, requires considering the main characteristics of the chosen Arduino® board: **Arduino Mega2560** (chosen partially because of its numerous pins).

- Supply voltage 7 - 12 V, so it can be supplied with one of the Mobee'City batteries plomb/acid (series 12 V batteries).
- 54 digital Input/Output pins. 14 of them allow the generation of analog signals using PWM[1] with values from de 0 (0V) to 255 (5V). The control of digital outputs is made with relays, with switch operation, they allow the passage of current if they received a HIGH signal (5V)

[3]Pulse-Width Modulation

and provides electric insulation between the control circuit (Arduino®) and the power circuit (actuators that use 12 C battery supply).

- 16 analog inputs, that include an ADC[2], 10 bits accuracy. Maximum input voltage is 5 V, while battery supply reaches 48 V. Measuring battery level requires a voltage divider, with output voltage under 5 V. Furthermore, analog inputs require the use of optocouplers, that provide galvanic insulation maintaining communication between them.

Programming

A code in Arduino® Programming Language has been developed. The main loop uses a series of functions that allow to: detect the input values of the board (driver commands), treating those values and keeping them in a vector (called *param),* and finally controlling the board outputs according to the kept values. The Arduino outputs control the motors and accessories of the vehicle (e. g. headlights, klaxon).

To show the state of some of the car elements visually, like speed, battery level, headlights or turning signals, an LCD 128x64 pixels has been connected to the Arduino® board. Its programming is based on the Arduino® library *U8glib.h*, allowing to make appear an object anywhere in the screen.

The development of the maintenance service menu is based on a code that uses Arduino® to connect the LCD with a 5-switch device which is used by the technician to check the state of some of the electronic signals (input or output). This code shows a drop-down list menu on the screen, that the technician can use to navigate and obtain the needed information, finding system failures.

Rapid prototyping

A big part of the project was devoted to the construction of a small remote-controlled car an Android App with Bluetooth communication. This allows to show the programming results in a more visual way, testing if the code controls some of the vehicle elements correctly: headlights, turn signal, klaxon, electronic differential adapted to the prototype characteristics (6Vdc motors, 9V LiPo battery).

The development of this prototype has been a challenge, implying the verification of the usefulness of rapid prototyping methods (laser cutting, 3D printing and softwares like CATIA or Adobe Illustrator), and leading to the in-depth investigation of programming languages like Java (for developing the App with Android Studio IDE) and the comprehension of Bluetooth communication between a smartphone and Arduino®. Programming the prototype has requires the use of a Arduino® Motor Shield and its corresponding libraries for motor CC and a small servo-motor controlling, as well as a Bluetooth HC-05 module. Between all the App features, the most outstanding is the possibility of controlling the direction of the vehicle by tilting the phone (using its accelerometer).

3. Results

A particular situation has been used for the Simulink model simulation (the vehicle turns right, then left at constant speed), and satisfactory results have been obtained. In the chosen example, the electronic differential system Works properly, the turning command did not have an influence on the average speed and has increased the outer wheel speed and decreased the inner wheel (refer Figure 4).

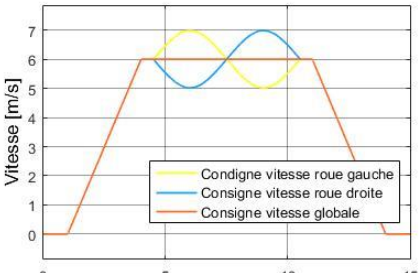


Figure 4 Simulation results: wheels speed and total speed

Hence, Equations 1 & 2 have been used for the Arduino® programming to generate the speed commands for the motors.

[3]Analog-to-Digital Converter

Furthermore, the operation of the LCD programming has been checked, including the maintenance service menu (Figures 5 & 6). The design of the drawings has been made using the software *TheDotFactory*.

Finally, a simple prototype was built and an ergonomic and intuitive, but carefully design, Android App was created. This has made it possible to check the operation of the developed Arduino® code for controlling some of the elements of the vehicle.

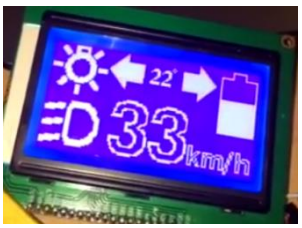


Figure 5 : LCD Result Main Screen

Figure 6 : LCD result Home page

Figure 7 : Home page Mobee'City

Figure 8 : Driving screen Mobee'City

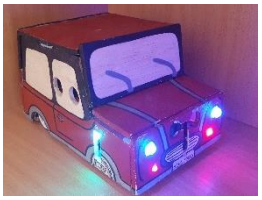


Figure 9 : Prototype

4. Conclusions

The project achieves the main objective: development of the on-board electronics using Arduino®, that serves as a basis for the future development of the project, making its continuation and future improvement easier.

The simulation of the operation of the electronic differential has been created based on a simplified model, enough for basic programming. However, the model does not consider aerodynamic effects, slip angles, floor irregularities or resistance on tyres. In the future, this modelling must be completed to program a system that controls the speed of the wheels even when there are factors that change the vehicle behavior.

The continuation of the project requires looking into the drive control of the real motors (driving and steering wheel). This would be possible once the installation of the "test bench" with the real elements of the car had been completed, including a control panel designed by Monsieur Perrard during the year 2017-2018. Adapting the code and doing the pertinent tests will allow the installation of the electronics in Mobee'City vehicle.

Finally, this project has needed the application of technical knowledge and capabilities acquired during the bachelor's degree in electromechanical engineering such as: electronics, system modelling, fabrication, learning a new programming environment (Android Apps) and using organizational tools, and project management and presentation.

5. References

[1] Aprendiendo Arduino [Internet] *Qué es Arduino*. 25 Septiembre 2016. Website : https://aprendiendoarduino.wordpress.com/2016/09/25/que-es-arduino/

[2] Kada Hartani, Mohamed Bourahla, Yahia Miloud, Mohamed Sekour. *Electronic Differential with Direct Torque Fuzzy Control for Vehicle Propulsion System.* 2009

Agradecimientos

Me gustaría dar las gracias en primer lugar a M. Patrick Perrard, por su ayuda y disponibilidad a lo largo de todo el proyecto.

Gracias al Laboratorio de Tribología y Dinámica de Sistemas de la École Centrale de Lyon (LTDS) que ha hecho posible la realización de este proyecto, ayudando en su financiación.

Doy las gracias también a M. Philippe Timonier por sus consejos en gestión de proyectos y habilidades de comunicación.

Por último, muchas gracias a mi familia por su constante apoyo. Gracias también a mis compañeros, tanto de ICAI como de la Ecole Centrale, que me han acompañado a lo largo de mis estudios de grado, especialmente a Álvaro.

DOCUMENTO 1. MEMORIA

Índice

Índice de figuras

Índice de tablas

Índice de extractos de Código

Glosario

Arduino® Compañía open source y open hardware de fabricación de placas electrónicas. 2, 4, 12, 15, 18, 36, 37, 44, 45, 47, 48, 51, 54, 62, 63

Android Studio Entorno de desarrollo de Android.. 60

DIP Dual In-line Package. 40

IDE Integrated Development Environment. 16, 21, 60

LCD Liquid Crystal Display. 2–4, 7, 42, 45, 49, 52–54, 73–75

NO Normally Open. 46

PIC Programmable Intelligent Computer. 21

PWM Pulse Width Modulation. 36, 37

XML Extensible Markup Language. 60

Parte I:

Memoria

1. Memoria descriptiva

1.1. Introducción

Durante la última década, la población urbana mundial ha pasado del 25 % al 50 %. Más de 3000 millones de personas viven actualmente en ciudades [1]. En este contexto, en el que las carreteras urbanas están congestionadas, la movilidad en las ciudades supone un gran desafío en nuestra sociedad. El carsharing (vehículos de uso temporal) es uno de los sistemas más efectivos que se han desarrollado en los últimos años que ofrecen un complemento a los habituales vehículos particulares y a los saturados servicios de transporte público.

El éxito y el gran crecimiento que están teniendo las plataformas de carsharing es un hecho. Solo basta ver en las grandes ciudades la proliferación de esta nueva movilidad [2]. Sin embargo, ¿cuál es el modelo más interesante para un carsharing? Intentando responder a esta pregunta, surge el proyecto universitario Mobee'City, que nació en el seno de la antigua Clase de Emprendedores de la École Centrale de Lyon (Classe Entrepreneuriale). El proyecto consiste en la concepción de un pequeño coche urbano, de motorización eléctrica, destinado al uso temporal. El objetivo es diseñar un vehículo con una propulsión de 4 kW, con una velocidad máxima de 45 km/h y con una autonomía de alrededor de 100km.

El trabajo de este curso 2017-2018, se centra en el diseño y programación de la electrónica de a bordo de dicho vehículo. El objetivo principal consiste en conectar las instrucciones del conductor con los motores de dirección y de propulsión. Para ello, resulta necesario el diseño de un cuadro de mando, que sirva como interfaz para el conductor. De la misma manera, servirá de interfaz para el mantenimiento del vehículo. La electrónica del coche debe contener la programación del funcionamiento de todos los elementos fundamentales que tiene un vehículo. Sin embargo, la implementación física de la electrónica en el prototipo no está prevista. El objetivo es reproducir la electrónica de un vehículo que posea todas las opciones que un usuario desearía encontrar.

Esta primera parte mostrará en primer lugar las diferentes soluciones que ya existen para el carsharing, así como el estado del proyecto Mobee'City hasta este año. A continuación, se detallarán las motivaciones y los objetivos de este proyecto de Fin de Grado, detallando el pliego de condiciones a seguir.

A día de hoy, la electrónica en los vehículos es una industria muy desarrollada. La electrónica aporta precisión, rapidez, adaptabilidad y fiabilidad. Este proyecto abordará el diseño y la programación de la electrónica del vehículo Mobee'City utilizando tecnología ARDUINO®, caracterizada por su sencilla interfaz de programación y la minimización de componentes electrónicos.

EL vehículo Mobee'City es un vehículo sencillo: debe cumplir las funciones básicas de un coche utilizando el menor número de componentes posible y con poco presupuesto. La electrónica de Mobee'City no busca ser extraordinaria ni destaca por tanto por su complejidad.

El interés del proyecto consiste en la validación de la electrónica desarrollada utilizando un prototipo. Este prototipo no es una réplica a escala reducida, sino un modelo que reúne las características esenciales del coche (servomotor y dos ruedas motrices) y que permite una fácil incorporación de los componentes electrónicos.

Normalmente, el conductor envía instrucciones a los distintos componentes del vehículo a través del volante, el acelerador y los distintos interruptores que forman parte del Puesto de Mando del coche. Para poder simular correctamente esas instrucciones, se ha desarrollado una aplicación móvil Android, que envía señales a los aparatos electrónicos a través de comunicación Bluetooth.

1.2. Estado de la cuestión.

1.2.1. Carsharing

El alquiler de vehículos de manera temporal (conocido como carsharing) ya sean bicicletas, coches o motos, está empezando a coger fuerza en muchas ciudades donde se está implantando debido a que hay un perfil de población que precisa de este tipo de servicio en su vida diaria [3]. Utilizando este sistema, el usuario sólo paga por el tiempo que utiliza el vehículo, la operadora del servicio se encarga de todo lo necesario para el mantenimiento y circulación del vehículo.

Cuando el carsharing incluye vehículos eléctricos en su flota es cuando el servicio adquiere su máximo valor añadido, ya que a los criterios de movilidad sostenible propios de este sistema (cada coche de carsharing elimina alrededor de 15 coches particulares de la circulación), se suma una conducción no contaminante en la propulsión [4].

Son muchas y diversas las plataformas y marcas de vehículos implantadas con éxito y en distintas ciudades. El ejemplo de éxito más conocido es Autolib (desde 2011), en París, donde se realizan desplazamientos bastante largos, y en los que hasta cuatro personas comparten el uso del vehículo como una forma de reducir su precio (el modelo es el Bolloré Bluecar). En Lyon desde 2013 se utiliza un servicio similar, el Bluely (Bluecar, Renault Twizy y Citroën C-Zéro). En otras ciudades como Grenoble, se ha optado por el i-Road de Toyota y el Renault Twizy (ambos con solo 2 plazas). Empresas como Avancar, Car2go o Bluemove son algunos de los operadores que trabajan en España. Car2Go, con presencia en Madrid y en ciudades de otros países (Alemania, EEUU, Canadá o China) ha optado por el Smart ForTwo. [5]

Cada marca prioriza unas características en sus vehículos: el tamaño, la capacidad, la velocidad máxima, la autonomía, la maniobrabilidad, la propulsión o el precio. El vehículo Mobee'City recuerda a algunas de las soluciones ya existentes como el Toyota i-Road o el Twizy, por su capacidad (dos pasajeros), su propulsión (4 kW) y su velocidad máxima (45 km/h).

1.2.2. Electrónica integrada en vehículos

La tecnología es imprescindible en los automóviles. A finales de los años 80 se empezó a hablar de micro-controladores en los coches. Desde entonces la electrónica en los vehículos no ha dejado de desarrollarse. Al principio los dispositivos electrónicos instalados en un

vehículo se podían contar con los dedos de una mano. Si embargo, a medida que iba avanzando la tecnología, esto fue cambiando. Cada vez había más aparatos eléctricos en el coche y el cableado de los vehículos comenzó a hacerse muy complejo, lo que dio lugar a nuevas tecnologías, como la CAN Bus, destinadas a reducir el cableado en el vehículo. Hoy en día, la electrónica supone un 30 % del precio de un vehículo. [6] [7]

La electrónica de un automóvil está formada principalmente por micro-controladores programables (que ocupen el mínimo espacio posible, no se calienten demasiado y consuman la mínima energía posible), sensores (de temperatura, de luminosidad...) y sistemas de comunicación que simplifiquen el cableado (tecnologías CAN y LIN).[6]

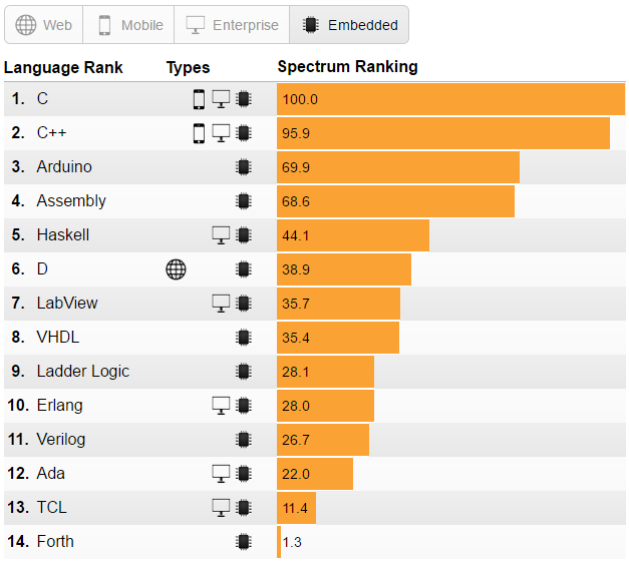


Figura 1: Mejores lenguajes de programación para sistemas integrados según el Institute of Electrical and Electronics Engineers

Según la clasificación realizada en 2016 por el IEEE, representada en la figura 1, los lenguajes más utilizados para la programación de sistemas integrados son el C y el C++. Se observa que hay otros lenguajes como ARDUINO®, Haskell, D, LabVIEW y VHDL que también ocupan puestos importantes en la clasificación. [8]

1.2.3. La tecnología ARDUINO®

El sistema de programación elegido para la realización del proyecto es la tecnología ARDUINO®, la cual presenta numerosas ventajas ante los proyectos elaborados con microcontroladores PIC (de Microchip).

ARDUINO® es una plataforma de hardware libre[1], basada en una placa con un microcontrolador y un entorno de desarrollo, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares.

Por otro lado ARDUINO® nos proporciona un software consistente en un entorno

[1]Dispositivos de hardware cuyas especificaciones y diagramas esquemáticos son de acceso público, ya sea bajo algún tipo de pago, o de forma gratuita.

de desarrollo (IDE[2]) que implementa el lenguaje de programación de ARDUINO® y el bootloader[3] ejecutado en la placa. La principal característica del software de programación y del lenguaje de programación es su facilidad de uso [9].

PIC es un microcontrolador, mientras que ARDUINO® es una plataforma de desarrollo basada en microcontroladores Atmega de Atmel. En el mercado hay muchas placas de desarrollo basadas en microcontroladores PIC, pero tienen un precio más elevado que las placas ARDUINO®. Otra de sus ventajas es que ARDUINO® trae el programador incorporado, mientras que en la plataforma PIC es necesario comprar el programador por separado. [10].

ARDUINO® es una plataforma de desarrollo muy potente debido a sus librerías (que facilitan el desarrollo de múltiples aplicaciones de manera simple y rápida) y a sus Shields (placas que se apilan sobre el ARDUINO® o sobre otros shields, de forma que nos permite ampliar el hardware y las funcionalidades de ARDUINO®).

Otro factor del éxito de ARDUINO® es la gran cantidad de información que existe en Internet, donde hay disponible todo tipo de contenido, de cursos, tutoriales, herramientas de consulta, etc...que facilitan enormemente su utilización.

En la sección 3.2 se describe la placa ARDUINO® elegida para el proyecto: la placa Mega2560.

1.2.4. Aplicaciones móviles

Los dispositivos móviles han supuesto una verdadera revolución en la vida de las personas, especialmente tras la llegada de los smartphones, y con ellos el desarrollo de apps (Aplicaciones), semejantes a los programas que utiliza un ordenador.

Android y iOS son las dos empresas que representan más del 90 % de la cuota de mercado de smartphones. Android domina actualmente la cuota de mercado, superior al 80 % (mientras que Apple iOS se encuentra en torno al 15 %). Con ambas es posible crear aplicaciones nativas, pero de manera diferente.

iOS, de Apple, dispone de su propia manera de desarrollar aplicaciones, con dos lenguajes comunes (Objective-C y Swift, y sus estructuras asociadas).

Android con Google como principal defensor y contribuyente. Diferente de iOS, Android está en open source, lo que significa que su comunidad de desarrollo puede integrar funcionalidades personalizadas y implementarlas en los dispositivos de varios constructores. Android utiliza el lenguaje de programación Java para crear aplicaciones.

En este proyecto, para realizar las pruebas en la electrónica desarrollada en la placa ARDUINO®, se utilizará una aplicación Android. El objetivo será controlar un coche

[2]Aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software. Suele consistir de un editor de código, un compilador, un depurador y un constructor de interfaz gráfica GUI.

[3]Pequeño programa que ha sido guardado previamente en el microcontrolador de la placa y que nos permite cargar código sin necesidad de hardware adicional.

pequeño por radiocontrol desde un dispositivo móvil,haciendo uso para ello de la tecnología inalámbrica Bluetooth.

Se ha elegido Android (basado en Linux) por las ventajas que ofrece que sea una plataforma de código abierto, la facilidad de acceso a los recursos del móvil y las facilidades para la estructuración del código.

Fue desarrollado inicialmente por la empresa Android Inc. a la que Google respaldo financieramente hasta que la adquirió en 2005 decidió comprarla. Mas tarde en 2007, Android pasaría a ser el principal producto de la Open Handset Alliance.

Android utiliza una variación del lenguaje de programación Java. En este lenguaje en el se trabaja con una maquina virtual, que se encarga de interpretar el código que genera nuestro programa y ejecutarlo. [11][12] [14].

En concreto como entorno de desarrollo / Integrated Development Environment (IDE) se ha utilizado Android Studio. En la sección 5.3 se especifica sobre la arquitectura de aplicaciones Android y las características de Android Studio.

1.3. Motivación

El presente proyecto se realiza de acuerdo a la finalidad de la asignatura "Proyecto Fin de Grado" de la titulación de Grado en Ingeniería Electromecánica especialidad electrónica, impartida en la Universidad Pontificia de Comillas (ICAI).

El proyecto se realiza en su totalidad en las instalaciones de la École Central de Lyon, que se encarga igualmente de proporcionar los recursos necesarios. La guía y supervisión corresponden a D. Patrick Perrard, ingeniero de investigación del Laboratorio de Tribología y Dinámica de Sistemas (LTDS) de la École Centrale de Lyon. Con la elección de este proyecto se ponen en práctica los diferentes conocimientos adquiridos en los distintos cursos y asignaturas del Grado.

Este proyecto surge porque la clase de Emprendedores de la École Centrale de Lyon del año 2011 decidió realizar una apuesta por una nueva línea de negocio en el ámbito del transporte que ha generado un gran mercado a nivel mundial, especialmente en grandes urbes: el carsharing.

El 'carsharing' elimina los inconvenientes que presentan los vehículos privados (pago de seguro, mantenimiento...). Por otro lado, supone un acelerador al crecimiento del coche eléctrico. Los coches eléctricos son un medio de transporte sostenible (no emite gases contaminantes). Aunque ya existen numerosas empresas enfocadas a esta modalidad de alquiler y con diferentes modelos de vehículo, este sistema es relativamente nuevo en nuestro país y cuenta con un amplio terreno de mejora y desarrollo.

El objetivo de este proyecto es continuar el trabajo que se lleva realizando varios años en la École Centrale de Lyon. El vehículo Mobee'City es un vehículo caracterizado por su sencillez y desarrollado en su totalidad por alumnos universitarios.

Durante los años anteriores, se han realizado múltiples estudios sobre la dinámica del vehículo, la mecánica (motores, drives, bujía...) y la electrónica.

Un equipo de alumnos trabajó durante el curso 2014-2015 en el estudio de la electrónica del coche utilizando un microcontrolador PIC 16F690 de Microchip y el entorno IDE (Integrated Development Environment) de MPLAB. Este microcontrolador presentaba fuertes limitaciones en términos de compacidad y de posibilidades de mantenimiento y evolución. Es por ello que la principal motivación de este proyecto consiste en remplazar el sistema existente por uno más adaptado: la tecnología ARDUINO®.

Los resultados serán presentados gracias a herramientas de prototipado rápido puestas a disposición por la universidad francesa. Estos resultados, junto con las múltiples ventajas que ofrece la utilización del lenguaje ARDUINO®, facilitarán el posterior desarrollo del proyecto Mobee'City y su continuación el próximo año (implantación de la electrónica en el modelo real).

Formar parte del proyecto Mobee'City supone colaborar con la aceleración de la Transición Energética[4] (siendo el transporte uno de los responsables principales del incremento de los gases de efecto invernadero). Mobee'City es un proyecto emprendedor, solidario con el medio ambiente, realizado por universitarios, que me permite desarrollar mis habilidades de programación y mis conocimientos de electrónica digital y analógica.

1.4. Objetivos del proyecto

El objetivo general del proyecto es el diseño y la programación de la electrónica de a bordo del vehículo Mobee'City. El objetivo específico más importante consiste en el desarrollo de esta electrónica con la tecnología Arduino. Se entiende por electrónica del coche el control de los diferentes elementos del mismo: dirección, aceleración y accesorios.

En concreto, el proyecto se centra en la modelización del diferencial del coche (simulación de la estabilidad del vehículo con Matlab simulink). Esta etapa de modelización es necesaria para la programación de la velocidad de los motores de las ruedas delanteras. Se debe realizar la programación, el cableado y las conexiones necesarias para el control de todos los elementos del vehículo con la placa ARDUINO®.

Debido a que el 'banco de ensayo' no ha estado disponible durante el curso, la implementación de la electrónica en él no estaba prevista. La construcción de un prototipo a escala reducida permitirá evaluar el correcto funcionamiento de la electrónica del vehículo. De esta manera, el ingeniero que continúe el proyecto el año siguiente podrá proceder a realizar pruebas en el 'banco de ensayo' y posteriormente a implementar la electrónica en el vehículo real.

Por otro lado, otro objetivo específico del proyecto es el diseño del Cuadro de Mando, utilizando una pantalla LCD, en el que debe estar indicado el estado de todas las salidas del sistema (luces, intermitentes, velocidad, batería...).

[4]Busca transformar el actual modelo energético hacia un sistema basado en las energías renovables, la electrificación, la eficiencia energética y la generación distribuida.

Al mismo tiempo, y a nivel de formación académica, el trabajo de Fin de Grado tiene como objetivo evaluar los conocimientos adquiridos durante el Grado en Ingeniería electromecánica (electrónica) y tiene algunos objetivos secundarios, como pueden ser:

- Desarrollo de habilidades de gestión de proyecto y de organización.
- Familiarizarse con el entorno de desarrollo ARDUINO®.
- Comprensión de la utilidad de las herramientas de prototipado rápido y su utilización.
- Controlar una conexión Bluetooth entre el microcontrolador ARDUINO® y un dispositivo móvil.
- Aprender a programar aplicaciones para el sistema operativo Android, y conocer los detalles de su arquitectura (nuevo lenguaje de programación: Java)

1.5. Metodología de trabajo

Para establecer el plan de trabajo, se ha optado por realizar un cronograma mediante un diagrama de Gantt (ver Anexo A).

- Una fase de documentación y aprendizaje de los lenguajes de programación. La primera etapa del proyecto consistió en el conocimiento del estado actual del mismo, es decir, analizar los recursos disponibles, el problema a resolver y aportar las soluciones (Ver la sección Análisis 2). Esta etapa de análisis y de gestión del proyecto es fundamental para poder proceder al desarrollo del mismo. Se realizó un análisis del vehículo Mobee'City, que incluye un estudio cinemático y la modelización con Matlab-Simulink del diferencial electrónico. Después se analizaron con los dispositivos electrónicos necesarios para diseñar la electrónica del vehículo.
- Posteriormente, se desarrolló un código con ARDUINO® para controlar los diferentes elementos del coche, la pantalla del puesto de mando y el menú de servicio de mantenimiento.
- Por último, se realizaron dos ensayos: visualización con una pantalla LCD y la construcción de un "prototipo" que permitió comprobar el funcionamiento de algunas de las "salidas" programadas en la placa ARDUINO®.

1.6. Recursos a emplear

La École Centrale de Lyon aporta los siguientes recursos.

Recursos materiales

- Placa ARDUINO® Mega 2560, Arduino Motor Shield, módulo Bluetooth HC-05, pantalla LCD 128x64p.
- Inventario de componentes electrónicos y equipamiento de medición (osciloscopios, multímetros).
- Herramientas y material para prototipado rápido (del Fablab de École Centrale de Lyon): cortadora láser, impresora 3D, madera DM y contrachapado, metacrilato transparente, cartón, poliestireno expandido...

- Aparatos de impresión y proyección

Recursos / Asesoramiento tecnológico

- En modelización de sistemas con MATLAB-SIMULINK®.
- En programación ARDUINO®.
- En electrónica digital y analógica.
- En programación de una aplicación móvil (control a distancia de un prototipo) ANDROID STUDIO®.
- Programas de diseño para el prototipado ADOBE ILLUSTRATOR y CATIA

1.7. Conclusión

Este proyecto se inscribe dentro de uno mayor: el proyecto emprendedor Mobee'City, un pequeño y sencillo vehículo eléctrico de tres ruedas destinado al carsharing en pequeñas zonas urbanas (ej. campus universitario).

La electrónica de a bordo del vehículo debe controlar los distintos elementos electrónicos del vehículo, de acuerdo con las ordenes dadas por el conductor. El objetivo es conseguir un programa en lenguaje ARDUINO® que se pueda implementar directamente en el sistema.

Este proyecto se centra especialmente en el análisis del comportamiento dinámico del vehículo, con el fin de poder programar con ARDUINO® el diferencial electrónico (controlar la velocidad de las ruedas motrices de manera independiente con el fin de asegurar la tracción).

La electrónica debe asegurar que varias acciones se puedan realizar de manera simultánea (acelerar y encender los faros, tocar el claxon y mantener el intermitente encendido etc.). Todas las funciones básicas de un vehículo han sido programadas en la placa ARDUINO®. Junto con Patrick Perrard, ante los problemas de falta de disponibilidad del banco de ensayo, se decidió que la manera más apropiada para realizar los tests y pruebas pertinentes en la electrónica del vehículo, era el desarrollo de una aplicación móvil intuitiva y ergonómica.

2. Análisis

2.1. El vehículo Mobee'City

Mobee'City es un vehículo totalmente eléctrico (movilidad sostenible), pensado para ser un medio de transporte sencillo y fácil de utilizar en pequeños entornos urbanos: campus universitarios, entorno industrial, ciudad financiera etc.

Se trata de un vehículo de 3 ruedas: una rueda directriz trasera y dos ruedas motrices delanteras. Para las ruedas motrices, el vehículo Mobee'city tiene dos motores que pueden ser controlados de manera independientes gracias a un diferencial electrónico. Su función es adaptar la velocidad de rotación de cada rueda en función del ángulo de giro.

Este proyecto se inscribe dentro de uno mayor, y por tanto, los estudio teóricos de la motorización del vehículo, ya han sido realizados por otros alumnos. Este proyecto se apoya en los trabajos que ya han sido realizados para poder programar el diferencial.

Cualquier vehículo eléctrico está provisto de un sistema de baterías. Es fundamental que el conductor conozca el nivel de carga eléctrica del vehículo en cada instante. La elección de las baterías, ya ha sido realizada con anterioridad por otros alumnos: baterías de plomo montadas en serie.

Figura 2: Imagen del vehículo Mobee'City

Mobee'City es un proyecto realizado por varios alumnos de la École Centrale de Lyon en distintos cursos.

Curso 2012: Estudios dinámicos del vehículo: funcionamiento de los motores y los drives, así como las leyes de servo control de los mismos. Se realizaron estudios para calcular el esfuerzo tangencial en la rueda trasera, realizando posteriormente una simulación con Matlab-Simulink. Teniendo en cuenta estos esfuerzos tangenciales, la velocidad de los motores no se adecuaba a la consigna dada por el conductor.

Curso 2013: Estudio de la arquitectura completa del vehículo y validación de las elecciones realizadas por el alumno del año anterior (motores, drives, motor de dirección, baterías).

Curso 2014: Automatización y mecánica del vehículo.

Curso 2015: estudio de la electrónica de a bordo del vehículo. Programación del diferencial electrónico gracias al IDE MPLAB y a un microcontrolador Programmable Intelligent Computer (PIC) 16F690.

Este proyecto se ha centrado en del diseño del puesto de mando, en el que deben aparecer todas las señales necesarias (indicador de batería, velocidad, intermitente..). El director del proyecto no está interesado en la implantación inmediata de la electrónica en el vehículo Mobee'City, sino en demostrar la posibilidad de programar un coche utilizando la tecnología ARDUINO® (más sencilla que la tecnología PIC), que facilite la continuación del proyecto los años siguientes por otros alumnos de ingeniería. Por ello, el interés principal ha sido demostrar la utilidad de las herramientas de prototipado rápido y de control a través de un dispositivo hoy en día al alcance de cualquiera: aplicación móvil.

La siguiente Figura 3 muestra el esquema realizado por alumnos que estudiaron la electrónica del vehículo en años anteriores.

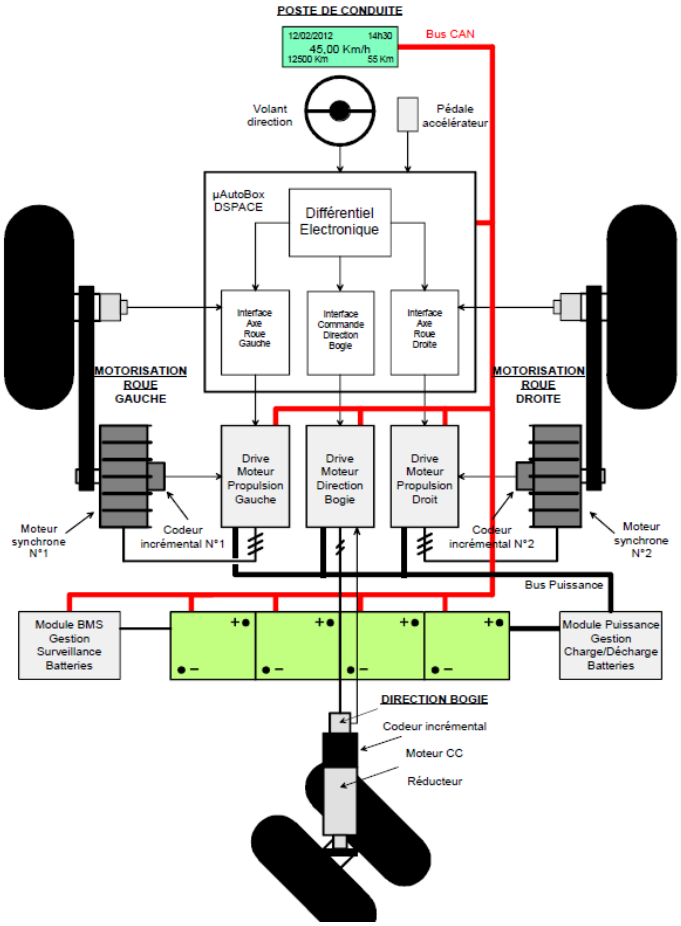


Figura 3: Arquitectura inicial del vehículo Mobee'City

De forma paralela a este proyecto, otro alumno de la École Centrale de Lyon ha realizado un estudio sobre el habitáculo del vehículo.

2.1.1. Los motores de Mobee'City

La dirección del vehículo Mobee'City es completamente eléctrica. Para el control de la dirección del vehículo se eligió un motor CC MAXON. Este motor se caracteriza por un comportamiento idóneo del par, una alta potencia, un rango de velocidades de giro amplio y una larga vida útil. Tiene un reductor planetario de 3 etapas. Como codificador rotatorio, se utiliza uno de tipo incremental de Faulhaber, con el que se obtiene la velocidad y la posición del motor.

Para las ruedas motrices se utilizan dos motores Síncronos-AC del tipo PMS 120, 1500 rpm, 3.2 kW 48 Vcc. Pesa aproximadamente 15 kg, tiene un momento de inercia de 26.3kgcm2 y un par de 20.4 Nm. El proveedor es Perm Motor, que fue comprada por Heizmann. La velocidad/par de estos motores se controla con los drives de tipo ACD 4805 motor controller de Heinzmann (Vcc=48 V).

A principio de curso, los motores Síncronos estaban siendo reparados en Alemania, en Heinzmann, dejando el banco de ensayo indisponible. Ante este incidente, Monsieur Perrard decidió aportar un nuevo enfoque al proyecto, centrándose en el diseño y validación de la electrónica con ARDUINO®, a través de un pequeño prototipo y con la ayuda de una aplicación Android. Esta solución no permite hacer ensayos de la electrónica sobre los motores reales, pero consiste en el desarrollo de un código que servirá de base para el futuro avance del proyecto.

2.2. Modelización del comportamiento dinámico del vehículo

La electrónica del vehículo tiene un papel fundamental a la hora de regular la estabilidad del mismo. Se ha estudiado el comportamiento dinámico del vehículo, basándose en un primer momento en los conceptos de sobrevirage y subvirage.

El sobreviraje es el deslizamiento de las ruedas traseras, mientras que el subviraje es el deslizamiento de las ruedas delanteras. Ambos fenómenos provocan un estado de inestabilidad en el vehículo.

El objetivo de este estudio es modelizar un sistema que reduzca estas inestabilidades: haciendo uso de un diferencial electrónico.

2.2.1. ¿Por qué usar un diferencial electrónico?

Estas inestabilidades se reducen utilizando un diferencial. El diferencial del coche es una de sus piezas fundamentales que ayudan a garantizar la tracción en diversas condiciones (al trazar una curva).

Cuando un vehículo traza una curva (giro), la rueda interior recurre una distancia menor que la rueda exterior (Figura 4). Para evitar el deslizamiento de los neumáticos, la velocidad de rotación de cada rueda debe ser diferente, dependiendo del ángulo de giro. El diferencial es el dispositivo que regula la velocidad de las ruedas, permitiendo que puedan girar a distinta velocidad, facilitando así la adherencia de dichas ruedas al pavimento al realizar una curva.

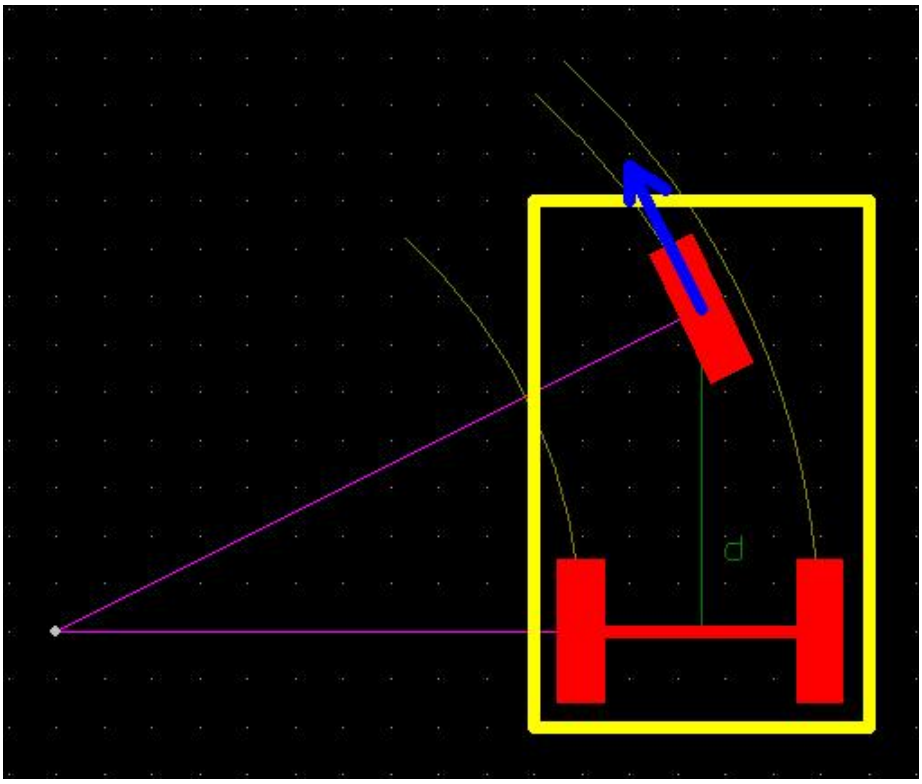


Figura 4: Triciclo trazando una curva. Ruedas motrices con distinto radio de giro. [15]

El vehículo Mobee'City tiene dos motores distintos (con sus respectivos drives), uno para cada rueda motriz. Uno de los objetivos más importantes de este proyecto es describir el funcionamiento de un diferencial electrónico, programable con ARDUINO®, que permita controlar los dos motores de manera independiente.

2.2.2. Estabilidad del vehículo Mobee'City - el diferencial

El objetivo es optimizar el control del vehículo Mobee'City, asegurando su estabilidad.

Para diseñar el sistema diferencial electrónico del coche, el estudio se basa en las hipótesis del modelo dinámico de Ackerman-Jeantaud, descubierto el el siglo XIX por Rudolf Ackerman. EL modelo de Ackerman demuestra que cuando el ángulo de la rueda directriz no es nulo (vehículo girando), la velocidad de la rueda exterior debe ser mayor a la de la interior, ya que recorre mayor distancia.

En un coche convencional (dos ruedas directrices), la geometría de Ackerman define la relación entre el movimiento de las dos ruedas directrices (tienen distinto ángulo de giro) [16]. Numerosos estudios han demostrado que la estructura Ackerman es válida y no introduce demasiados errores a velocidades bajas y con ángulos reducidos de giro. Sin embargo, Ackerman introduce hipótesis que hacen que el modelo no tenga en cuenta efectos aerodinámicos importantes (se generan ángulos de deriva/deslizamiento significativos a velocidades elevadas) [13] [17].

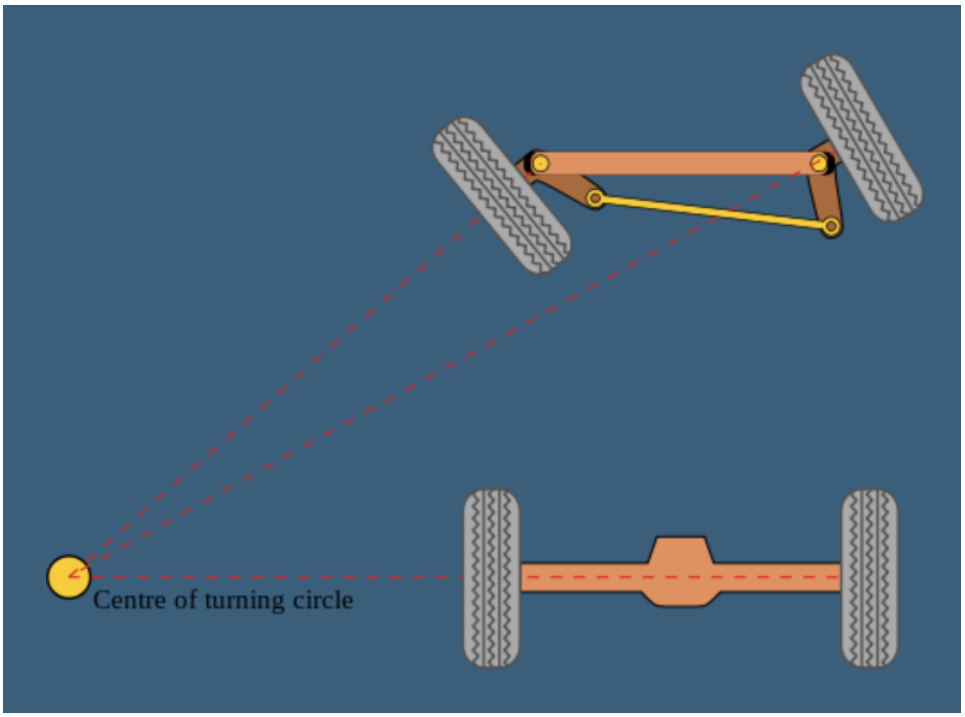


Figura 5: Geometría de Ackerman [13]

Para este proyecto, se tienen en cuenta algunas de esas hipótesis:

- Sólido rígido y sin transferencia de carga (carga uniforme en todo el vehículo).
- Cabeceo y balanceo despreciables.
- Sin efectos aerodinámicos.
- Control de la posición.
- Bajas velocidades (<50 km/h). Fuerzas centrífugas despreciables.

Siendo el objetivo del proyecto desarrollar una electrónica básica, desarrollar un modelo muy complejo no es primordial, ya que además, utilizar un modelo como el de Ackermann necesitaría una potencia de cálculo muy elevado por parte del procesador. Por ello, esta modelización simplificada no tendrá en cuenta el ángulo de deslizamiento de los neumáticos.

El modelo cinemático del coche está representado en la Figura [6]. El CIR de las tres ruedas es el centro de giro del vehículo.

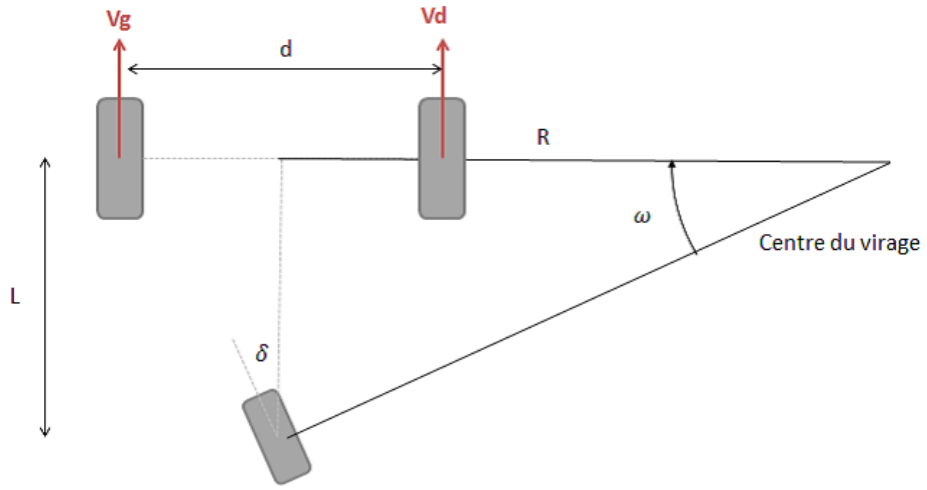


Figura 6: Esquema cinemático del vehículo Mobee'City

Datos :

- R, radio de la curva de giro
- L, distancia entre ruedas delanteras y rueda trasera
- d, distancia entre las ruedas delanteras
- ω, velocidad angular del vehículo alrededor del eje de giro
- δ, ángulo de dirección de la rueda trasera (directriz)
- V, velocidad lineal vehículo (punto situado entre las ruedas motrices)
- V_d, velocidad lineal de la rueda derecha
- V_d, velocidad lineal de la rueda izquierda

También se definen las siguientes variables:

- R_r, radio de las ruedas
- ω_g, velocidad angular de la rueda izquierda alrededor de su eje
- ω_d, velocidad angular de la rueda derecha alrededor de su eje
- ω_{m_g}, velocidad angular del motor izquierdo
- ω_{m_d}, velocidad angular del motor derecho
- k_r, factor de reducción a la salida del motor, tal que $k_r = \frac{\omega_d}{\omega_{m_d}} = \frac{\omega_g}{\omega_{m_g}}$

Se quiere obtener una ecuación que exprese la velocidad de cada motor en función del ángulo de dirección δ y de la velocidad del vehículo V. Para empezar, la ecuación 1 se deduce por relaciones geométricas.

$$\tan(\delta) = \frac{L}{R} \tag{1}$$

Por descomposición de velocidades, por definición de velocidad angular:

$$\omega = \frac{V}{R} = \frac{V_d}{R - \frac{d}{2}} = \frac{V_g}{R + \frac{d}{2}} \tag{2}$$

Combinando las ecuaciones 1 et 2, se deducen las siguientes ecuaciones.

$$V_d = V\left(1 - \frac{1}{2}\frac{d}{L}\tan(\delta)\right) \tag{3}$$

$$V_g = V\left(1 + \frac{1}{2}\frac{d}{L}\tan(\delta)\right) \tag{4}$$

Las velocidades angulares de cada rueda se describen con el sistema:

$$\omega_d = \frac{V_d}{R_r}$$

$$\omega_g = \frac{V_g}{R_r}$$

Por último, puede calculares las consignas de los dos motores gracias a las ecuaciones 5 y 6.

$$\omega_{m_d} = \frac{V}{k_r.R_r}\left(1 - \frac{1}{2}\frac{d}{L}\tan(\delta)\right) \tag{5}$$

$$\omega_{m_g} = \frac{V}{k_r.R_r}\left(1 + \frac{1}{2}\frac{d}{L}\tan(\delta)\right) \tag{6}$$

La diferencia entre las velocidades angulares de las dos ruedas se expresa con la relación:

$$\Delta\omega = \omega_d - \omega_g = \omega\frac{d\tan(\delta)}{L} \tag{7}$$

Cuando el vehículo toma una curva, el conductor impone un ángulo de giro a las ruedas. EL diferencial electrónico actúa directamente en los dos motores, reduciendo la velocidad de la rueda interior y aumentando el de la exterior (ver Figuras 7 y 8):

$$\omega_d* = \omega = \omega - \frac{\Delta\omega}{2} \tag{8}$$

$$\omega_g* = \omega = \omega + \frac{\Delta\omega}{2} \tag{9}$$

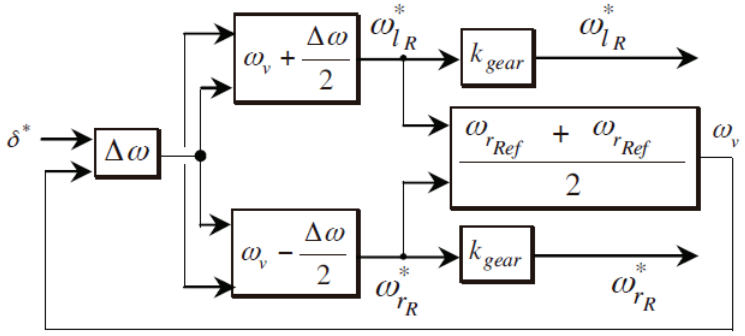


Figura 7: Diagrama de bloques explicativo de un diferencial [18]

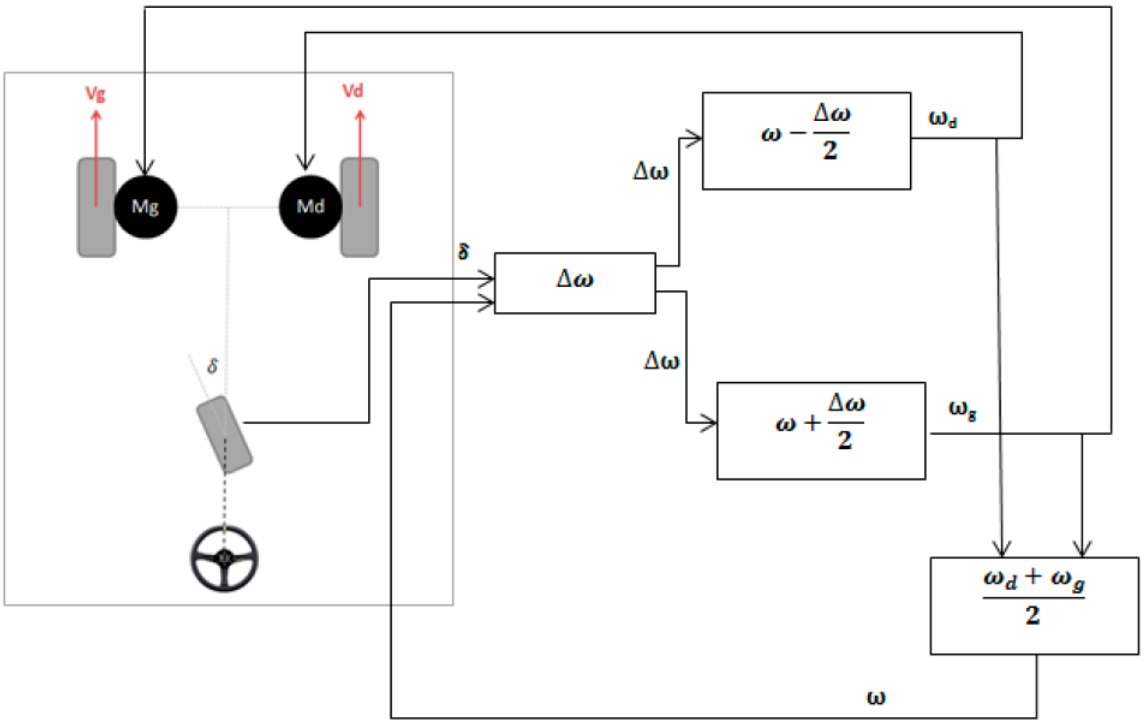


Figura 8: Acción del diferencial en el vehículo Mobee'City

2.2.3. Modelización simplificada de los motores

Para poder realizar una simulación del diferencial de la manera más sencilla posible, se han modelizado los motores síncronos como motores CC. Además, en los ensayos realizados sobre el prototipo, se utilizan pequeños motores CC. Convendrá modificar esta modelización cuando se vaya a implementar la electrónica en el modelo real Mobee'City.

Una de las partes más importantes del motor, el devanado de inducido, consiste en un arrollamiento de varias espiras que puede girar inmerso en un campo magnético constante. Al circular una corriente i(t) por el devanado de inducido se ejerce sobre él un par que es directamente proporcional al campo magnético, y a la corriente de inducido i(t). La Figura 9 muestra un esquema simplificado de un motor CC.

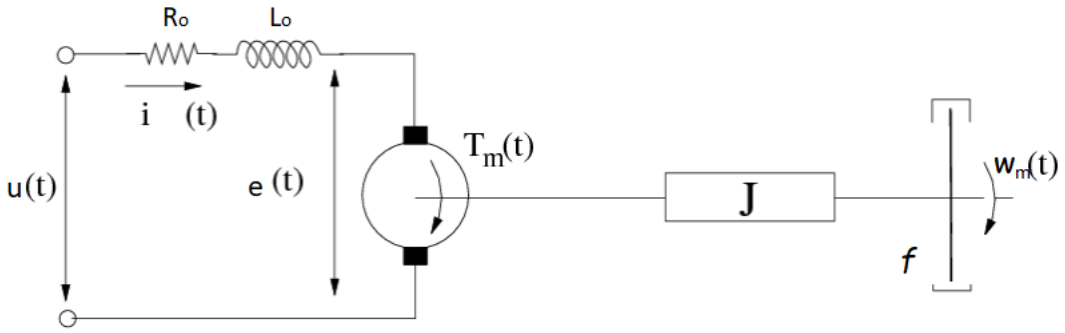


Figura 9: Esquema de un motor CC [19]

Donde:

- R_0, resistencia del inducido,
- L_0, inductancia del inducido,
- $u(t)$, tensión en los bornes del inducido,
- $i(t)$, corriente del inducido,
- $e(t)$, fuerza contra-electromotriz,
- $T_m(t)$, par motor,
- k_e, constante de fuerza contra-electromotriz,
- k_c, constante del par,
- $\omega_m(t)$, velocidad angular del árbol motor,
- J, inercia equivalente del sistema sobre el árbol motor,
- f, coeficiente de rozamiento viscoso.

El sistema se rige por las siguientes ecuaciones.
Ecuaciones electro-mecánicas del motor:

$$e(t) = k_e.\omega_m(t) \tag{10}$$

$$T_m(t) = k_c.i(t) \tag{11}$$

Ecuación eléctrica:

$$u(t) = L_0\frac{di(t)}{dt} + R_0.i(t) + e(t) \tag{12}$$

Ecuación mecánica:

$$J\frac{d\omega_m(t)}{dt} = k_i.i(t) - f.\omega_m(t) \tag{13}$$

Sustituyendo las ecuaciones 2.2.3 y eq2 en 12 y 13 y suponiendo condiciones iniciales nulas, se obtienen las siguientes transformadas de Laplace.

$$U(s) = L_0 s.I(s) + R_0.I(s) + k_e.\Omega_m(s)$$

$$Jp.\Omega_m(s) = k_c.I(s) - f.\Omega_m(s)$$

La función de transferencia que modeliza el comportamiento del motor relacionando la tensión de inducido con la velocidad angular es por tanto:

$$\frac{\Omega_m(s)}{U(s)} = \frac{k_c}{(Js+f)(L_0 s + R_0) + k_e.k_c} \tag{14}$$

El diagrama de bloques correspondiente es el siguiente:

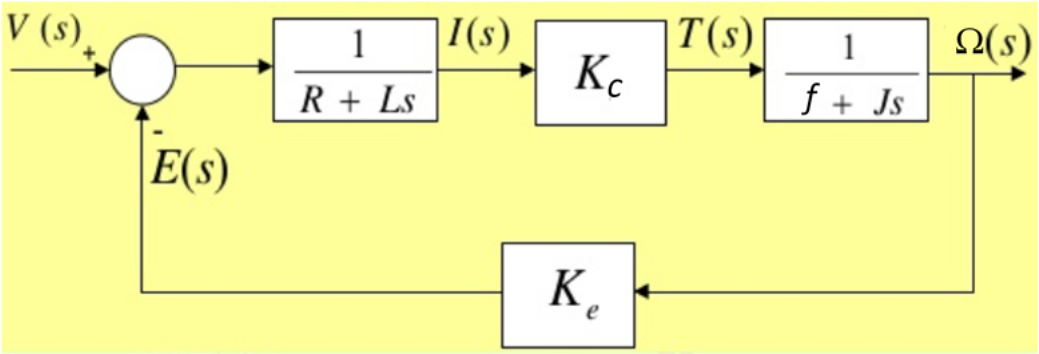


Figura 10: Diagrama de bloques de un motor CC [20]

2.2.4. Control del diferencial - corrector PID

El diferencial trabaja a partir del ángulo de giro y la velocidad de consigna que le da el conductor del automóvil y la transforma mediante las ecuaciones de la sección 2.2.2, para poder comparar las velocidades de consigna generadas con las velocidades angulares reales. El control se realiza calculando el error entre las velocidades reales y las de consigna, empleando un corrector Proporcional Integral Derivativo (PID) que actúa sobre este error. Éste corrector debe ser ajustado de tal manera que actúe de la forma más rápida y eficaz posible para conseguir el mínimo error y obteniendo una respuesta que sea estable. La salida del controlador es la tensión en los bornes del inducido del motor. [21].

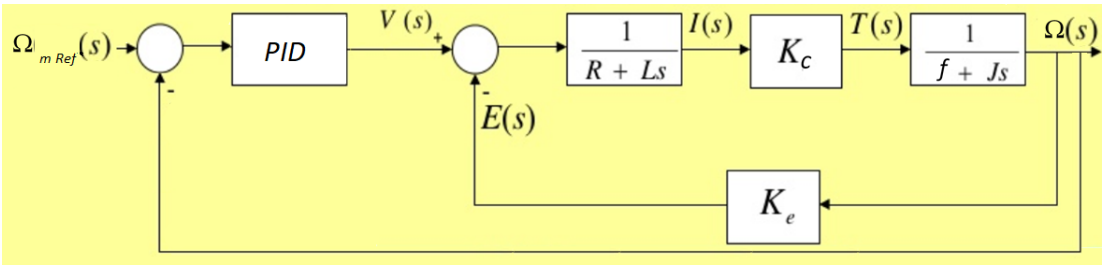


Figura 11: Diagrama de bloques de un motor CC con corrector PID [20]

2.2.5. Simulación del modelo

A partir de los cálculos de los dos apartados anteriores, se ha desarrollado el modelo con Matlab Simulink.

La Figura 12 muestra el bloque que corresponde a los cálculos del apartado 2.2.2, que genera una consigna a partir de la velocidad y del ángulo de giro (órdenes del conductor). Este bloque permite calcular las velocidades angulares de los dos motores (Ω de referencia).

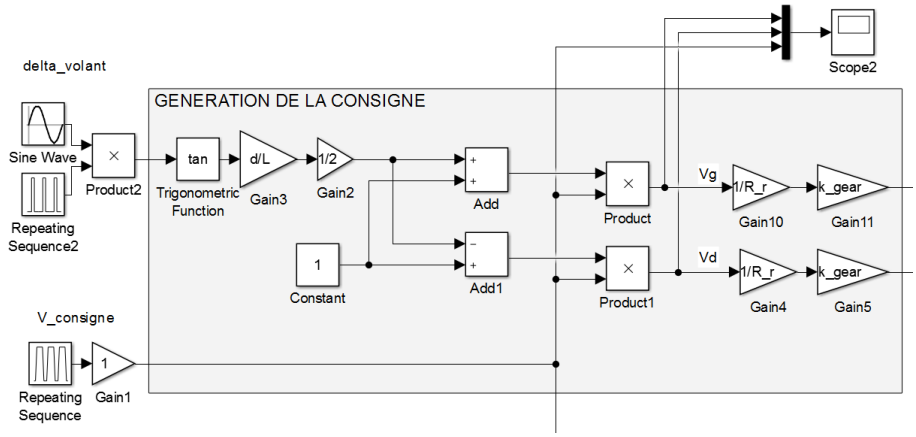


Figura 12: Modelo Simulink Generación de la consigna

La Figura 13 muestra la modelización del motor, tal y como se describió en el apartado 2.2.3 Figura 11.

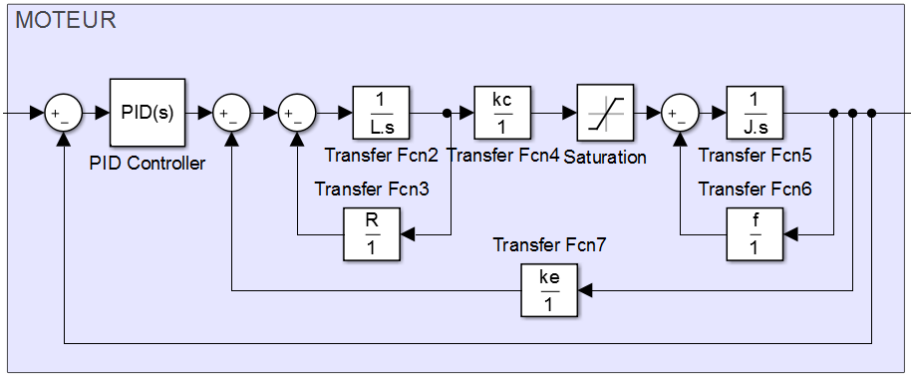


Figura 13: Modelo Simulink Motor

Por último, el sistema completo está representado en la Figura 14. El Bloque "generación de la consignarepresenta la función que debe realizar el ARDUINO®, que realizará el cálculo del diferencial, generando las consignas de velocidad para los dos motores. En resumen:

1. Se envían dos órdenes/consignas (velocidad y ángulo de giro),
2. El primer bloque genera las consignas de velocidad para los motores,
3. Con los bloques "motor"se calculan las velocidades reales,
4. Se calcula la velocidad del coche se calcula y se compara con la consigna.

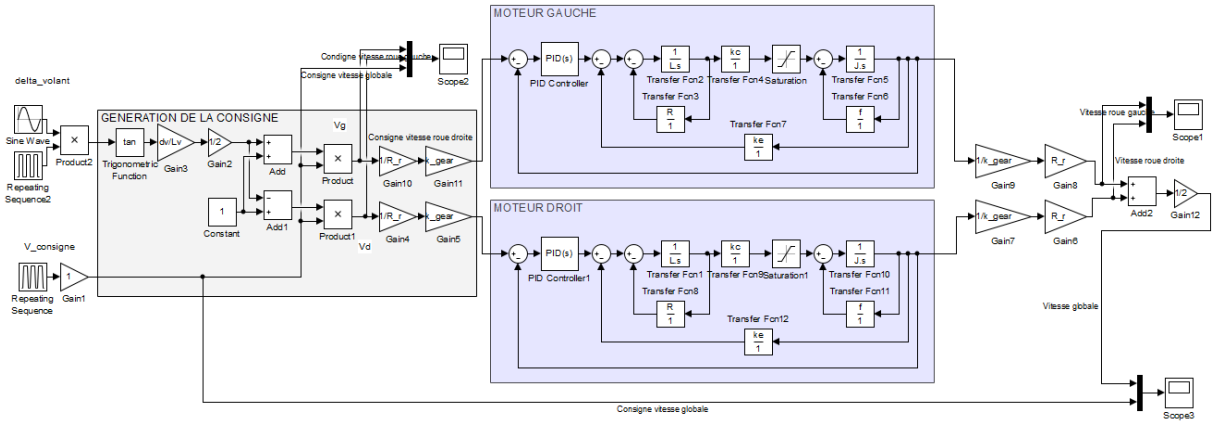


Figura 14: Modelo Simulink completo (ver anexo C)

Para verificar la validez del modelo, se ha definido la forma de las dos entradas. Para la velocidad se ha elegido una forma trapezoidal: el coche acerlera, se mantiene a una velocidad constante y vuelve a disminuir la velocidad (aceleración uniforme). En cuanto al ángulo de giro, se ha elegido una forma sinusoidal durante el periodo a velocidad constante y nula en las fases de aceleración y frenada (giros a la derecha y a la izquierda).

$$V = \begin{cases} 0 & si\ \ 0 < t < 1 \\ 2t - 2 & si\ \ 1 < t < 4 \\ 6 & si\, 4 < d < 11 \\ -2t + 28 & si 11 < t < 14 \\ 0 & si 14 < t < 15 \end{cases} \quad m.s^{-1} \tag{15}$$

$$\delta = \begin{cases} 0 & si\ \ 0 < t < 4{,}5 \\ 0{,}3491 \sin(\frac{\pi}{3}t - \frac{3\pi}{2}) & si 4{,}5 < t < 10{,}5 \\ 0 & si\, 10{,}5 < t < 15 \end{cases} \quad rad \tag{16}$$

La amplitud máxima de V es de 6m.s^{-1} lo que corresponde a 21,6km/h, quna velocidad apropiada para el vehículo Mobee'City (<45kh/h). Además, el ángulo de giro es 0.3491rad, lo que corresponde a un pequeño ángulo de 20°.

Una vez definidas las entradas, se puede observar el resultado en las Figuras 15, 16 et 17 a continuación.

La figure 15 muestra las consignas de velocidad enviadas a los dos motores en función del tiempo. Esas consignas se obtienen a la salida del bloque "generación de la consigna"(Figura 12).Se puede observar que durante los giros, las velocidades de las ruedas se de-sincronizan. EL diferencial actúa únicamente en los instantes en los que el vehículo gira.

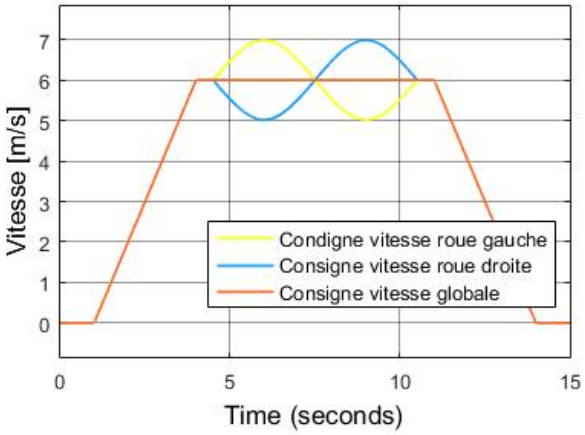


Figura 15: Consignas de velocidad de los motores

La figura 16 muestra las velocidades de las ruedas delanteras.

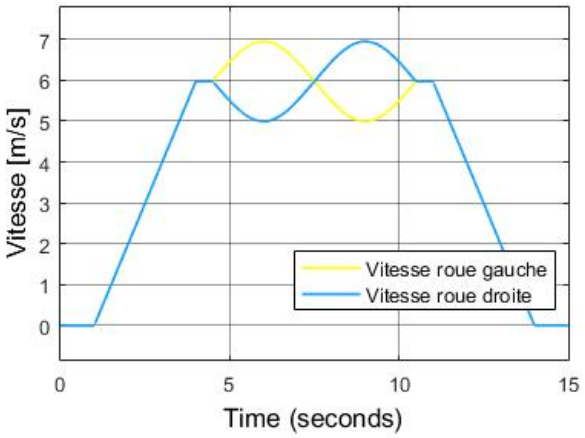


Figura 16: Vitesses des roues

Por otro lado, se ha calculado la velocidad media del coche, tomada en el punto medio entre las dos ruedas, para compararla con la velocidad de consigna.

$$V_{coche} = \frac{V_d + V_g}{2} \tag{17}$$

La figure 17 permite comparar la velocidad calculada y la consigna trapezoidal enviada.

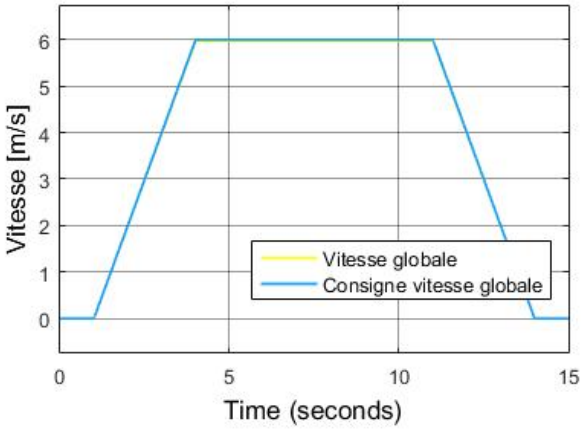


Figura 17: Comparación de la velocidad consigna y la salida de los motores

Esto muestra que el modelo funciona para la situación especifica simulada (el coche aumenta y disminuye la velocidad sin girar y el coche gira con una velocidad constante). Sin embargo para asegurar la validez del modelo e cualquier situación, sería necesario realizar distintos ensayos.

En este ejemplo el diferencial funciona correctamente, pues la consigna de giro no ha influido en la velocidad media del vehículo y a aumentado la velocidad de la rueda exterior y disminuido la de la interior.

En esta parte, gracias a una modelización simplificada de la cinemática y de la motorización del vehículo, se ha podido obtener un modelo teórico del funcionando del coche y además las ecuaciones para modelizar y programar el diferencial electrónico.

Las ecuaciones 5 y 6 se utilizarán en la programación ARDUINO® para generar las consignas de velocidad de los motores.

Finalmente, tras la realización de todas las pruebas pertinentes, se puede concluir que la sintonización del diferencial electrónico realizada no es correcta para todo tipo de situaciones en carretera. Se ha realizado una simulación básica del funcionamiento del diferencial electrónico para una situación determinada sin tener en cuenta varios factores que actúan sobre el comportamiento del vehículo (ver hipótesis 2.2.2).

El sistema descrito sería suficiente para obtener el control diferencial sobre ambas ruedas, pero funcionaría correctamente en condiciones de carretera ideales. Puesto que nuestro vehículo circula por una carretera en la que existen irregularidades en el tipo de suelo, este hecho conlleva que se produzcan pares resistivos diferentes en cada una de las ruedas, los cuales son debidos al deslizamiento y se requerirá de un control de la corriente suministrada a los motores.

Sin embargo, este modelo simple es suficiente para este proyecto, ya que el objetivo principal del mismo es la obtención de un código ARDUINO® que sirva de base para el futuro desarrollo del proyecto Mobee'City. En un futuro se deberá completar y mejorar esta modelización, consiguiendo una simulación que describa con exactitud el funcionamiento del modelo real.

2.3. Optimización de la electrónica de a bordo

Se quiere realizar un cambio de soporte para la electrónica de a bordo. Hasta el momento, los motores se controlaban con un micocontrolador PIC de Microchip. Esto presentaba fuertes limitaciones en términos de compacidad y de posibilidades de mantenimiento y evolución. El objetivo es remplazar el sistema por un módulo mas adaptado: la tecnología ARDUINO® (ver apartado 4.1).

Para realizar los ensayos, los elementos del vehículo (motores, luces etc) serán controlados a distancia con una aplicación Android. Esta aplicación ha sido desarrollada con Andoid Studio, que permite editar los archivos Java y de desarrollo de una Aplicación. (La información sobre Android Studio y los detalles de la aplicación desarrollada están en la apartado 5.3)

3. Electrónica - Pliego de condiciones

3.1. Entradas y salidas del sistema electrónico

El sistema se está compuesto por un conjunto de entradas (volante, conmutadores para las luces etc.) y de salidas (luces, velocidad de los motores etc.). Se debe decidir cómo las distintas entradas van a condicionar el estado de las salidas.

Ejemplo simple: cuando un usuario acciona el conmutador (interruptor) de las luces de posición, estas deben encenderse y al mismo tiempo una señal luminosa debe indicar al conductor que las luces están efectivamente encendidas.

La primera etapa del proyecto consistió en analizar las diferentes entradas y salidas y definir el comportamiento (pliego de condiciones) de estas últimas.

En la Tabla 1 se enumeran las entradas estándar para cualquier vehículo (básicas).

Nombre de la entrada	**Tipo**	**Descripción**
Interruptor iluminación interior	Digital	Enciende la luz del habitáculo
Interruptor claxon	Digital	Accionador la bocina eléctrica del coche
Conmutador luces	Digital	Luces de posición y de cruce
Conmutador intermitentes	Digital	Intermitentes derecho e izquierdo
Sentido de la marcha	Digital	Sentido de rotación de las ruedas
Freno de mano	Digital	Freno del vehículo
Pedal de Freno	Digital	Freno del vehículo
Acelerador	Analógico	Control de la velocidad del vehículo
Sensor de velocidad	Analógico	Sensor de la velocidad de las ruedas
Volante	Analógico	Control de la dirección del vehículo
Conmutador limpia parabrisas	Digital	Puesta en marcha de los limpia-parabrisas
Captor térmico	Analógico	Temperatura en el exterior del vehículo
Presencia del conductor	Digital	Sensor que determina si el conductor está presente
Nivel de batería	Analógico	Determina el nivel de carga de la batería

Tabla 1: Entradas del sistema

Para el desarrollo de este proyecto, debido a una limitación de recursos, no se dispone de las señales de entrada reales del vehículo Mobee'City: señal del volante, carga de batería... Algunas de ellas serán creadas directamente con la placa ARDUINO® y la orden enviada por el conductor a través de la Aplicación Móvil.

En la tabla 2 se enumeran las diferentes salidas del sistema.
El sistema tiene dos estados distintos: *marcha* y *parado*. El coche sólo puede pasar de un estado a otro si se respetan una serie de condiciones: - Conductor presente. - Freno de mano activado. - No se pisa el acelerador.

3.2. Características de la placa ARDUINO® Mega 2560

Como ya se ha comentado en la Sección 4.1, la utilización de ARDUINO® presenta numerosas ventajas.

Nombre de la salida	Tipo	Descripción
Iluminación habitáculo	Digital	Enciende la luz del interior del coche
Claxon	Digital	Bocina eléctrica de los automóviles
Luz placa matrícula	Digital	Iluminación de la placa de matriculación
Limpia-parabrisas	Digital	Encender/apagar limpia-parabrisas
Luces de posición	Digital	Luces de posición
Luces de cruce	Digital	Luces de cruce
Intermitente derecho	Digital	Intermitente derecho encendido/apagado
Intermitente izquierdo	Digital	Intermitente izquierdo encendido/apagado
Luz de marcha atrás	Digital	Luz que se enciende cuando el vehículo da marcha atrás
Luz de freno	Digital	Se enciende cuando se pisa el pedal de freno
Pantalla LCD	Analógico	Pantalla en el cuadro de mando
Motor derecho	Analógico	Motor que controla la rueda derecha
Motor izquierdo	Analógico	Motor que controla la rueda izquierda
Motor de dirección	Analógico	Motor que controla la rueda directriz

Tabla 2: Salidas del sistema

De la gama de modelos que presenta ARDUINO®, se ha elegido el ARDUINO® Mega 2560 (Figura 18 y pinout en Anexo D). El criterio para la selección de la tarjeta ARDUINO® ha sido el del número de pines necesarios para el sistema elecrónico del vehículo. Tiene las siguientes características [22]:

- Microcontrolador ATmega2560.
- Voltaje de entrada de – 7-12V. Se puede alimentar con conexión USB o con una fuente externa (baterías). Si se elige alimentar la placa con una fuente externa, en vez de la conexión USB, el adaptador debe tener un conector de centro positivo y diámetro de 2.1mm para poder conectarlo a la clavija tipo Jack.
- 54 pines digitales de Entrada/Salida (14 de ellos son salidas Pulse Width Modulation (PWM)). Operan a 5 voltios. Cada pin puede proporcionar o recibir 20 mA como condición de funcionamiento recomendada y tiene una resistencia de pull-up (desconectada por defecto) de 20-50 k ohmios. Un máximo de 40 mA es el valor que no debe superarse para evitar daños permanentes en el microcontrolador.
- 16 entradas analógicas.
- 256k de memoria flash.
- Velocidad del reloj de 16Mhz: Esto hace un total de 16 millones de operaciones por segundo, lo que dará la rapidez suficiente para poder realizar el control de la máquina.
- Trabaja con señales entre 5V y 0V, siendo los valores entre el rango 2,5V - 5V el estado de HIGH y 0V - 2,5V el de LOW, y cuenta con varios pines que nos proporcionan las tensiones (5V y GND).

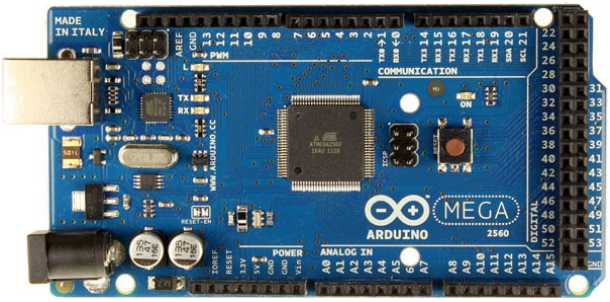


Figura 18: Placa ARDUINO® Mega2560 [9]

3.2.1. Las entradas analógicas en ARDUINO®

En las 16 entradas analógicas que tiene la placa ARDUINO® Mega, se utiliza un Convertidor Analógico-Digital (ADC). Los valores de tensión de entrada admisibles están entre 0 y 5V. Las entradas analógicas disponen de 10 bits de resolución, lo que proporciona $2^{10} = 1024$ niveles digitales (valores entre 0 y 1023), lo que a 5V supone una precisión de la medición de +-2,44mV (precisión global de 4,8 mV). Esta precisión es suficiente para nuestro modelo.

3.2.2. Generación de señales analógicas con ARDUINO®

El procedimiento para generar señales analógicas es el llamado PWM o modulación por ancho de pulso, que permite sintetizar señales continuas a partir de señales digitales. Para ello se utiliza el valor conocido como "Duty cycle", Ciclo de Trabajo, el cual determina el porcentaje de tiempo que el pulso (o voltaje aplicado) está en estado activo (HIGH) durante un ciclo.

$$duty\ cycle = \frac{tiempo\ que\ la\ salida\ est\ a\ 1\ o\ HIGH}{periododelafuncin}$$

Las señales PWM son comúnmente usadas para el control de velocidad de motores DC. En la placa ARDUINO® Mega la frecuencia de la señal es de 490 Hz aproximadamente y permite cambiar el "duty cycle.º el tiempo que el pulso está activo (HIGH), utilizando la función analogWrite(). Los valores de la salida PWM van desde 0 para 0V (0 %) hasta 255 para 5V (100 %).

A continuación se muestran ejemplos de generación de señales con analogWrite().

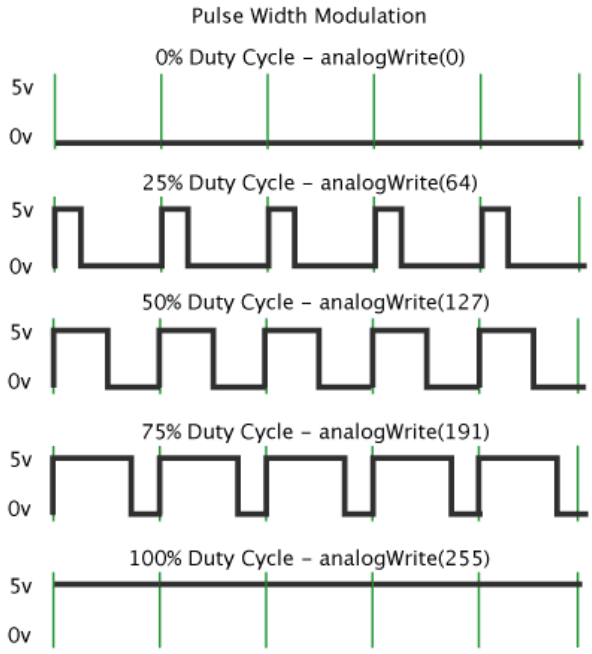


Figura 19: Ejemplos señales generadas con analogWrite().[9]

3.3. Dispositivos electrónicos e implementación física

Optoacopladores

Un optoacoplador es un dispositivo electrónico que permite aislar dos circuitos manteniendo una comunicación entre ellos. La aplicación principal es en aislamiento entre los circuitos de control y los de potencia. Se suelen utilizar para aislar dos circuitos, uno que trabaja a poco tensión (en este caso el ARDUINO®, que no trabaja a más de 5 V), llamado de control y otro a mucha tensión o a una tensión diferente (48 V emitidos por la batería para el funcionamiento de los motores) llamado de potencia.

La función de este circuito en el proyecto es adaptar y acomodar la señales analógicas y de los interruptores del Puesto de Mando del coche (entrada al sistema), a la señal máxima admisible por el ARDUINO® 5 V. En otras palabras, se usará dicho componente como un reductor de tensión que permite proteger la electrónica del vehículo (evitando sobretensiones y de este modo posibles daños irreversibles).

Su funcionamiento se basa en un circuito integrado muy básico compuesto por un diodo LED y un foto-transistor unidos de tal forma que cuando una señal eléctrica circula a través del LED, hace que brille y la luz emitida es recibida por la base del foto-transistor (receptor de luz). Este al recibir esta señal luminosa genera en sus bornes una tensión eléctrica, que será la tensión de salida. Como la luz que emite el LED varía en función de la tensión y la corriente que circulan por él, y esta luz a su vez modifica el comportamiento del transistor, la señal eléctrica de salida depende de la señal de entrada.

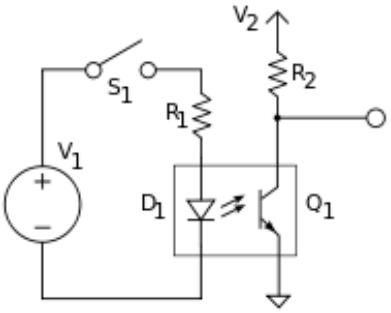


Figura 20: Esquema optoacoplador. [13]

Existen distintos tipos en el mercado según los dispositivos de salida: fototransistor, fototriac o fototriac de paso por cero. La mayor parte de los optoacopladores utilizan un encapsulado llamado DIP (Dual in-line package), bloque con dos hileras paralelas de pines).

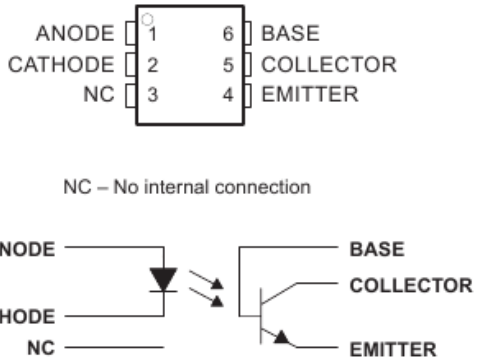


Figura 21: Esquema pines optoacoplador [23]

Se ha propuesto la utilización de optoacopladores ILD615 de 4 canales (Figura 22), que ofrece una tensión de Aislamiento de hasta5,3 kVrms, y ayuda a ahorrar espacio en el montaje de la electrónica.

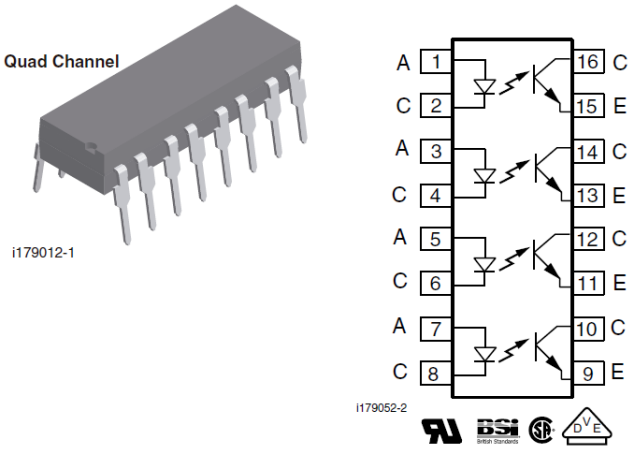


Figura 22: Optoacoplador ILD615 [25]

Relés

Los controladores de los motores [24], funcionan con tensiones entre 12 y 48 V, las cuales no pueden generarse con la placa Arduino (tensión limitada a 5 V), y necesitan la tensión provista por la batería. Por ello, se necesita un dispositivo electrónico de control: los relés.

Un relé es un dispositivo electromecánico que permite la conmutación de una línea eléctrica de media o alta potencia a través de un circuito electrónico de baja potencia. La principal ventaja es que la línea eléctrica está completamente aislada de la parte electrónica que controla el relé.

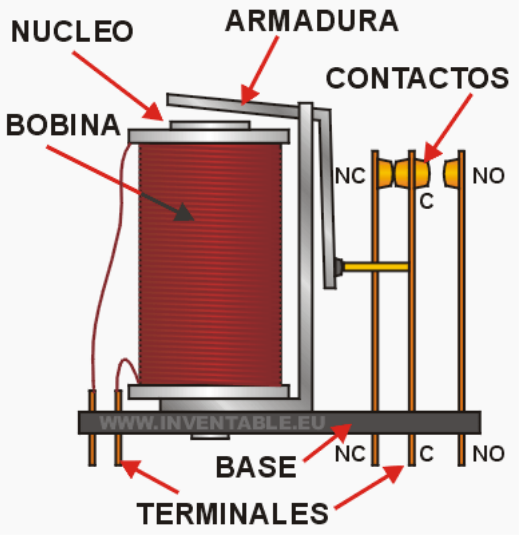


Figura 23: Esquema relé. [26]

Un relé esta compuesto por una bobina, una armadura metálica (fromando un electroimán) y un grupo de contactos que pueden ser conmutados a través de un campo magnético generado por la bobina.

La bobina se conecta a la electrónica de baja tensión, en nuestro caso ARDUINO®, y recibe la señal de encendido y apagado. Esta señal en la bobina crea un campo magnético. Este campo magnético atrae la armadura que, acercándose al núcleo de la bobina, mueve los contactos del relé efectuando la conmutación y cerrando el circuito secundario (de potencia).

Los relés funcionan por tanto como interruptores, que activan el paso de corriente si reciben una señal HIGH (5V) de la salida digital del Arduino. Se utilizarán relés DIP.

Alimentación del ARDUINO®

Como se vio en el apartado 3.2, la placa Arduino Mega se alimenta con un voltaje de entre 7 y 12 V.

El modelo real Mobee'City, según los informes realizados en años anteriores por alumnos de la École Centrale de Lyon, utiliza 4 baterías de 12 V en serie (48 V de tensión

necesaria para los motores). Podemos utilizar los terminales para alimentar la placa a través del jack de alimentación externa. [27].

Medición del nivel de batería

Según los informes realizados los años anteriores en la École Centrale de Lyon, las baterías de Mobee'City suministran 48 V (max). Es fundamental medir la tensión que tiene la batería en todo momento. Sin embargo, ARDUINO® admite una tensión de entrada máxima de 5 V, por tanto, para controlar el nivel de carga de la batería es necesario adaptar las tensiones para evitar sobretensiones que dañen el microcontrolador.

Es necesario utilizar un dispositivo que disminuya la tensión como mínimo a 5 V. Se ha optado por un divisor de tensión: repartir la tensión de alimentación entre una o más impedancias en serie.

Se debe intentar que:

- Los valores de impedancias ser muy superiores a la impedancia equivalente del dispositivo a medir.
- Sean despreciables respecto a la impedancia de la entrada ARDUINO®.
- Limiten la corriente que circula por ellas para minimizar pérdidas.
- Deben ser capaces de disipar la potencia que van a soportar [28]

A continuación se muestra un esquema del divisor de tensión.

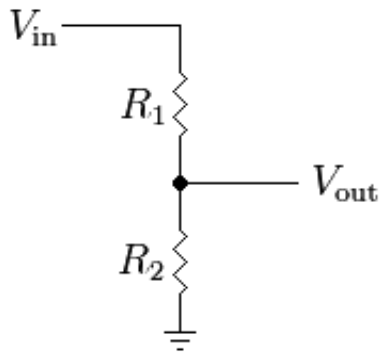


Figura 24: Esquema eléctrico divisor de tensión

Donde:
$V_{in} = 48V$
Y se quiere obtener: $V_{out} = 5V$ Sabiendo que:

$$V_o ut = \frac{R_2}{R_1 + R_2} \tag{18}$$

Se han calculado los siguientes valores para las impedancias:
$R_2 = 22k\Omega$ $R_1 = 190k\Omega = 180k\Omega + 10k\Omega$

Con estos valores, si V_{in} es máxima (48V) V_{out}=4,98V.

Es importante que el nivel de la batería no alcance valores muy bajos, evitando que se produzcan descargas completas que produzcan daños en la misma. Por ello, impondremos que la batería está al 0 % cuando V_{in}= 38.4 V, es decir cuando la entrada al ARDUINO® es de 4 V. Esta mínima tensión será una de las condiciones que se deben cumplir para arrancar el vehículo (pasar del estado *parado* a *marcha*).

Panel de control : pantalla LCD

El conductor del vehículo debe conocer cierta información básica sobre el estado del vehículo: nivel de batería, velocidad, estado de los intermitentes y de las luces.

El dispositivo gráfico elegido para transmitir esta información es una pantalla de cristal líquido o LCD de 128x64 pixeles (ver Figura 25).

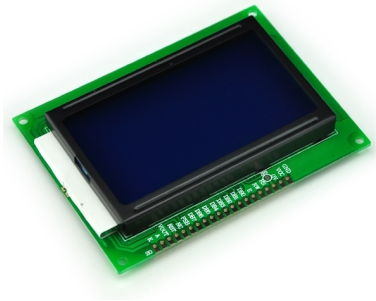


Figura 25: Imagen pantalla LCD 128x64 pixeles

El dispositivo se conecta a tres salidas de la placa Arduino, que se contectan a tres pines del LCD: SID (Data Pin), CS (latch pin), y SCK (clock pin). El esquema de la Figura 27 muestra la conexión del LCD con la placa. La programación de la misma se explicará en el apartado 4.1.2.

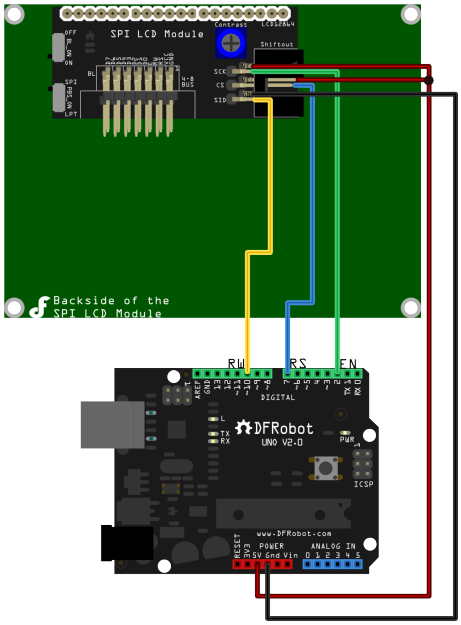


Figura 26: Conexión entre la pantalla LCD y la placa Arduino

EL display LCD aporta una gran cantidad de información en tiempo real, muy útil para la supervisión del funcionamiento del sistema de control elegido.

Servicio de mantenimiento

Para las medidas de mantenimiento y de corrección preventiva, es necesario que un técnico pueda acceder a la información del vehículo, es decir, conocer el estado de los distintos componentes electrónicos y identificar donde hay un fallo o problema. Por ello, se a integrado un *Menú de Mantenimiento*en el sistema.

Se trata de un menú sencillo, con el que el técnico de mantenimiento podrá conocer si, por ejemplo, el intermitente derecho no se enciende, si se trata de un problema en la lectura de la orden, o bien en el circuito de salida (en el que está la bombilla).

El técnico conectará al ARDUINO® un pequeño dispositivo con cinco pulsadores: `izquierda`, `derecha`, `arriba`, `abajo` y `OK`. El menú mostrará las opciones en la pantalla LCD. A continuación se muestra un esquema del dispositivo de mantenimiento:

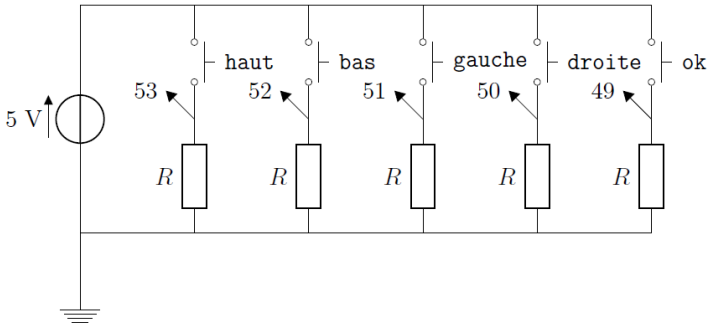


Figura 27: Esquema dispositivo de mantenimiento

Se alimenta con la placa ARDUINO®, y cada entrada es leída por un input digital del ARDUINO®. Se detallará la programación en el apartado 4.1.2.

4. Programación

En este apartado se explicarán los lenguajes de programación y los softwares empleados, así como el funcionamiento de las principales líneas de código.

4.1. Arduino®

4.1.1. Programar en Arduino®

Para programar el microcontrolador, Arduino® tiene su propio entorno de desarrollo integrado (IDE) que se obtiene de forma gratuita desde la página oficial, aunque también se pueden utilizar otras herramientas como un editor de texto y un programador de microcontroladores. Arduino® utiliza un software libre basado Wiring y en el IDE de Processing, utilizando un lenguaje simplificado de C/C++ (Arduino® Programming Language).

Arduino simplifica el proceso de trabajar con microcontroladores ya que el entorno de programación de Arduino es intuitivo pero suficientemente flexible y potente para usuarios avanzados.

Arduino® proporciona un entorno de programación sencillo y potente para programar, pero además incluye las herramientas necesarias para compilar el programa y "quemar" el programa ya compilado en la memoria flash del microcontrolador. Además el IDE nos ofrece un sistema de gestión de librerías y placas muy práctico.

Un programa de Arduino® se denomina sketch o proyecto y tiene la extensión .ino. El software incluye las herramientas necesarias para compilar el programa y cargarlo en la memoria del microcontrolador (herramientas para programar los microcontroladores AVR de Atmel son avr-binutils, avr-gcc y avr-libc) (9). El lenguaje puede ampliarse a través de las numerosas librerías que ofrece la plataforma.

Los sketchs contienen siempre dos funciones principales `setup()` y `loop()`. La función `setup()` se establece cuando se inicia un programa. Se emplea para iniciar variables, establecer el estado de los pins , inicializar librerías, etc. Se ejecuta una vez al conectar la placa o reiniciarla. Mientras que la función `loop()` se repite de manera iterativa, controlando de forma activa la placa Arduino. Las funciones que se definen en Arduino indican el tipo de variable que devuelven: int (entero), float u otras variables admisibles en C, o void si no devuelve ningún valor [29][30].

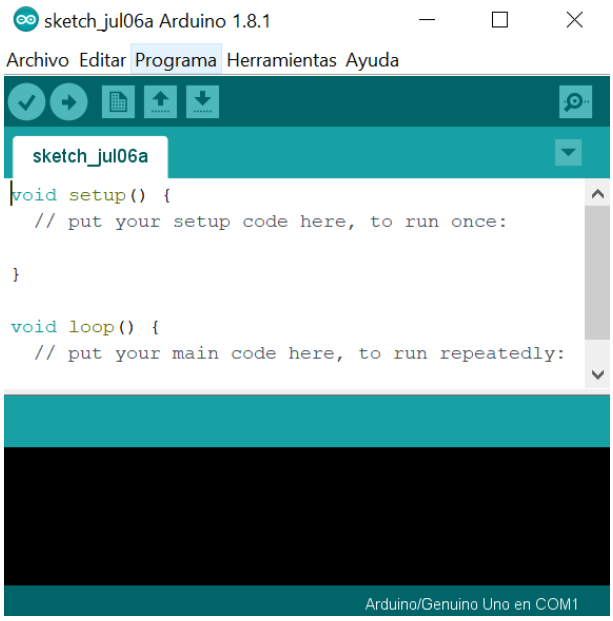


Figura 28: IDE de ARDUINO®, funciones `setup` y `loop`

Una de las herramientas más utilizadas a la hora de depurar el funcionamiento del programa es el Monitor Serial, que nos permite comunicarse mediante el USB de nuestro PC y ver mensajes de la ejecución que hayamos programado.

4.1.2. Estructura General del programa

El programa se basa en la explotación continua del vector `param`, que contiene 20 parámetros. Una lista explicativa de todos los parámetros se encuentra en el Anexo E. El código se estructurará en diferentes funciones, algunas de las cuales actualizarán/modificarán dicho vector.

El bucle principal

La programación de ARDUINO® *Poste de Pilotage / Puesto de Mando* se divide en varias funciones.

- función `read_values()`: actualización de todas las entradas
- función `write_values()`: actualización de todas las salidas
- función `conditions()`: verificación de condiciones según el estado del coche: arranque, marcha, parado, "modo mantenimiento"
- función `draw()`: actualización del LCD
- función `tecnic_state()`: define el control del "modo mantenimiento"
- función `comunication_Bluetooth()`: se encarga de la comunicación Bluetooth con la aplicación móvil

La placa ARDUINO® lee la función `loop()` una y otra vez, llamando a las funciones.

Actualización del vector `param` : la función `read_values()`

El vector `param` se actualiza gracias a la función `read_values()` la cual utiliza a su vez la función `check()`.

La función `check()` tiene en cuenta tres parámetros :

- `num_param` número del parámetro será modificado del vector `param`
- `pin` pin asociado a esa entrada
- `val_on` el número correspondiente a la activación de un estado[5].

La función `check()` lee, gracias a la función `digitalRead()`, el valor binario del pin n°`pin` seleccionado. La programación se hace en modo Normally Open (NO) (Normalmente Abierto) : circuito abierto mientras no se actúe sobre el interruptor. EL interruptor en reposo envía un 0, y pulsado envía un 1.

```
1 void check (int num_param,int pin,int val_on) {
2   etat = digitalRead(pin);          //lit l'etat du pin
3   if (etat == HIGH) {               //si HIGH alors on a appuye donc on change d'etat
4     if (param[num_param] == val_on) {param[num_param] = 0 ;}
5     else {param[num_param] = val_on;}
6   }
7 }
```

Código 1: Función que recoge la información de los pulsadores

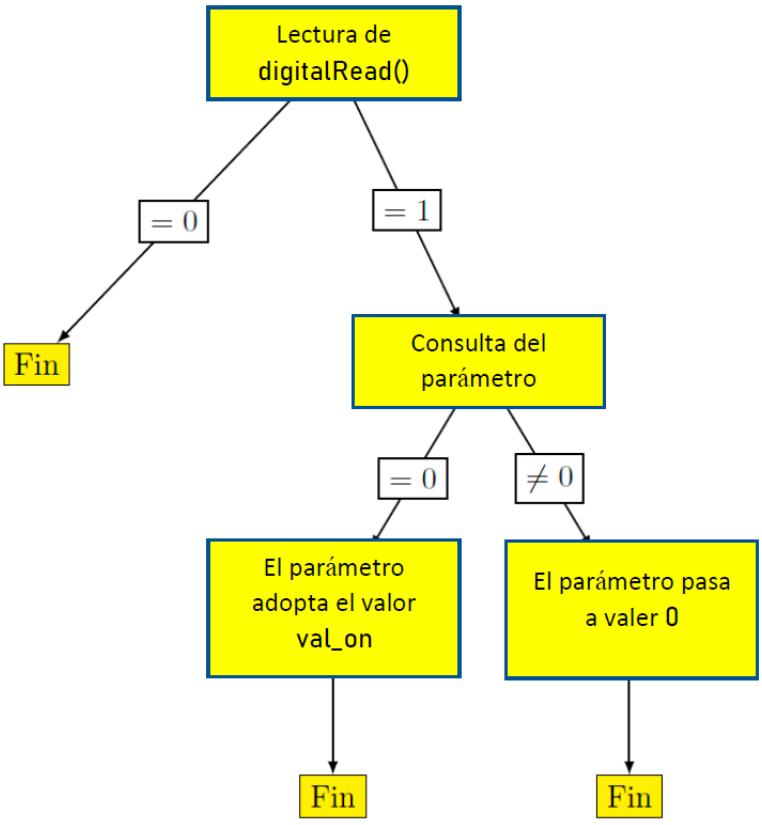


Figura 29: Esquema explicativo de la función `check`

[5]Puede ser 1 , 2 o 3 según el pin observado, ver la descripción del vector `param` en el Anexo E

Por ejemplo, el parámetro 13 controla a la vez los dos intermitentes y las luces de emergencia (warnings) (ver Anexo E). Si `param[13]` vale 1 entonces el intermitente izquierdo está encendido. La activación del interruptor (conmutador) asociado al intermitente derecho, remplazará el valor de `param[13]` por 2. El intermitente izquierdo se apagará, encendiendo el derecho.

La función `read_values()` modifica uno a uno los valores del vector `param`leyendo las entradas asociadas a cada uno de la placa ARDUINO®.

```
1  void read_values() {
2
3    param[1]   = analogRead(pin_speed)          ; // Pedale (Consigne de vitesse)
4    param[2]   = analogRead(pin_break)          ; // Frein (Activation frein)
5    param[3]   = analogRead(pin_direction)      ; // Volant (Consigne de direction)
6
7    if       (digitalRead(pin_drive_state_forward)) {param[8]=1;}
8    else if (digitalRead(pin_drive_state_backward)){param[8]=2;}
9    else {param[8]=0;}
10
11   param[5]   = digitalRead(pin_key)           ; // Clef de contact
12
13   check(9,pin_int_light,1)                    ; // Eclairage interieur
14   check(10,pin_highlight,1)                   ; // Feux de croisement
15   check(11,pin_wipers,1)                      ; // Essuie_glaces
16   check(12,pin_klaxon,1)                      ; // Klaxon
17   check(13,pin_turn_signal_left,1)            ; // Clignotant gauche
18   check(13,pin_turn_signal_right,2)           ; // Clignotant droite
19   check(13,pin_warnings,3)                    ; // Feux de detresse
20
21   param[15] = analogRead(pin_sensor_temperature); // temperature
22   param[18] = analogRead(pin_sensor_light)      ; // intensite lumineuse
23
24   param[16] =                                   ; // lecture charge batterie
25 }
```

Código 2: Función que actualiza los parámetros del vector `param`

Se utiliza la función `analogRead` para leer los valores de las entradas analógicas (velocidad, dirección y nivel de batería). Esa misma función se utiliza para informar de la activación del freno, aunque en realidad es un dato binario (on/off) ya que el vehículo Mobee'City carece de un sensor que cuantifique la frenada mecánica. La función `read_values` asocia a los parámetros el valor de su entrada correspondiente. Este es el caso de la presencia de la llave de contacto, de los sensores exteriores de temperatura y luminosidad. Por otro lado, la función `check` modifica los parámetros asociados a las entradas digitales controladas por interruptores. Por ejemplo, las luces de cruce se encienden al pulsar el interruptor y permanecen encendidas sin necesidad de estar pulsando continuamente.

Actualización de las salidas digitales

Se recuerdan las salidas digitales a continuación=

- Iluminación habitáculo
- Claxon
- Luz placa matrícula
- Limpia parabrisas
- Luces de posición

- Luces de cruce
- Intermitente derecho
- Intermitente izquierdo
- Luz de marcha atrás
- Luz de freno

El estado de estas salidas de la placa ARDUINO® se actualiza con la función `set digital output` (ver Código 5 y el Código completo 1). Si la orden de entrada para esa salida es 1 (HIGH), entonces la salida toma el valor 1, que activará el relé correspondiente haciéndolo conmutar.

```
1  void set_digital_output() {
2  //...
3    // Essuie-glaces
4    if (param[11] == 1) {
5      digitalWrite(wiper, HIGH);
6    }
7    else {
8      digitalWrite(wiper, LOW);
9    }
10 //...
11 }
```

Código 3: Función que actualiza las salidas digitales

Ciertos parámetros presentan algunas particularidades:

- La iluminación de la placa de matriculación se enciende cuando las luces están encendidas. Es una salida que no se controla directamente con ninguna entrada.
- Las luces de cruce no se encienden solas: encender las luces de cruce implica encender las luces de posición.
- Para encender los intermitentes se utiliza la función `millis()`[6] en lugar de la función `delay()`[7] que se utiliza a menudo para itroducir retardos en una salida. De hecho, en este programa, no conviene parar el programa durante unos milisegundos. Utilizando la función `millis()` se puede resvolver ese problema, con la condición de que las salidas sean actualizadas con suficiente frecuencia (para obtener una descripción más detallada, ver el anexo F).

La conducción del vehículo: el diferencial electrónico

Como se ha visto en el apartado 2.2.2, el diferencial electrónico se rige por las ecuaciones 5 y 6 que se recuerdan a continuación.

$$\omega_{m_d} = \frac{V}{k_r.R_r}\left(1 - \frac{1}{2}\frac{d}{L}\tan(\delta)\right) \qquad ((5))$$

$$\omega_{m_g} = \frac{V}{k_r.R_r}\left(1 + \frac{1}{2}\frac{d}{L}\tan(\delta)\right) \qquad ((6))$$

[6] la función `millis()` devuelve el valor del reloj de la placa ARDUINO®, el número de milisegundos desde que ARDUINO® se ha reseteado. Vuelve a resetearse en aproximadamente 50 días [9]

[7] la función `delay()` hace una pausa en el programa durante el tiempo (en milisegundos) especificado como parámetro

Para realizar el cálculo se produce una interrupción en el programa (la función se ejecuta en intervalos de tiempo regulares).

```
//...
double omega_mot_d;
double omega_mot_g;
//...
void calculDifferentiel_gauche(double V, double delta) {
  double dv = 1.8; // Definition des constantes
  double Lv = 2;
  double k_r = 1/7.2;
  double R_r = 0.26;
  omega_mot_d = V*( 1 - dv*tan(delta)/(2*Lv) ) / (k_r*R_r);// Calcul du differentiel
  return omega_mot_d; // Renvoi de la vitesse moteur droit
}
void calculDifferentiel_droit(double V, double delta) {
  double dv = 1.8; // Definition des constantes
  double Lv = 2;
  double k_r = 1/7.2;
  double R_r = 0.26;
  omega_mot_g = V*( 1 + dv*tan(delta)/(2*Lv) ) / (k_r*R_r);
  return omega_mot_g; // Renvoi de la vitesse moteur gauche
}
```

Código 4: Código del diferencial electrónico

Visualización de la información de la conducción LCD

Gracias a la biblioteca `U8glib.h`, se puede porgramar la aparición de un objeto/motivo en cualquier lugar de la pantalla.

```
#include "U8glib.h"
U8GLIB_ST7920_128X64 u8g(13, 11, 12, U8G_PIN_NONE);
```

Código 5: Code qui permet de déclarer la présence de l'écran LCD

Con estas líneas de código se incluye la biblioteca `u8glib.h`, y a continuación se definen los tres pins a los que la pantalla está conectada.

A continuación, se define una matriz que contiene la imagen que queremos mostrar (en este caso, una flecha en el Código 6). Para convertir una imagen en una matriz como la del ejemplo, se ha utilizado el software TheDotFactory©.

```
const unsigned char indicator_right [] U8G_PROGMEM = {
  0x00, 0x20, 0x00, //           #
  0x00, 0x30, 0x00, //           ##
  0x00, 0x38, 0x00, //           ###
  0x00, 0x3C, 0x00, //           ####
  0x00, 0x3E, 0x00, //           #####
  0xFF, 0xFF, 0x00, // ################
  0xFF, 0xFF, 0x80, // #################
  0xFF, 0xFF, 0xC0, // ##################
  0xFF, 0xFF, 0xE0, // ###################
  0xFF, 0xFF, 0xF0, // ####################
  0xFF, 0xFF, 0xF8, // #####################
  0xFF, 0xFF, 0xF8, // #####################
  0xFF, 0xFF, 0xF0, // ####################
  0xFF, 0xFF, 0xE0, // ###################
  0xFF, 0xFF, 0xC0, // ##################
  0xFF, 0xFF, 0x80, // #################
  0xFF, 0xFF, 0x00, // ################
  0x00, 0x3E, 0x00, //           #####
  0x00, 0x3C, 0x00, //           ####
  0x00, 0x38, 0x00, //           ###
```

```
22   0x00, 0x30, 0x00, //          ##
23   0x00, 0x20, 0x00, //          #
24 };
```

Código 6: Declaración de una imagen para el LCD

Por último, para mostrar la imagen en la posición deseada, se utiliza la siguiente línea de código:

```
1 u8g.drawBitmapP( 80, 1, 3, 22, indicator_right);
```

Código 7: Visualización de una imagen en el LCD

Los dos primeros argumentos corresponden a la posición de la esquina superior izquierda en la pantalla (mapa de bits). Los otros dos informan de las dimensiones de la matriz[8]. El resultado de este ejemplo (intermitente derecho, se muestra eb la Figura 31.

Figura 30: Visualización del intermitente derecho en la pantalla

Menú del servicio de mantenimiento

Para utilizar el menú de Servicio de mantenimiento el técnico conecta un dispositivo con cinco interruptores. A continuación, el menú muestra una lista desplegable de todos los parámetros así como su valor. EL técnico puede recorrer la lista pulsando los interruptores `arriba` y `abajo`. Manteniendo el interruptor pulsado, la lista se recorre más rápido.

EL técnico puede consultar la información referente al parámetro elegido gracias al interruptor `OK`. Se puede incluso modificar el valor de los parámetros con los interruptores `derecha` e `izquierda`.

Los interruptores cambian el estado de las entradas digitales, enviando un 0 o un 1.

Si hay imperfecciones en la señal de entrada, estas pueden implicar errores de interpretación por parte del programa. Por ejemplo, si se toma la siguiente sucesión de valores : `00000000111111011111100000000`. ¿El programa debe entender que ha habido un solo apoyo o dos? La programación de las órdenes de los interruptores que utiliza el técnico debe ser muy precisa, para evitar errores de malinterpretación de las señales. La programación de este menú se ha basado en 31.

A continuación se muestra un esquema de cómo funciona la adquisición de las órdenes dadas por el técnico:

[8]advertencia: las dimensiones no vienen dadas en número de pixeles. Se trata de las dimensiones de la matriz codificada en 8 bits

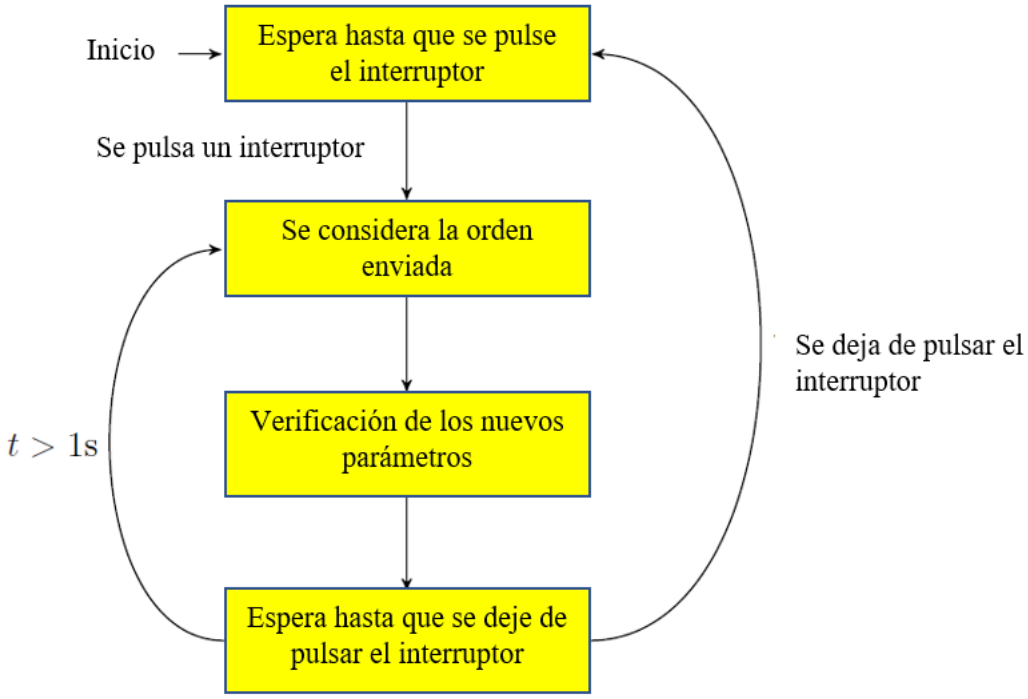


Figura 31: Esquema explicativo del código del Menú de Mantenimiento

Para profundizar en la técnica y el desarrollo en lenguaje ARDUINO®, ver el código completo en 3.

5. Pruebas del programa

Una vez desarrollado el código en lenguaje Arduino, el objetivo del proyecto es realizar ensayos que demuestren su correcto funcionamiento. La indisponibilidad de acceso al banco de ensayo y a los motores reales del vehículo Mobee'city han dificultado esta tarea. Ante la necesidad de demostrar la validez del programa desarrollado, se han llevado a cabo dos ensayos: la visualización de la información en la pantalla LCD y por otro lado la construcción de un pequeño coche que permite verificar el funcionamiento de algunas de las salidas de la placa Arduino. Para simular las entradas de este pequeño prototipo se utiliza una aplicación Android, que envía señales al microcontrolador por Bluetooth.

5.1. Visualización de la información en la pantalla LCD

A continuación se recuerda brevemente la información que debe aparecer en todo momento en la pantalla (ver 4.1.2):

- La velocidad del vehículo en km/h
- EL nivel de carga de la batería. Deben existir al menos 5 niveles diferentes
- El estado de las luces de cruce y de posición
- El estado de los intermitentes
- La temperatura exterior

El vector `param` ha sido introducido en el test con algunos valores determinados. (Recordatorio : En el anexo E contiene el significado de cada parámetro)

```
1 param[20] = {33, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 3, 0, 22, 5, 0, 0, 0}
```

Código 8: Vector de entrada para hacer el test de la pantalla LCD

Se ha hecho una foto al resultado obtenido (Figura 32).

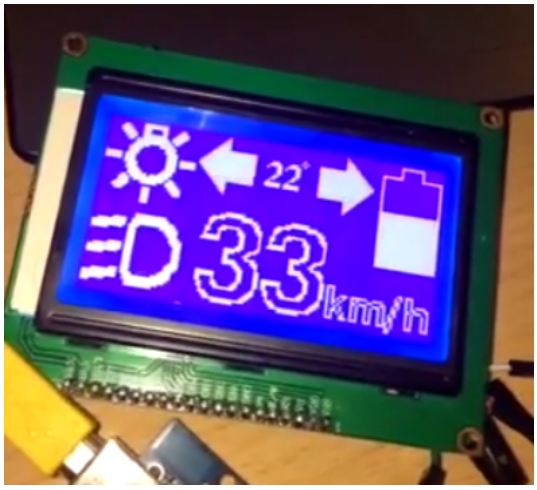


Figura 32: Resultado del test de visualización en la pantalla LCD

Se puede observar que se visualizan dos flechas. En realidad, esas dos flechas parpadean al ritmo de los intermitentes reales. El nivel de batería se ha dividido en 7 niveles distintos.

Por otro lado, se realizó un test dinámico, es decir, las variables cambiando cada segundo actualizando el LCD. Los resultados fueron satisfactorios, los cambios se realizan de manera fluida. La información se muestra en el LCD de manera ordenada, sin retrasos.

También se programaron dos pantallas de Inicio. A continuación se muestran dichas pantallas y el vector `param` correspondiente a cada una de ellas.

```
1 param[20] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0}
```

Código 9: `param` para mostrar la Pantalla de Inicio 1

Figura 33: Accueil 1

```
1 param[20] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0}
```

Código 10: `param` para mostrar la Pantalla de Inicio 2

Figura 34: Accueil 2

La visualización con el LCD cumple los requisitos establecidos en el pliego de condiciones.

5.2. Prototipo: modelo a escala reducida controlado por Bluetooth

EL proyecto exigía la validación, al menos parcial, del pliego de condiciones establecido al principio. Con esa finalidad, se optó por construir un sencillo coche teledirigido. La construcción de este coche (que será llamado prototipo a partir de ahora) permite comprobar el funcionamiento de algunas de las salidas electrónicas de la placa ARDUINO®. Esto permite mostrar los resultados obtenidos de una manera muy visual (esencial para continuar con la financiación del proyecto los próximos años), utilizando herramientas de prototipado rápido y además un sistema de comunicación por Bluetooth que conecte el prototipo con un teléfono móvil.

Este prototipo ha permitido comprobar el funcionamiento de la programación de:

- los diferentes estados de funcionamiento *marcha* y *parado*
- las luces de posición
- las luces de cruce
- los intermitentes
- luces de marcha atrás
- el claxon
- la influencia del sensor de luz en las salidas
- diferencial electrónico adaptado al prototipo (pequeños motores cc, dimensiones diferentes etc)
- nivel de batería (batería de litio de 9 V)

Con el prototipo se ha podido demostrar que todos estos elementos funcionan correctamente. Esta manera de probar los resultados presenta limitaciones en cuanto a que no demuestra el funcionamiento de la electrónica programada en el modelo real Mobee'City. No se ha comprobado el funcionamiento de:

- iluminación del habitáculo
- limpia-parabrisas
- luces de freno
- diferencial electrónico real: control de los motores reales de dirección y de propulsión (funcionamiento de los variadores de frecuencia, bujía, etc)
- utilización de la batería real del vehículo y control de su nivel de carga

En ningún momento durante el proyecto se tuvo acceso al vehículo Mobee'City, al banco de ensayo o a sus componentes reales (motores, batería etc.).

Sin embargo, estos ensayos han supuesto una de las partes más importantes del proyecto y un gran reto para el desarrollador. Por un lado, se ha podido comprobar la utilidad que

puede suponer recurrir a herramientas de prototipado rápido para desarrollar y mostrar los resultados de un proyecto. Por otro, se ha conseguido una comunicación satisfactoria a través de Bluetooth entre la placa Arduino que llevará integrada el coche y una aplicación Android, intuitiva pero con un diseño cuidado y con atención al detalle. A continuación se detallará un poco el desarrollo de este prototipo, y en especial el funcionamiento de la aplicación Android y el entorno de programación utilizado para su creación.

5.2.1. La construcción del prototipo

El prototipado rápido (RP) permite la fabricación rápida de modelos físicos utilizando datos de diseño asistido por ordenador (CAD) en tres dimensiones. El prototipado rápido, que se utiliza en un amplio abanico de sectores, permite a las empresas transformar ideas innovadoras en productos finales de éxito de forma rápida y eficiente 32.

El prototipo del proyecto no pretendía ser una representación fiel del modelo real de coche, sino un coche sencillo en el que poder implementar la electrónica de manera rápida y sencilla.

La construcción del coche (su carrocería) se realizó con placas de madera ligera de contrachapado fino. Se utilizó el software Adobe Illustrator para diseñar las piezas que formaban el coche. En su diseño se tuvo en cuenta los agujeros que debía tener cada placa para la posterior instalación de ciertos elementos electrónicos: puerto usb, luces, intermitentes, interruptor apagado/encendido... Estas piezas posteriormente se cortaban utilizando una cortadora láser, puesta a disposición por el Fablab de la École Centrale de Lyon. una vez cortadas se pintaron para darles un acabado más atractivo y posteriormente se ensamblaron con cola de madera, quedando perfectamente fijadas.

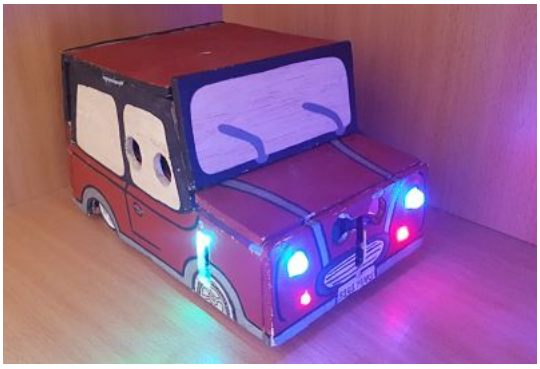


Figura 35: El prototipo construido

El coche tiene dos ruedas directrices, controladas por dos motores 6Vcc y una rueda directriz controlada por un servomotor (Micro Servo 9g SG90 de Tower Pro). Los motores de corriente continua incluyen engranajes internos reductores 48:1 y su velocidad sin carga a 6 V es de 240rpm. Para controlar estos motores se ha conectado a Arduino un Arduino motor shield V1.0 de DK electronics.

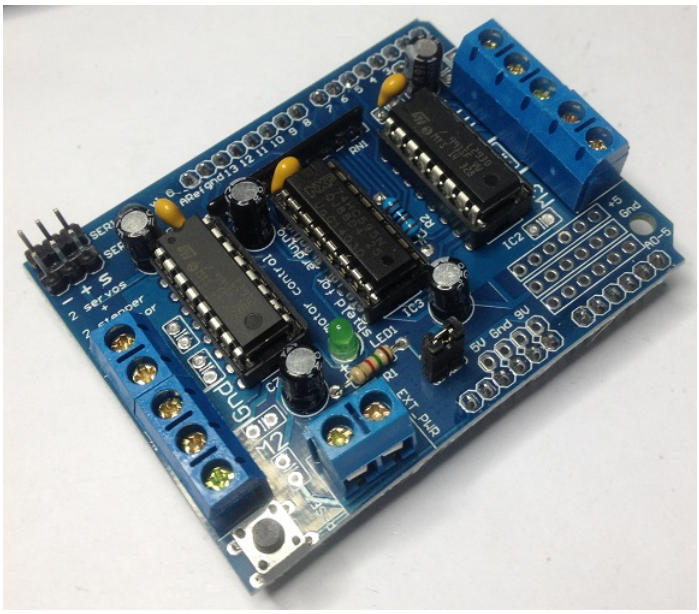


Figura 36: Motor Shield v1.0 para Arduino

Este shield de Arduino tiene dos L293D y permite controlar 4 cargas inductivas + 2 servos conectados al timer dedicado de alta resolución de Arduino. Para el caso de motores, en cada canal se puede controlar tanto la velocidad como la dirección de giro independientemente [33]. Esta propiedad resulta muy útil para este proyecto, ya que nos permite realizar de manera sencilla las pruebas del diferencial electrónico.

Se diseñó en CATIA una pieza que conecta la parte giratoria del servo con la rueda directriz. La pieza sostiene un eje fijo, en el que rota una rueda de forma libre. Esta pieza ha sido impresa gracias a una impresora 3D, y se muestra a continuación:

Figura 37: Soporte Servo impreso en 3D

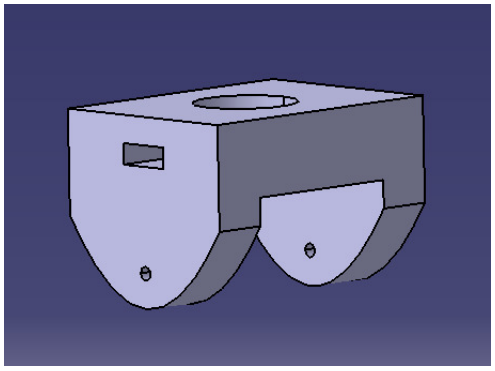


Figura 38: Soporte Servo en Catia

Para establecer las conexiones eléctricas entre los elementos dentro del coche se ha tenido especial cuidado. El coche está sumido a vibraciones o eventuales choques, y esto puede suponer la pérdida de conexión de un cable. Por esta razón, se ha decidido soldar los cables que conectan los LEDs y a las resistencias que les acompañan. Adicionalmente se han aislado correctamente con PVC (asilante eléctrico), evitando que se produzcan contactos no deseados entre conductores.

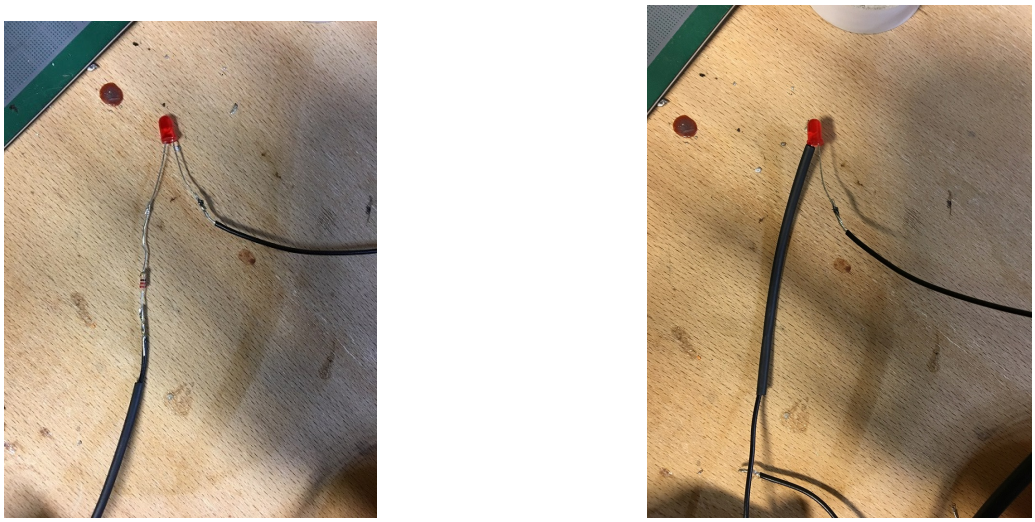


Figura 39: Conexión de LEDs y aislamiento

El elemento clave que se ha conectado a la placa es el módulo Bluetooth HC-05, que permitirá la comunicación con la aplicación Android.

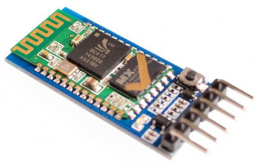


Figura 40: Módulo Bluetooth HC-05

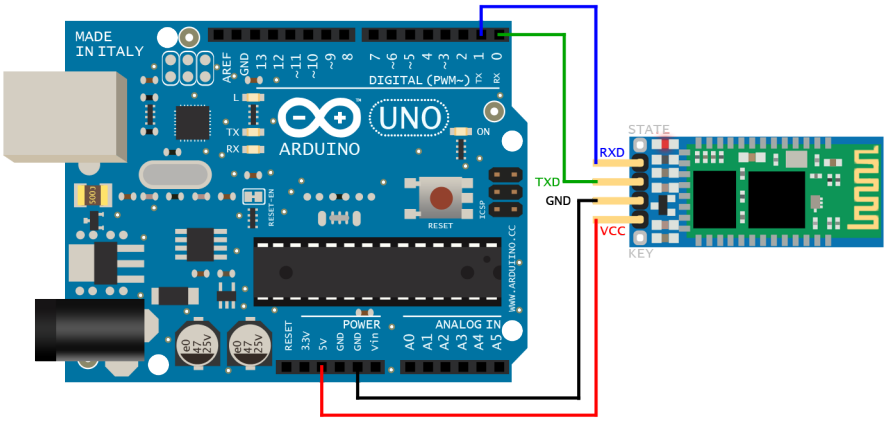


Figura 41: Support Servo Catia

5.2.2. Librerías Arduino y conexión Bluetooth

Para poder controlar los motores conectados al Motor Shield de Arduino, hace falta incluir dos librerías en el código: `AFMotor.h` y `Servo.h`. Las siguientes líneas de código definen los motores y los pines donde están conectados:

```
#include <AFMotor.h>
#include <Servo.h>

AF_DCMotor Motor1(1);
AF_DCMotor Motor2(2);
Servo servo2
```

Código 11: Librerías y definición de los motores

Para el control de la velocidad y el sentido de giro Arduino incluye las siguientes funciones:

- `Motor1.setSpeed(num)`; Siendo num un número de 0 a 255 que indicará el nivel de voltaje de salida aplicado al motor: 0 mínimo, 255 máximo.
- `Motor1.run(FORWARD)`, hace que al motor que hemos definido como Motor1 gire hacia adelante.
- `Motor1.run(BACKWARD)`, hace que al motor que hemos definido como Motor1 gire hacia atrás.

El servo se puede conectar a los pines 9 y 10 del shield, en este caso se ha conectado al 9, `servo2.attach(9)` y se controla su ángulo de giro directamente con la función `servo2.write(Angle)`. El servo demanda una intensidad que no se puede conseguir si la fuente de alimentación del Arduino es el propio ordenador, se necesita una batería (6-10mA en reposo, en rotación 220mA ±20 %, llegando hasta 650mA si hay resistencia).

Por otro lado, es necesario establecer la comunicación Bluetooth. Se trata de una tecnología inalámbrica destinada a conectar dispositivos que se encuentran a corta distancia, que trabaja trabaja mediante transmisiones de radio de onda corta en la banda de los 2,4 GHz creando redes de área personal y consume poca energía [34].

El módulo de bluetooth HC-05 es el que ofrece una mejor relación de precio y características, ya que es un módulo Maestro-Esclavo, quiere decir que además de recibir conexiones desde una PC o móvil, también es capaz de generar conexiones hacia otros dispositivos bluetooth.

Antes de realizar la conexión del módulo bluetooth, debe configurarse y para ello se hace uso de los comandos AT, con ellos se pueden configurar algunas funcions como por ejemplo, cambiar el nombre del módulo, el código PIN para realizar la conexión o el Baud Rate (9600 por defecto).

Una vez configurado y conectado a la placa Arduino (en este caso se ha conectado a los pines RX y TX, GND y alimentación 5V), ya puede ser utilizado para el proyecto. EL módulo funciona únicamente en modo esclavo (recibe información enviada por la aplicación Android).

EL módulo Bluetooth se ha llamado `serial1`. Se inicializa en la función `setup`:

```
1    Serial1.begin(9600);                                          //
         Inicialisation comunicacion avec portable
```

Código 12: Inicialización serial Bluetooth

Las siguientes líneas de código muestran la adquisición de datos por Bluetooth, que a continuación se insertan en un vector.

```
1  void lire(){                                                    // LECTURES
       PARAMETRES BLUETOOTH
2  for (int i=0;i<7;i++){
3    Auxparametres[i]=parametres[i];
4  }
```

```
if (Serial1.available() > 0){string = "";}
while(Serial1.available() > 0)
{command=((byte)Serial1.read());
if(command==':'){
break;}
else{string+=command;}
delay(2);}
Auxstring=string[0];
parametres[0]=Auxstring.toInt();                  // marche avant/arriere
Auxstring=string[1];
parametres[1]=(Auxstring.toInt()*100);           // Acceleration centaines
Auxstring=string[2];
parametres[1]=parametres[1]+(Auxstring.toInt()*10);      // Acceleration décent
Auxstring=string[3];
parametres[1]=parametres[1]+(Auxstring.toInt());        // Acceleration unitées
Auxstring=string[4];
parametres[2]=(Auxstring.toInt())*100;                  // Dirreccion centaines
Auxstring=string[5];
parametres[2]=parametres[2]+(Auxstring.toInt())*10;     // Dirreccion décent
Auxstring=string[6];
parametres[2]=parametres[2]+(Auxstring.toInt());        // Dirreccion unitées
Auxstring=string[7];
parametres[3]=Auxstring.toInt();                        // Clignotant
Auxstring=string[8];
parametres[4]=Auxstring.toInt();                        // Feux
Auxstring=string[9];
parametres[5]=Auxstring.toInt();                        // Essuie-Glasse
Auxstring=string[10];
parametres[6]=Auxstring.toInt();                        // Klasson
Auxstring=string[11];
parametres[7]=(Auxstring.toInt()*100);                  // Verification centaines
Auxstring=string[12];
parametres[7]=parametres[7]+(Auxstring.toInt()*10);     // Acceleration décent
Auxstring=string[13];
parametres[7]=parametres[7]+(Auxstring.toInt());        // Acceleration unitées
sumo=0;
for(int i=0;i<7;i++){sumo+=parametres[i];}
if (sumo!=parametres[7]){
  for (int i=0;i<7;i++){
  parametres[i]=Auxparametres[i];
  }
}
}
```

Código 13: Adquisición de datos por Bluetooth

Arduino recibe una cadena de caracteres (el significado de cada caracter viene especificado en el apartado 5.3 destinado a la aplicación Android). El código recibe esta cadena `command`, y la almacena en `string`. Es recomendable añadir un pequeño `delay` de 2ms, para evitar que el programa lea las órdenes demasiado rápido y se produzcan errores.

Los caracteres se van guardando en el vector `paramentres[]` en forma de números enteros (función `toInt`. El `parámetro[7]` es la suma de todos los parámetros anteriores. Los últimos 3 caracteres que recibe Android por Bluetooth es un numero entero que es la suma de todos los valores enviados. Estos dos valores se comparan para verificar que el mensaje recibido es correcto (ver apartado 5.3).

5.3. Aplicación móvil

La idea de desarrollar una *Aplicación Mobee'CIty* es uno de los requisitos para crear una plataforma de Carsharing (como pretende ser Mobee'City).

Sin embargo, el desarrollo de esa aplicación (control GPS de los puestos de carga,

reserva del coche ect.) pasó a ser un objetivo secundario de este proyecto, cuya prioridad es desarrollar la base de la electrónica con Arduino y realizar las pruebas necesarias para comprobar su funcionamiento.

Ante esta situación, Monsieur Perrard propuso el desarrollo de una aplicación móvil que controlara a distancia un pequeño prototipo que llevara integrada la electrónica desarrollada con Arduino a lo largo del curso.

La aplicación permite al usuario controlar el vehículo a distancia. La comunicación Bluetooth es de tipo unidireccional maestro/esclavo. El dispositivo móvil envía información al microcontrolador del coche. Se debe poder controlar con la aplicación: la velocidad, la dirección y los accesorios. Sin embargo, no se ha programado la recepción de información envíada del coche al usuario.

5.3.1. Android Studio

La aplicación se ha desarrollado Android Studio, que es el IDE oficial de Google para desarrollar aplicaciones Android, y esto ha sido el principal motivo de su elección frente a otras herramientas como Eclipse.

Existen multitud de sitios web para adquirir conocimientos acerca de Android Studio y Java (videos, tutoriales, webs...)

Android Studio está basado en IntelliJ IDEA de la compañía JetBrains, está programado en Java y es multiplataforma [35].

Los proyectos Android están divididos en diferentes carpetas, que internamente se denominan paquetes (packages), y es donde se encuentran todos los recursos del proyecto.

Hay distintos paquetes: unos con los ficheros `Activity` de código java, otros con los archivos `Layout` en Extensible Markup Language (XML) de visualización y configuración, y también otros recursos como las imágenes e iconos de la aplicación.

El proyecto tiene cuatro ficheros java. El `MainActivity` es la actividad principal desde la que se accede a los demás elementos de la aplicación (interfaz de Inicio). Además, `interfacePilotage` `interfaceHelp` e `interfaceConnection` se encargan de la actividad de las interfaces de la aplicación.

5.3.2. La aplicación Mobee'City

Las Figuras 42, 43 y 44 muestran las tres pantallas/interfaces principales de la aplicación Mobee'City:

Figura 42: Pantalla de Inicio

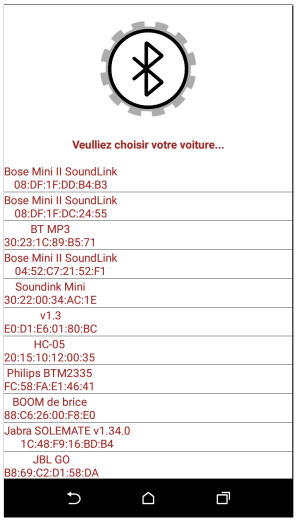


Figura 43: Pantalla de conexión Bluetooth

La pantalla de Inicio, aparece al abrir la aplicación. Su diseño es sencillo y atractivo, para animar al usuario a continuar utilizando la aplicación.

Al pulsar sobre el icono `Appairer un véhicule`, aparece la segunda pantalla (Figura 43), que permite activar el módulo Bluetooth del dispositivo y presenta una lista desplegable con los dispositivos disponibles a los que puede conectarse.

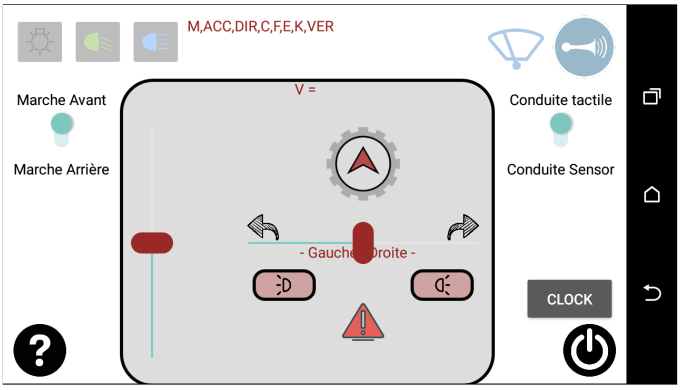


Figura 44: Interfaz de conducción

La tercera pantalla (figura 44) es la que contiene un mayor número de elementos, ya que es la interfaz de conducción. A parir de ella se definen los parámetros que se envían por Bluetooth al coche. Un cuadro gris en el centro de la pantalla contiene dos barras deslizantes (`SeekBar`), que permiten controlar la velocidad y la dirección del vehículo.

La pantalla muestra a la derecha un conmutador (Conduite sensor/ Conduite tactile). Haciéndolo el usuario puede elegir dos modos de controlar la conducción:

- un modo manual en el que el usuario controla tanto la aceleración como la dirección con las barras táctiles.
- un modo *sensor*, con el que la dirección se puede controlar inclinando el teléfono.

Además, se han desarrollado dos pantallas secundarias. A la primera (Figura 45) se accede desde la pantalla de Inicio y presenta los créditos, información sobre Mobee'City. La segunda es una interfaz de ayuda a la conducción (Figura 46). Se accede a ella desde la Interfaz de conducción y se divide en tres etapas que muestran progresivamente al usuario la utilidad de cada botón.

Figura 45: Créditos de la App

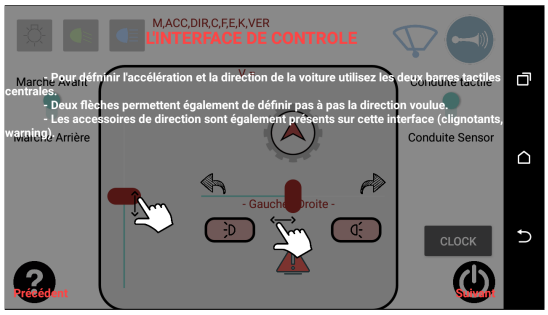


Figura 46: Interfaz de ayuda

Se puede observar que la aplicación Mobee'City se caracteriza por su simplicidad y ergonomía, conteniendo únicamente los elementos necesarios para facilitar al usuario la conducción del vehículo a distancia.

El principio de funcionamiento general se basa en la comunicación Bluetooth entre el smartphone y la placa ARDUINO®. Esa comunicación la establece el usuario al utilizar la aplicación, y esto permite enviar toda la información necesaria a la placa electrónica.

Una vez establecida la conexión, se envía un mensaje de menara recurrente en intervalos de tiempo regulares. Ese mensaje es tratado por la placa Arduino (como se indicó en el apartado 4.1), para enviar las consignas a los distintos elementos del coche.

Los dos aspectos técnicos más relevantes que conviene aclarar sobre el funcionamiento de la aplicación son: por un lado la comunicación Bluetooth y por otro lado la conducción en modo *sensor* que utiliza sensores del propio teléfono. Para ello es necesario declarar en el `Manifest` las funciones de hardware o software usada por la aplicación. El objetivo de una declaración de <uses-feature>es informar a una entidad externa sobre el conjunto de funciones de hardware y software que la aplicación requiere. Necesitaremos añadir dos permisos para poder usar el Bluetooth y descubrir dispositivos que se encuentren a nuestro alcance. En concreto, los permisos son *BLUETOOTH* y *BLUETOOTHADMIN* [36].

```
1    <uses-permission android:name="android.permission.BLUETOOTH" />
2    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
3    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
4
```

```
5     <uses-feature
6         android:name="android.hardware.sensor.accelerometer"
7         android:required="true" />
```

Código 14: Declaración en el manifiesto de la utilización de Bluetooth y el acelerómetro

Para empezar, como ya se ha comentado, la conexión Bluetooth se establece con la placa ARDUINO® gracias al módulo Bluetooth configurado y conectado a ella. El smartphone es el dispositivo maestro, es decir, envía información al otro dispositivo. Esta información se envía en forma de cadena de caracteres de números enteros que corresponde a `M,ACC,DIR,C,F,E,K,VER` donde :

- `M` sentido de la marcha,
- `ACC` aceleración (velocidad),
- `DIR` valor de la dirección,
- `C` información referente a los intermitentes,
- `F` las luces,
- `E` información del limpia-parabrisas,
- `K` claxon,
- `VER` número entero de verificación que es la suma de todos los valores precedentes. Este número entero permite a la placa ARDUINO® verificar que el mensaje recibido es correcto.[9].

Estos caracteres toman valores según la acción del usuario sobre los distintos elementos gráficos de la interfaz de conducción. Por ejemplo, el valor de `ACC` depende directamente de la posición del cursor de la barra de aceleración. Su posición se transcribe como un numero entero entre 0 y 100 en la variable `ACC`.

El mensaje se construye a partir de cada elemento gráfico que tiene una incidencia en el estado de las variables. Cada 10 milisegundos, se construye el mensaje en forma de cadena de caracteres (`string`) y se envía a la placa electrónica. La placa recibe por tanto una "palabra"de la forma "`MACCDIRCFEKVER `", que posteriormente es descriptada y verificada gracias al parámetro `VER`. Si tras la verificación el mensaje es correcto, se actualizan las salidas de la placa ARDUINO®

Por otro lado, al haber introducido el modo de conducción *sensor*, la aplicación utiliza sensores que forman parte del smartphone. Este modo de conducción permite controlar la dirección el coche inclinando el móvil. Para ello, la aplicación utiliza un acelerómetro presente en el smartophone, un sensor que incluyen todos los fabricantes de smartphone.

El algoritmo se divide en las siguientes etapas:

1. Inicialización del acelerómetro para acceder a sus medidas,
2. Tratamiento de los datos para determinar la inclinación del dispositivo,
3. generar un valor para la consigna de dirección

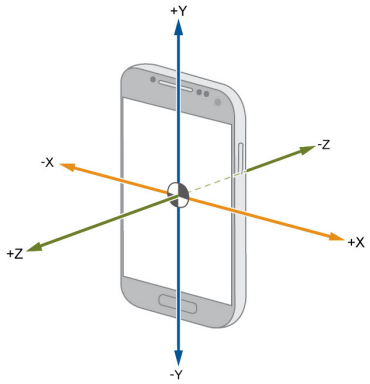


Figura 47: Ejes de referencia asociados al smartphone

Para la primera etapa, se tarta simplemente de declarar en el código que el acelerómetro del smartphone va a ser utilizado y que la información que aporta va a ser utilizada regularmente. El eje x define el movimiento lateral, mientras que el eje y define el movimiento vertical. El eje z define el movimiento dentro y fuera del plano definido por los ejes x e y. El sensor permite enviar tres valores, los componentes de la aceleración del dispositivo (aceleración de la gravedad) (X esPositivo cuando el dispositivo se inclina hacia la derecha (es decir, su lado izquierdo se levanta), Y es positivo, cuando la parte inferior del dispositivo (la del micrófono) se eleva, y Z : Positivo cuando la pantalla esté mirando hacia arriba).

Conociendo los componentes de la fuerza G aplicados a nuestro acelerómetro (G_X, G_Y, G_Z) podemos conocer el ángulo de inclinación (cambios en la orientación del dispositivo). A partir de expresiones trigonométricas se puede obtener el ángulo de balanceo [37].

$$\phi = \arctan\left(\frac{G_Y}{\sqrt{G_X^2 + G_Z^2}}\right) \tag{19}$$

Por último el ángulo de balanceo se declara proporcional a la dirección del vehículo.

Para la mayoría de pruebas, se ha utilizado el simulador AVD (Android Virtual Device) que ya está integrado en el IDE. El emulador utilizado es el API Android 9, 28 (la elección no es muy importante, ya que la App se cargará en distintos dispositivos Android). El emulador permite conocer los parámetros captados por los sensores del vehículo, lo que ha ha resultado útil para hacer pruebas de los valores detectados por el acelerómetro.

[9]Se trata de hacer una "suma de control". La programación de este sistema se basa en la fuente [38]

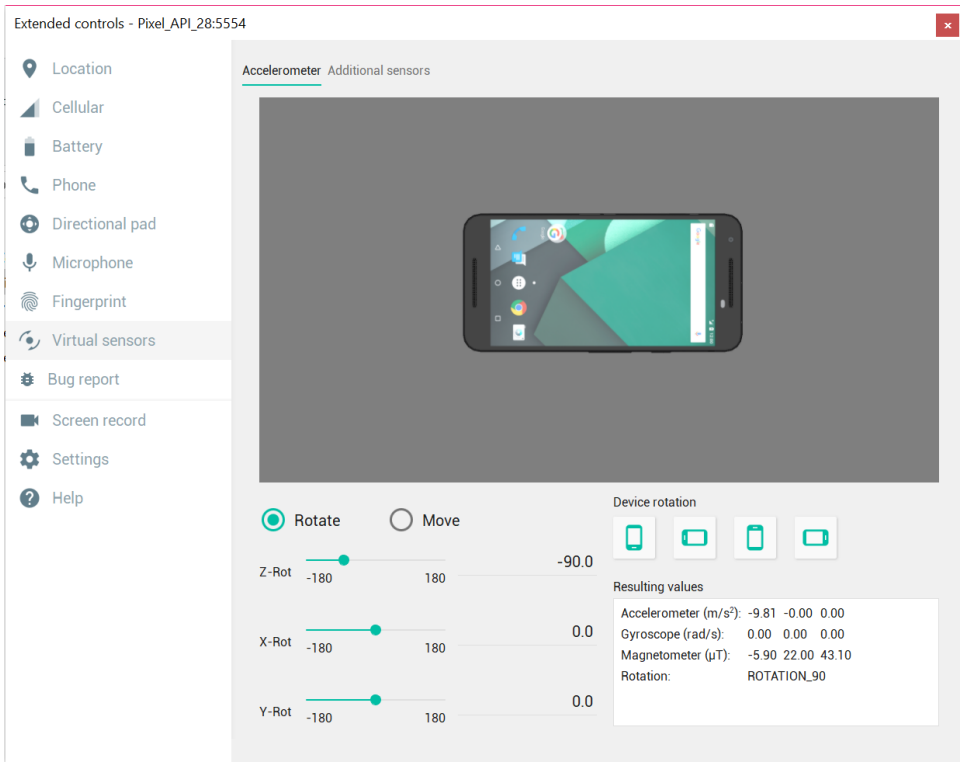


Figura 48: Controles adicionales del emulador de Android Studio

Para realizar los test de funcionamiento de la aplicación y de la comunicación Bluetooth con el prototipo, se utilizó un Sony Xperia.

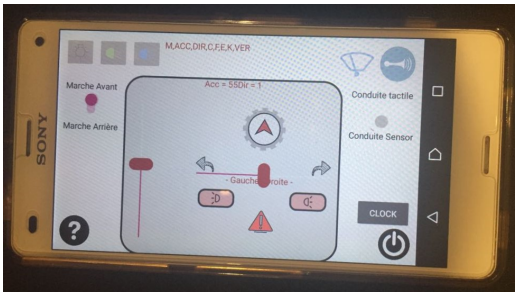


Figura 49: App Mobee'City probada en un smartphone Sony

6. Conclusiones y mejoras a futuro

El proyecto cumple el objetivo principal: desarrollo con Arduino de la electrónica de a bordo del vehículo, que sirva de base para el desarrollo del proyecto y facilite su continuidad y mejora.

La simulación del funcionamiento del diferencial electrónico se ha realizado a partir de un modelo simplificado, suficiente para una programación de base. Sin embargo, el modelo no tiene en cuenta efectos aerodinámicos, ángulos de deslizamiento de los neumáticos, irregularidades en la carretera o resistencia sobre las ruedas. En un futuro, se debe completar esta modelización para programar un sistema que controle la velocidad de las ruedas incluso cuando existan factores externos que modifiquen el comportamiento del vehículo.

La continuación del proyecto requiere profundizar en el estudio del control de los variadores y los motores reales del vehículo (de la rueda motriz y la rueda directriz). Esto será posible una vez se haya instalado el "banco de ensayo", con los elementos reales del vehículo y con un panel de control (formado por interruptores y joysticks) que ha sido diseñado por Monsieur Perrard a lo largo del curso 2017-2018 y cuya programación ya ha sido comenzada (ver Anexo G). La adaptación del código y las pruebas realizadas permitirán la posterior implementación en el vehículo Mobee'City.

La inmediata continuación del proyecto (curso 2018-2019) incluirá la utilización de un Shield BusCAN (en concreto CAN-BUS Shield MCP2551, Datasheet 39). Se utilizarán dos placas Arduino, la primera situada en la parte delantera del vehículo y encargada del puesto de mando y actuadores frontales, y la segunda se situará detrás y se encargará del cálculo del diferencial y actuadores traseros del coche. Con esta elección se consigue simplificar el cableado y disminuir el tiempo de cálculo de los microcontroladores (uno sólo tendría que tratar un número muy elevado entradas y salidas).

Por último, con este proyecto se han aplicado conocimientos técnicos y capacidades adquiridas en la formación de grado en ingeniería electromecánica: conocimientos de electrónica, modelización de sistemas, fabricación, capacidad de profundizar en un nuevo entorno de programación (Aplicaciones Android) y utilización de herramientas de organización, de gestión y presentación de proyectos.

7. Bibliografía

Referencias

[1] Anne-Charlotte Laugier. *Consommation collaborative : l'autopartage c'est tendance.* Charlotte au volant [Internet]. 19 mayo 2017. Disponible en: `https://www.charlotteauvolant.net/autopartage/`. Última consulta: Julio 2018

[2] Weblogs Branded Content Team. Motorpasion [Internet]. *La transformación del carsharing no es el futuro del coche, es el presente.* 13 junio 2017. Disponible en: `https://www.motorpasion.com/espaciotoyota/la-transformacion-del-carsharing-no-es-el-futuro-del-coche-es-el-presente`. Última consulta: Julio 2018

[3] Raúl Marín Gimeno. *TFG: Análisis estratégico del negocio de coches compartidos y su implantación en Valencia* 2014.

[4] Asociación AEDIVE [Internet]. Disponible en: `http://aedive.es/carsharing/`. Última consulta: Julio 2018 4

[5] Carlos Noya. Foro Coches Eléctrico [Internet] *¿Cuál es el modelo perfecto para los sistemas de car sharing?*. 2015. Disponible en: `https://forococheselectricos.com/2015/06/cual-es-el-modelo-perfecto-para-los-sistemas-de-car-sharing.html`. Última consulta: Julio 2018

[6] Mathieu Passenaud. synergeek.fr [Internet]. *Les systèmes embarqués dans l'automobile.* 15 enero 2010. Disponible en: `https://www.synergeek.fr/les-systemes-embarques-dans-automobile/`. Última consulta: Julio 2018

[7] Gonzalo Lara. Motorpasión [Internet]. *CAN Bus: la forma de transmitir información en el automóvil.* 24 enero 2013. Disponible en: `https://www.motorpasion.com/coches-hibridos-alternativos/can-bus-como-gestionar-toda-la-electronica-del-automovil`. Última consulta: Julio 2018

[8] Nick Diakopoulos and Stephen Cass. IEEE [Internet]. *Interactive: The Top Programming Languages 2016.* 26 Julio 2016. Disponible en: `https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2016`. Última consulta: Julio 2018

[9] Aprendiendo Arduino [Internet] *Qué es Arduino.* 25 Septiembre 2016. Disponible en: `https://aprendiendoarduino.wordpress.com/2016/09/25/que-es-arduino/`. Última consulta: Julio 2018

[10] Nextia Fénix [Internet]. *Arduino vs PIC: La gran batalla.* 17 Julio 2014. Disponible en: `https://www.nextiafenix.com/arduino-vs-pic-la-gran-batalla/`. Última consulta: Julio 2018

[11] Alfonso Gómez Matesanz. Trabajo Fin de Grado. *Aplicación Android para la empresa Travelling-Service.* Septiembre 2014.

[12] Juan Francisco Sánchez López. Trabajo Fin de Grado. *Aplicación para Android para el control de un vehículo a través de un trazo*. 2013.

[13] Wikipedia, la Enciclopedia libre [Internet] Disponible en: `https://es.wikipedia.org`Última consulta: Julio 2018

[14] Bartolomé Vargas Díaz. Trabajo Fin de Grado. *Desarrollo de aplicaciones móviles Android MyMoney*. Enero 2016.

[15] [Internet] Disponible en: `http://skiras.blogspot.com/2008/05/3ms-iii-configuraciones-mecnicas.html`. Última consulta: Julio 2018

[16] Geometría de Ackerman [Internet] Disponible en: `http://lacuevitaf1.blogspot.com/2008/12/geometra-en-la-direccin-de-ackerman.html`. Última consulta: Julio 2018

[17] Équipe Mobee'City 2015 *Rapport final de soutenance 2014/2015*

[18] Kada Hartani, Mohamed Bourahla, Yahia Miloud, Mohamed Sekour. *Electronic Differential with Direct Torque Fuzzy Control for Vehicle Propulsion System*. 2009

[19] Escuela Politécnica Superior de Ingeniería Gijón [Internet] *Modelado de un motor CC* Disponible en: `http://isa.uniovi.es/idiaz/ADSTel/Practicas/ModeladoMotorCC.html`. Última consulta: Julio 2018

[20] Bruno Elio Vargas Tamani *Ingeniería de control. Modelo y control de un motor D.C.*

[21] Germán Salmerón Palacios *Estudio y modelización de un diferencial electrónico* Junio 2016

[22] Página oficial de arduino. [Internet] Disponible en: `https://www.arduino.cc/`. Última consulta: Julio 2018

[23] Jorge Campo Ávila *Diseño y desarrollo de un sistema vehicle to home para vehículos eléctricos*. 2017

[24] DATASHEET ACD 4805 Motor Controller

[25] DATASHEET Optocoupler, Phototransistor Output (Dual, Quad Channel) ILD615, ILQ615 Vishay Semiconductors

[26] *Introducción a los relés* [Internet] Disponible en: `https://www.inventable.eu/introduccion-a-los-reles/`. Última consulta: Julio 2018

[27] *Alimentar el Arduino* [Internet] Disponible en: `https://www.geekfactory.mx/tutoriales/tutoriales-arduino/alimentar-el-arduino-la-guia-definitiva/`. Última consulta: Julio 2018]

[28] Luis Llamas *Ingeniería, Informática y Diseño.* [Internet] Disponible en: `https://www.luisllamas.es/entradas-analogicas-en-arduino/`. Última consulta: Julio 2018

[29] Iosu García de Baquedano Mauleón y Jesús López Taberna*Control escalar de un convertidor trifásico con una placa Arduino*. 2014.

[30] Borja Abad Carpintero *Desarrollo e implantación de una placa de test gestionada por Arduino*.

[31] *Un simple bouton*, Disponible en: `http://eskimon.fr/96-arduino-204-un-simple-bouton`. Última consulta: Julio 2018

[32] *Rapid Prototyping* [Internet] Disponible en: `http://www.stratasys.com/es/resources/rapid-prototyping`. Última consulta: Julio 2018.

[33] *Shield de motores para Arduino* [Internet] `http://www.electronicoscaldas.com/shields-escudos-arduino/511-shield-de-motores-para-arduino-shd-mstepper.html`. Última consulta: Julio 2018

[34] *¿Qué es Bluetooth y para qué sirve?* Disponible en: `https://www.tuexperto.com/2013/05/06/que-es-bluetooth-y-para-que-sirve/`. Última consulta: Julio 2018[]

[35] *Android Studio v1.0: características y comparativa con Eclipse* [Internet]. Disponible en: `https://academiaandroid.com/android-studio-v1-caracteristicas-comparativa-eclipse/`. Última consulta: Julio 2018

[36] Developer Android. Disponible en: `https://developer.android.com/guide/topics/manifest/uses-feature-element?hl=es-419`. Última consulta: Julio 2018

[37] *Tilt sensing using a three-axis accelerometer.* `http://www.nxp.com/assets/documents/data/en/application-notes/AN3461.pdf`. Última consulta: Julio 2018

[38] A checksum algorithm.Disponible en: `http://www.flounder.com/checksum.htm`. Última consulta: Julio 2018

[39] Datasheet High-Speed CAN Transceiver, `https://www.sparkfun.com/datasheets/DevTools/Arduino/MCP2551.pdf`

Parte II:

Código fuente

1. Función set digital output. Actualización de las salidas digitales.

```
void set_digital_output() {
  // Éclairage intérieur
  if (param[9] == 1) {
    digitalWrite(interior_light, HIGH);
  }
  else {
    digitalWrite(interior_light, LOW);
  }

  // Klaxon
  if (param[12] == 1) {
    digitalWrite(horn, HIGH);
  }
  else {
    digitalWrite(horn, LOW);
  }

  // Essuie-glace
  if (param[11] == 1) {
    digitalWrite(wiper, HIGH);
  }
  else {
    digitalWrite(wiper, LOW);
  }

  // Eclairage plaque
  if (param[10] == 1 || param[10] == 2) {
    digitalWrite(license_plate_light, HIGH);
  }
  else {
    digitalWrite(license_plate_light, LOW);
  }

  // Feux
  if (param[10] == 2) {
    digitalWrite(sidelights, HIGH);
    digitalWrite(headlights, HIGH);
  }
  else if (param[10] == 1) {
    digitalWrite(sidelights, HIGH);
    digitalWrite(headlights, LOW);
  }
  else {
    digitalWrite(sidelights, LOW);
    digitalWrite(headlights, LOW);
  }

  // Clignotants
  if (param[13] == 3) {
    if (millis() % 666 > 333) {
      digitalWrite(indicator_left, HIGH);
      digitalWrite(indicator_right, HIGH);
    }
    else {
      digitalWrite(indicator_left, LOW);
      digitalWrite(indicator_right, LOW);
    }
  }
  else if (param[13] == 2) {
    digitalWrite(indicator_left, LOW);
    if (millis() % 666 > 333) {
      digitalWrite(indicator_right, HIGH);
    }
    else {
      digitalWrite(indicator_right, LOW);
    }
```

```
67    }
68    else if (param[13] == 1) {
69      digitalWrite(indicator_right, LOW);
70      if (millis() % 666 > 333) {
71        digitalWrite(indicator_left, HIGH);
72      }
73      else {
74        digitalWrite(indicator_left, LOW);
75      }
76    }
77    else {
78      digitalWrite(indicator_left, LOW);
79      digitalWrite(indicator_right, LOW);
80    }
81
82    // Feux de recul
83    if (param[8] == 2) {
84      digitalWrite(reverse_light, HIGH);
85    }
86    else {
87      digitalWrite(reverse_light, LOW);
88    }
89
90    // Feux de frein
91    if (param[2] == 1) {
92      digitalWrite(brake_lights, HIGH);
93    }
94    else {
95      digitalWrite(brake_lights, LOW);
96    }
97
98  }
```

Código 15: Función que actualiza las salidas numéricas

2. Code de la commande du LCD

```
#include "U8glib.h"
#include "bibli.h"

U8GLIB_ST7920_128X64 u8g(13, 11, 12, U8G_PIN_NONE);


float param[20] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

void draw() {

  int kmph = param[0];
  int lights = param[10];
  int indicators = param[13];
  int temperature = param[15];
  int battery_level = param[16];
  int welcome = param[17];


  if      (welcome == 1) {
    u8g.drawBitmapP( 7, 15, 15, 34, mobeecity);
  }
  else if (welcome == 2) {
    u8g.drawBitmapP( 0, 10, 16, 44, centralelyon);
  }
  else {
    // Speed
    // positioning

    int unit = kmph % 10;
    int tens = (kmph - (kmph % 10)) / 10;

    int xtens = 40;
    int ytens = 29;
    int xunit = xtens + 24;
    int yunit = ytens;

    // tens digit
    if (tens == 1) {
      u8g.drawBitmapP( xtens, ytens, 3, 33, one);
    }
    else if (tens == 2) {
      u8g.drawBitmapP( xtens, ytens, 3, 33, two);
    }
    else if (tens == 3) {
      u8g.drawBitmapP( xtens, ytens, 3, 33, three);
    }
    else if (tens == 4) {
      u8g.drawBitmapP( xtens, ytens, 3, 33, four);
    }
    else if (tens == 5) {
      u8g.drawBitmapP( xtens, ytens, 3, 33, five);
    }
    else if (tens == 6) {
      u8g.drawBitmapP( xtens, ytens, 3, 33, six);
    }
    else if (tens == 7) {
      u8g.drawBitmapP( xtens, ytens, 3, 33, seven);
    }
    else if (tens == 8) {
      u8g.drawBitmapP( xtens, ytens, 3, 33, height);
    }
    else if (tens == 9) {
      u8g.drawBitmapP( xtens, ytens, 3, 33, nein);
    }

    // unit digit
    if (unit == 0) {
      u8g.drawBitmapP( xunit, yunit, 3, 33, zero);
```

```
69      }
70      else if (unit == 1) {
71        u8g.drawBitmapP( xunit, yunit, 3, 33, one);
72      }
73      else if (unit == 2) {
74        u8g.drawBitmapP( xunit, yunit, 3, 33, two);
75      }
76      else if (unit == 3) {
77        u8g.drawBitmapP( xunit, yunit, 3, 33, three);
78      }
79      else if (unit == 4) {
80        u8g.drawBitmapP( xunit, yunit, 3, 33, four);
81      }
82      else if (unit == 5) {
83        u8g.drawBitmapP( xunit, yunit, 3, 33, five);
84      }
85      else if (unit == 6) {
86        u8g.drawBitmapP( xunit, yunit, 3, 33, six);
87      }
88      else if (unit == 7) {
89        u8g.drawBitmapP( xunit, yunit, 3, 33, seven);
90      }
91      else if (unit == 8) {
92        u8g.drawBitmapP( xunit, yunit, 3, 33, height);
93      }
94      else if (unit == 9) {
95        u8g.drawBitmapP( xunit, yunit, 3, 33, nein);
96      }
97
98      // km/h
99      u8g.drawBitmapP( xunit + 24, ytens + 20, 5, 14, kmparh);
100
101     // Sidelights
102
103     if (lights == 1 || lights == 2) {
104       u8g.drawBitmapP( 1, 1, 5, 30, sidelights);
105     };
106
107     // Headlights
108
109     if (lights == 2) {
110       u8g.drawBitmapP( 1, 35, 5, 26, headlights);
111     };
112
113     // Indicators
114
115     int t = millis() % 666;
116     if (t > 333) {
117       if (indicators == 1 || indicators == 3) {
118         u8g.drawBitmapP( 35, 1, 3, 22, indicator_left );
119       };
120       if (indicators == 2 || indicators == 3) {
121         u8g.drawBitmapP( 80, 1, 3, 22, indicator_right);
122       };
123     }
124
125     // Battery
126
127     if (battery_level == 0) {
128       u8g.drawBitmapP( 104, 1, 3, 39, battery_0);
129     };
130     if (battery_level == 1) {
131       u8g.drawBitmapP( 104, 1, 3, 39, battery_1);
132     };
133     if (battery_level == 2) {
134       u8g.drawBitmapP( 104, 1, 3, 39, battery_2);
135     };
136     if (battery_level == 3) {
137       u8g.drawBitmapP( 104, 1, 3, 39, battery_3);
138     };
139     if (battery_level == 4) {
140       u8g.drawBitmapP( 104, 1, 3, 39, battery_4);
```

```
141     };
142     if (battery_level == 5) {
143       u8g.drawBitmapP( 104, 1, 3, 39, battery_5);
144     };
145     if (battery_level == 6) {
146       u8g.drawBitmapP( 104, 1, 3, 39, battery_6);
147     };
148     if (battery_level == 7) {
149       u8g.drawBitmapP( 104, 1, 3, 39, battery_7);
150     };
151
152     // Temperature
153
154     u8g.drawCircle(75, 6, 1);
155     if (temperature < 10 && temperature > 0) {
156       u8g.setFont(u8g_font_timB10);
157       u8g.drawStr(67, 17, String(temperature).c_str());
158     }
159     else {
160       u8g.setFont(u8g_font_timB10);
161       u8g.drawStr(58, 17, String(temperature).c_str());
162     };
163
164   }
165 }
166
167 void setup(void) {
168
169   param[15] = 21; // Temperature
170   param[16] = 5; // Battery level
171 }
172
173 void loop(void) {
174
175
176   while (true) {
177     u8g.firstPage();
178     do {
179       draw();
180     } while ( u8g.nextPage() );
181     delay(50);
182   }
183 }
```

Código 16: Función que actualiza la visualización de la pantalla LCD

3. Código del *Menú de servicio de mantenimiento*

```
#include "U8glib.h"

// Branchement :
// 2 -> gauche
// 3 -> droite
// 4 -> haut
// 5 -> entrer
// 6 -> bas
// 11 -> SID
// 12 -> CS
// 13 -> SCK

float param[20] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
float param_min[20] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -99, 0, 0, 0, 0};
float param_max[20] = {99, 255, 1, 255, 1, 1, 1, 1, 2, 0, 2, 1, 1, 3, 0, 99, 7, 2, 0,
     0};

// Pins des boutons
int bouton_gauche = 2;
int bouton_droite = 3;
int bouton_haut = 4;
int bouton_bas = 6;
int bouton_entrer = 5;

// Selection correcpond a la variable selectionnee
int selection = 0;
int ok = 0; // Profondeur dans le menu

// Suite est une variable qui vaut 1 quand le bouton reste enfonce
int suite;

// Parametre de l'affichage
int pos_droite = 110;
int pos_gauche = 7;
int pos_haut = 8;
int interligne = 10;

// Longueur de param
int nb_param = sizeof(param) / sizeof(float);


// Pins du LCD
U8GLIB_ST7920_128X64 u8g(13, 11, 12, U8G_PIN_NONE);


void verif() { // Toutes les contraintes sur les differents parametres

  if (selection <= 0) {
    selection = 0;
  }
  if (selection >= nb_param - 1) {
    selection = nb_param - 1;
  }

  for (int i = 0; i <= 19; i++) {

    if (param[i] < param_min[i]) {
      param[i] = param_min[i];
    }
    if (param[i] > param_max[i]) {
      param[i] = param_max[i];
    }
  }

}

void menu(int dir)
// 0 : Rien
```

```
68  // 1 : Gauche
69  // 2 : Droite
70  // 3 : Haut
71  // 4 : Bas
72  // 5 : Entrer
73  {
74    // Action a faire lors de l'appuie sur un bouton
75    if (dir == 1) {
76      param[selection] = param[selection] - 1;
77    };
78    if (dir == 2) {
79      param[selection] = param[selection] + 1;
80    };
81    if (dir == 3) {
82      selection = selection - 1;
83    };
84    if (dir == 4) {
85      selection = selection + 1;
86    };
87    if (dir == 5 && ok == 1 && suite == 0) {
88      ok = 0;
89    }
90    else if (dir == 5 && ok == 0 && suite == 0) {
91      ok = 1;
92    };
93
94
95    verif(); // On verifie que les valeurs sont possibles
96
97    // Affichage
98    if (ok == 0) {
99      u8g.setFont(u8g_font_6x12);
100
101     if (selection <= 3) { // Pour les trois premieres valeurs la fleche est en haut
102       for (int i = 0; i <= 6; i++) {
103
104         String debut = "param[";
105         String numero = String(i);
106         String affichage = debut + numero + "]";
107
108         u8g.drawStr(pos_gauche, pos_haut + i * interligne, affichage.c_str());
109         u8g.drawStr(pos_droite, pos_haut + i * interligne, String(int(param[i])).
              c_str());
110       }
111
112       u8g.drawStr(0, 10 * selection + 10, ">");
113     }
114     else if (selection <= nb_param - 4) { // La fleche ne bouge pas
115
116       for (int i = 0; i <= 6; i++) {
117
118         String debut = "param[";
119         String numero = String(selection + i - 3);
120         String affichage = debut + numero + "]";
121
122         u8g.drawStr(pos_gauche, pos_haut + (i)*interligne, affichage.c_str());
123         u8g.drawStr(pos_droite, pos_haut + (i)*interligne, String(int(param[selection
               + i - 3])).c_str());
124       }
125
126       u8g.drawStr(0, 10 * 3 + 10, ">");
127     }
128     else {
129
130       for (int i = nb_param - 6; i <= nb_param - 1; i++) {
131
132         String debut = "param[";
133         String numero = String(i);
134         String affichage = debut + numero + "]";
135
136         u8g.drawStr(pos_gauche, pos_haut + (i - 14)*interligne, affichage.c_str());
```

```
137         u8g.drawStr(pos_droite, pos_haut + (i - 14)*interligne, String(int(param[i]))
                .c_str());
138       }
139
140       u8g.drawStr(0, 10 * (selection - nb_param + 6) + 10, ">");
141     }
142
143   }
144   else {
145     u8g.setFont(u8g_font_6x13);
146     u8g.drawStr(pos_gauche + 100, pos_haut + 5, String(int(param[selection])).c_str()
          );
147     u8g.setFont(u8g_font_5x7);
148     if (selection == 0 ) {
149       u8g.drawStr(pos_gauche, pos_haut, "Vitesse")              ;
150       u8g.drawStr(pos_gauche, pos_haut + 10, "[0;99]"  ) ;
151       u8g.drawStr(pos_gauche, pos_haut + 20, "Vitesse vehicule (km/h)" );
152     };
153     if (selection == 1 ) {
154       u8g.drawStr(pos_gauche, pos_haut, "Consigne de vitesse")  ;
155       u8g.drawStr(pos_gauche, pos_haut + 10, "[0;255]" ) ;
156       u8g.drawStr(pos_gauche, pos_haut + 20, "Consigne vitesse"        );
157     };
158     if (selection == 2 ) {
159       u8g.drawStr(pos_gauche, pos_haut, "Frein")                ;
160       u8g.drawStr(pos_gauche, pos_haut + 10, "[0;1]"   ) ;
161       u8g.drawStr(pos_gauche, pos_haut + 20, "Intensite freinage"      );
162     };
163     if (selection == 3 ) {
164       u8g.drawStr(pos_gauche, pos_haut, "Volant")               ;
165       u8g.drawStr(pos_gauche, pos_haut + 10, "[0;255]" ) ;
166       u8g.drawStr(pos_gauche, pos_haut + 20, "Angle virage"            );
167     };
168     if (selection == 4 ) {
169       u8g.drawStr(pos_gauche, pos_haut, "Frein a main")         ;
170       u8g.drawStr(pos_gauche, pos_haut + 10, "[0;1]"   ) ;
171       u8g.drawStr(pos_gauche, pos_haut + 20, "0 : Frein deverouille"   );
172       u8g.drawStr(pos_gauche, pos_haut + 30, "1 : Frein verrouille"   );
173     };
174     if (selection == 5 ) {
175       u8g.drawStr(pos_gauche, pos_haut, "Clef de contact")      ;
176       u8g.drawStr(pos_gauche, pos_haut + 10, "[0;1]"   ) ;
177       u8g.drawStr(pos_gauche, pos_haut + 20, "0 : Clef pas tournee"    );
178       u8g.drawStr(pos_gauche, pos_haut + 30, "1 : Clef tournee"       );
179     };
180     if (selection == 6 ) {
181       u8g.drawStr(pos_gauche, pos_haut, "Ceinture securite")    ;
182       u8g.drawStr(pos_gauche, pos_haut + 10, "[0;1]"   ) ;
183       u8g.drawStr(pos_gauche, pos_haut + 20, "0 : Ceinture pas mise"   );
184       u8g.drawStr(pos_gauche, pos_haut + 30, "1 : Ceinture mise"      );
185     };
186     if (selection == 7 ) {
187       u8g.drawStr(pos_gauche, pos_haut, "Presence conducteur")  ;
188       u8g.drawStr(pos_gauche, pos_haut + 10, "[0;1]"   ) ;
189       u8g.drawStr(pos_gauche, pos_haut + 20, "0 : Conducteur absent"   );
190       u8g.drawStr(pos_gauche, pos_haut + 30, "1 : Conducteur present" );
191     };
192     if (selection == 8 ) {
193       u8g.drawStr(pos_gauche, pos_haut, "Sens de circulation")  ;
194       u8g.drawStr(pos_gauche, pos_haut + 10, "[0;2]"   ) ;
195       u8g.drawStr(pos_gauche, pos_haut + 20, "0 : Point mort"          );
196       u8g.drawStr(pos_gauche, pos_haut + 30, "1 : Marche avant"       );
197       u8g.drawStr(pos_gauche, pos_haut + 40, "2 : Marche arriere"      );
198     };
199     if (selection == 9 ) {
200       u8g.drawStr(pos_gauche, pos_haut, "Non attribue")         ;
201       u8g.drawStr(pos_gauche, pos_haut + 10, "[0;0]"   ) ;
202       u8g.drawStr(pos_gauche, pos_haut + 20, ""                        );
203     };
204     if (selection == 10) {
205       u8g.drawStr(pos_gauche, pos_haut, "Feux")                 ;
206       u8g.drawStr(pos_gauche, pos_haut + 10, "[0;2]"   ) ;
```

```
207       u8g.drawStr(pos_gauche, pos_haut + 20, "0 : Feux eteints"        );
208       u8g.drawStr(pos_gauche, pos_haut + 30, "1 : Feux de position"    );
209       u8g.drawStr(pos_gauche, pos_haut + 40, "2 : Feux de croisement"  );
210     };
211     if (selection == 11) {
212       u8g.drawStr(pos_gauche, pos_haut, "Essuies glace")        ;
213       u8g.drawStr(pos_gauche, pos_haut + 10, "[0;1]"    ) ;
214       u8g.drawStr(pos_gauche, pos_haut + 20, "0 : Eteints"             );
215       u8g.drawStr(pos_gauche, pos_haut + 30, "1 : Allumes"            );
216     };
217     if (selection == 12) {
218       u8g.drawStr(pos_gauche, pos_haut, "Klaxon")               ;
219       u8g.drawStr(pos_gauche, pos_haut + 10, "[0;1]"    ) ;
220       u8g.drawStr(pos_gauche, pos_haut + 20, "0 : Eteint"              );
221       u8g.drawStr(pos_gauche, pos_haut + 30, "1 : Allume"             );
222     };
223     if (selection == 13) {
224       u8g.drawStr(pos_gauche, pos_haut, "Clignotants")          ;
225       u8g.drawStr(pos_gauche, pos_haut + 10, "[0;3]"    ) ;
226       u8g.drawStr(pos_gauche, pos_haut + 20, "0 : Eteints"             );
227       u8g.drawStr(pos_gauche, pos_haut + 30, "1 : Gauche"             );
228       u8g.drawStr(pos_gauche, pos_haut + 40, "2 : Droite"  );
229       u8g.drawStr(pos_gauche, pos_haut + 50, "3 : Warning"  );
230     };
231     if (selection == 14) {
232       u8g.drawStr(pos_gauche, pos_haut, "Non attribue")         ;
233       u8g.drawStr(pos_gauche, pos_haut + 10, "[0;0]"    ) ;
234       u8g.drawStr(pos_gauche, pos_haut + 20, ""                        );
235     };
236     if (selection == 15) {
237       u8g.drawStr(pos_gauche, pos_haut, "Temperature")          ;
238       u8g.drawStr(pos_gauche, pos_haut + 10, "[-99;99]") ;
239       u8g.drawStr(pos_gauche, pos_haut + 20, "Temperature mesuree (C)" );
240     };
241     if (selection == 16) {
242       u8g.drawStr(pos_gauche, pos_haut, "Niveau  batterie")     ;
243       u8g.drawStr(pos_gauche, pos_haut + 10, "[0;7]"    ) ;
244       u8g.drawStr(pos_gauche, pos_haut + 20, "0 : Batterie vide"       );
245       u8g.drawStr(pos_gauche, pos_haut + 30, "..."                    );
246       u8g.drawStr(pos_gauche, pos_haut + 40, "7 : Batterie pleine"    );
247     };
248     if (selection == 17) {
249       u8g.drawStr(pos_gauche, pos_haut, "Mode d'affichage")     ;
250       u8g.drawStr(pos_gauche, pos_haut + 10, "[0;2]"    ) ;
251       u8g.drawStr(pos_gauche, pos_haut + 20, "0 : Conduite"            );
252       u8g.drawStr(pos_gauche, pos_haut + 30, "1 : Logo Mobee'City"    );
253       u8g.drawStr(pos_gauche, pos_haut + 40, "2 : Logo Centrale Lyon" );
254     };
255     if (selection == 18) {
256       u8g.drawStr(pos_gauche, pos_haut, "Non attribue")         ;
257       u8g.drawStr(pos_gauche, pos_haut + 10, "[0;0]"    ) ;
258       u8g.drawStr(pos_gauche, pos_haut + 20, ""                        );
259     };
260     if (selection == 19) {
261       u8g.drawStr(pos_gauche, pos_haut, "Non attribue")         ;
262       u8g.drawStr(pos_gauche, pos_haut + 10, "[0;0]"    ) ;
263       u8g.drawStr(pos_gauche, pos_haut + 20, ""                        );
264     };
265
266   }
267 }
268
269
270 void setup(void) {
271   pinMode(bouton_gauche, INPUT);
272   pinMode(bouton_droite, INPUT);
273   pinMode(bouton_haut, INPUT);
274   pinMode(bouton_bas, INPUT);
275   pinMode(bouton_entrer, INPUT);
276   Serial.begin(9600);
277 }
278
```

```
279 void loop(void) {
280   int val_gauche = digitalRead(bouton_gauche);
281   int val_droite = digitalRead(bouton_droite);
282   int val_haut = digitalRead(bouton_haut);
283   int val_bas = digitalRead(bouton_bas);
284   int val_entrer = digitalRead(bouton_entrer);
285
286
287   // Attente de l'appui : situation de base
288   while (val_gauche == 1 && val_droite == 1 && val_haut == 1 && val_bas == 1 &&
          val_entrer == 1) {
289
290     // Mise a jour variable
291     val_gauche = digitalRead(bouton_gauche);
292     val_droite = digitalRead(bouton_droite);
293     val_haut = digitalRead(bouton_haut);
294     val_bas = digitalRead(bouton_bas);
295     val_entrer = digitalRead(bouton_entrer);
296
297     u8g.firstPage();// Mise a jour de l'affichage
298     do {
299       menu(0);
300     } while ( u8g.nextPage() );
301
302     delay(10);
303     suite = 0;
304
305   }
306
307   // Si on appuie sur un bouton
308   if (val_gauche == 0) {
309     Serial.println("GAUCHE");
310     menu(1);
311   }
312   if (val_droite == 0) {
313     Serial.println("DROITE");
314     menu(2);
315   }
316   if (val_haut == 0) {
317     Serial.println("HAUT");
318     menu(3);
319   }
320   if (val_bas == 0) {
321     Serial.println("BAS");
322     menu(4);
323   }
324   if (val_entrer == 0) {
325     Serial.println("ENTRER");
326     menu(5);
327   }
328   Serial.print(ok);
329
330   int debut = millis();
331   // Tant qu'il y a un bouton enfonce
332   while ( val_gauche == 0 || val_droite == 0 || val_haut == 0 || val_bas == 0 ||
          val_entrer == 0) {
333     val_gauche = digitalRead(bouton_gauche);
334     val_droite = digitalRead(bouton_droite);
335     val_haut = digitalRead(bouton_haut);
336     val_bas = digitalRead(bouton_bas);
337     val_entrer = digitalRead(bouton_entrer);
338
339     u8g.firstPage();
340     do {
341       menu(0);
342     } while ( u8g.nextPage() );
343     delay(10);
344     int attente = 1000;
345     if (suite == 1) {
346       attente = 50;
347     }
348     if (millis() - debut > attente) {
```

```
349         suite = 1;
350         break;
351       }
352     }
353 }
```

Código 17: Menú de servicio de mantenimiento

4. Código utilizado en el prototipo (control por Bluetooth)

```
#include <AFMotor.h>
#include <Servo.h>

AF_DCMotor Motor1(1);
AF_DCMotor Motor2(2);
Servo servo2;

//--------------------------------------------------------------------
float va=0;
float vb=0;

void MotorSet(int marche, double Acceleration, double Angle){                //
    COMMANDE MOTEUR
   va=(Acceleration+Acceleration*(Angle-90)/90);                             //
       calcule differenciel
   vb=(Acceleration-Acceleration*(Angle-90)/90);
   if (marche==0){Motor1.run(FORWARD);Motor2.run(BACKWARD);}                  //
       Marche arriere
   else {Motor1.run(BACKWARD);Motor2.run(FORWARD);     }                      //
       Marche Avant
   Motor1.setSpeed(abs(va));                                                 //Envoie
        commande moteur
   Motor2.setSpeed(abs(vb));
   servo2.write(Angle);                                                      //Envoie
        commande servomoteur
}
//--------------------------------------------------------------------

int parametres[]={0, 0, 0, 1, 0, 0, 0, 0}; // Vecteur avec tout les parametres:
int Auxparametres[]={0, 0, 0, 1, 0, 0, 0, 0};

/*parametres[0] marche; // 0 avant  .... 1 arriere
parametres[1]  Acceleration; // 0.....100
parametres[2]  direction; // 0.....100 gauche -droite
parametres[3]  clignotant; // 0...1...2...3 gauche rien droite les deux
parametres[4]  feux; // 0..1...2...3
parametres[5]  essuie-glasse; //  0....1
parametres[6]  klaxon; //  0....1
parametre[7]   verification
*/
char command;   // variables pour comunication avec mobile
String string;
String Auxstring;
int sumo;
//--------------------------------------------------------------------
void lire(){                                                          // LECTURES
     PARAMETRES BLUETOOTH
for (int i=0;i<7;i++){
  Auxparametres[i]=parametres[i];
}
if (Serial1.available() > 0){string = "";}
while(Serial1.available() > 0)
{command=((byte)Serial1.read());
if(command==':'){
break;}
else{string+=command;}
delay(2);}
Auxstring=string[0];
parametres[0]=Auxstring.toInt();                                      // marche
    avant/arriere
Auxstring=string[1];
parametres[1]=(Auxstring.toInt()*100);                                //
    Acceleration centaines
Auxstring=string[2];
parametres[1]=parametres[1]+(Auxstring.toInt()*10);                   //
    Acceleration
```

```
57 Auxstring=string[3];
58 parametres[1]=parametres[1]+(Auxstring.toInt());                    //
       Acceleration unitees
59 Auxstring=string[4];
60 parametres[2]=(Auxstring.toInt())*100;                              //
       Dirreccion centaines
61 Auxstring=string[5];
62 parametres[2]=parametres[2]+(Auxstring.toInt())*10;                 //
       Dirreccion
63 Auxstring=string[6];
64 parametres[2]=parametres[2]+(Auxstring.toInt());                    //
       Dirreccion unitees
65 Auxstring=string[7];
66 parametres[3]=Auxstring.toInt();                                     //
       Clignotant
67 Auxstring=string[8];
68 parametres[4]=Auxstring.toInt();                                     // Feux
69 Auxstring=string[9];
70 parametres[5]=Auxstring.toInt();                                     // Essuie-
       Glasse
71 Auxstring=string[10];
72 parametres[6]=Auxstring.toInt();                                     // Klasson
73 Auxstring=string[11];
74 parametres[7]=(Auxstring.toInt()*100);                               //
       Verification centaines
75 Auxstring=string[12];
76 parametres[7]=parametres[7]+(Auxstring.toInt()*10);                  //
       Acceleration decent
77 Auxstring=string[13];
78 parametres[7]=parametres[7]+(Auxstring.toInt());                     //
       Acceleration unitee
79 sumo=0;
80 for(int i=0;i<7;i++){sumo+=parametres[i];}
81 if (sumo!=parametres[7]){
82   for (int i=0;i<7;i++){
83   parametres[i]=Auxparametres[i];
84   }
85 }
86 }
87 //-------------------------------------------------------------------------
88 void Visualisation(){                                                 //
       VISUALISATION PARAMETRES ECRAN ORDINATEUR
89   Serial.print("Marche ");  Serial.print("Acc     ");
90   Serial.print("Dir    ");  Serial.print("Clign    ");
91   Serial.print("Feux   "); Serial.print("EssuiG   ");
92   Serial.println("Klasson ");
93 for (int i=0;i<8;i++){
94   Serial.print(parametres[i]);
95   Serial.print("\t");
96   }
97   Serial.println();
98 }
99 //-------------------------------------------------------------------------
100 double Acceleration=0;
101 double Angle=0;
102 double Vmax=100;                                                     // valeur
       maximal fourni par
103 double Angmax=100;
104 //-------------------------------------------------------------------------
105 void VarMotor(){                                                     // FONCTION
       DE CONTROLE PAR BLUETOOTH
106 Acceleration=parametres[1]/Vmax*255;                                //
       Changement de variable: Vitesse [0,255] valeur max pour moteur
107 Angle=parametres[2]/Angmax*180;                                     //
       Changement de variable: Angle [0,180]
108 }
109
110 //-------------------------------------------------------------------------
111 int pinklason=14;
112 int pinsensordist=A9;
113 int distancef=0;
114 int pinledr=38;
```

```
115 int pinledg=40;
116 int pinledb=36;
117 int son=0;
118 float auxv=0;
119 int t=0;
120 //---------------------------------------------------------------------------
121 void accesoires(){                                                    //
        FONCTION DES ACCESOIRES
122  if (parametres[5]==1){digitalWrite(pinledr,0); digitalWrite(pinledg,1);digitalWrite(
        pinledb,1);}    // essuie-glasse
123  else {digitalWrite(pinledr,0); digitalWrite(pinledg,0);digitalWrite(pinledb,0); }
124
125  if (parametres[6]==1){tone(pinklason,461);}
126  else {noTone(pinklason);}
127
128  if (parametres[0]==1){
129  distancef=analogRead(pinsensordist);
130  distancef=12746.73*pow(distancef, -1.2134);
131  if (distancef<13){t=500;}
132  if (distancef<10){t=300;}
133  if (distancef<7){t=200;}
134  if (distancef<5){t=10;}
135  else {noTone(pinklason);t=0;}
136  if(millis()<(auxv+t)){tone(pinklason,400);}
137     else if(millis()<(auxv+2*t)){noTone(pinklason);}
138   else {auxv=millis();}}
139  }
140
141 //---------------------------------------------------------------------------
142 int pinfeuxg=35;
143 int pinfeuxd=37;
144 int pinclignotantd=39;
145 int pinclignotantg=41;
146 int pinLDR=A8;
147 int lum[1000];
148 int moyennelum=0;
149 int limlum=293;
150 int auxlum=0;
151 long aux=0;
152 int auxparametres=0;
153
154 //---------------------------------------------------------------------------
155 void feux(){                                                    // FONCTION
        DES FEUX
156   // if (analogRead(pinLDR)>1020){parametres[4]=1;}                  //
         allumbrage automatique le soir
157   switch (parametres[4]) {                                      // feux
158   case 0:
159   digitalWrite(pinfeuxd,0);
160   digitalWrite(pinfeuxg,0);
161   break;
162   case 1:
163   digitalWrite(pinfeuxd,1);
164   digitalWrite(pinfeuxg,0);
165   break;
166   case 2:
167   digitalWrite(pinfeuxd,0);
168   digitalWrite(pinfeuxg,1);
169   break;
170   case 3:
171   digitalWrite(pinfeuxd,1);
172   digitalWrite(pinfeuxg,1);
173   break;
174   }
175
176   if(parametres[3]!=0){
177     if(millis()<(aux+500)){auxparametres=parametres[3];}
178     else if(millis()<(aux+1000)){auxparametres=0;}
179   else {aux=millis();}
180   }
181   else auxparametres=0;
```

```
182   switch (auxparametres){                                                    //
          Clignotants
183   case 0:
184   digitalWrite(pinclignotantd,0);
185   digitalWrite(pinclignotantg,0);
186   break;
187   case 1:
188   digitalWrite(pinclignotantd,0);
189   digitalWrite(pinclignotantg,1);
190   break;
191   case 2:
192   digitalWrite(pinclignotantd,1);
193   digitalWrite(pinclignotantg,0);
194   break;
195   case 3:
196   digitalWrite(pinclignotantd,1);
197   digitalWrite(pinclignotantg,1);
198   break;
199   }
200
201 }
202 //---------------------------------------------------------------------------
203 //---------------------------------------------------------------------------
204 void setup(){
205    Serial.begin(9600);                                                       //
          Inicialisation comunicacion avec ordinateur
206    Serial1.begin(9600);                                                      //
          Inicialisation comunicacion avec portable
207    servo2.attach(9);
208    pinMode(pinfeuxd,OUTPUT);
209    pinMode(pinfeuxg,OUTPUT);
210    pinMode(pinclignotantd,OUTPUT);
211    pinMode(pinclignotantg,OUTPUT);
212    pinMode(pinklason,OUTPUT);
213    pinMode(pinledr,OUTPUT);
214    pinMode(pinledg,OUTPUT);
215    pinMode(pinledb,OUTPUT);
216    pinMode(pinLDR,INPUT);
217    pinMode(pinsensordist,INPUT);
218 }
219 //---------------------------------------------------------------------------
220 //---------------------------------------------------------------------------
221 void loop(){
222 lire();
223 //Visualisation();
224 VarMotor();
225 MotorSet(parametres[0],Acceleration,Angle);
226 feux();
227 accesoires();
228 }
229 //---------------------------------------------------------------------------
230 //---------------------------------------------------------------------------
231 \end{lstlisting}
```

Código 18: Control móvil del prototipo

5. Extractos de código de la App Mobee'City

En este apartado se mostrarán los distintos archivos utilizados para la creación de la aplicación Android en el siguiente orden:

- El manifiesto de la aplicación AndroidManifest.xml: configuración básica de la App.
- Java Main.Activity
- Java Interfaz de Conexión
- Java Interfaz de Conducción
- Layout Main.Activity
- Layout Interfaz de Conexión
- Layout Interfaz de Conducción

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example">

    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

    <uses-feature
        android:name="android.hardware.sensor.accelerometer"
        android:required="true" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/mobeecity_logo_app"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".MainActivity"
            android:screenOrientation="portrait"
            android:theme="@android:style/Theme.DeviceDefault">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".interfacePilotage"
            android:configChanges="orientation|keyboardHidden"
            android:screenOrientation="landscape"
            android:theme="@android:style/Theme.DeviceDefault" />
        <activity
            android:name=".interfaceConnection"
            android:theme="@android:style/Theme.NoTitleBar.Fullscreen" />

    </application>

</manifest>
```

```java
package com.example;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.Window;
import android.view.animation.Animation;
import android.view.animation.RotateAnimation;
import android.widget.ImageView;


public class MainActivity extends Activity {

/**********************************************************************************************/
    /* DECLARATION DES VARIABLES */
/**********************************************************************************************/

    private ImageView imageRoue;

/**********************************************************************************************/
    /* ON CREATE */
/**********************************************************************************************/

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        /* Enlever la barre d'état du haut pour mettre le layout en fullscreen */
        this.requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.activity_main);
        /* Initialisation des accesseurs sur les vues*/
        imageRoue = (ImageView) findViewById(R.id.imageView1);
        /* Animation pour faire tourner la roue */
        RotateAnimation rotate = new RotateAnimation(0, 360, Animation.RELATIVE_TO_SELF, 0.5f, Animation.
RELATIVE_TO_SELF, 0.5f);
        rotate.setDuration(8000);                   //8s
        rotate.setRepeatCount(Animation.INFINITE);  //se répète à l'infini
        imageRoue.setAnimation(rotate);             //Application de l'animation sur l'image
    }

    /* Fin de l'application */
    public void quitter(View view){
        System.exit(0);
    }

    /* Démarrage de l'activité de pilotage */
    public void startPilotage(View view){
        Intent intent = new Intent(MainActivity.this, interfacePilotage.class);
        startActivity(intent);
    }

    /* Démarrage de l'activité de connnection à la voiture */
    public void demarrerConnection(View view){
        Intent intent = new Intent(MainActivity.this, interfaceConnection.class);
        startActivity(intent);
    }
}

```

```java
package com.example;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.IntentFilter;
import android.os.Bundle;
import android.view.View;
import android.view.animation.Animation;
import android.view.animation.RotateAnimation;
import android.widget.AdapterView;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;
import android.widget.ArrayAdapter;
import android.content.Intent;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;

import java.io.OutputStream;
import java.util.ArrayList;
import java.util.Set;

public class interfaceConnection extends Activity {


/*********************************************************************************************/
    /* DECLARATION DES VARIABLES */
/*********************************************************************************************/

    Button btnPaired;
    ListView devicelist;
    ImageView imageViewBluetooth1;
    ImageView imageViewBluetooth2;
    TextView textPick;

    private BluetoothAdapter myBluetooth = null;
    private Set<BluetoothDevice> pairedDevices;
    private OutputStream outStream = null;
    public static String EXTRA_ADDRESS = "device_address";

/*********************************************************************************************/
    /* ON CREATE */
/*********************************************************************************************/

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_interface_connection);

        btnPaired = (Button)findViewById(R.id.buttonConnect);
        devicelist = (ListView)findViewById(R.id.listViewConnect);
        imageViewBluetooth1 = (ImageView)findViewById(R.id.imageViewConnect1);
        imageViewBluetooth2 = (ImageView)findViewById(R.id.imageViewConnect2);
        textPick = (TextView)findViewById(R.id.textViewPick);

        myBluetooth = BluetoothAdapter.getDefaultAdapter();
        if(myBluetooth == null)
        {
            //Affichage du message disant que l'appareil n'a pas de bluetooth
            Toast.makeText(getApplicationContext(), "Votre appareil n'a pas de module Bluetooth" +
                    "\nVous ne pouvez pas utilisez cette application...", Toast.LENGTH_LONG).show();
            //On revient à la première activitivé
            finish();
        }
        else
        {
            if (myBluetooth.isEnabled())
            { }
            else
            {
                //Demande à l'utilisateur d'acitiver le bluetooth
                Intent turnBTon = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
                startActivityForResult(turnBTon,1);
            }
        }

    }

    public void startConnection(View view)
    {
        imageViewBluetooth1.setClickable(false);     //On ne peut plus cliquer sur l'image
        imageViewBluetooth2.setClickable(false);     //On ne peut plus cliquer sur l'image
        btnPaired.setClickable(false);               //On ne peut plus cliquer sur le bouton
        btnPaired.setVisibility(View.INVISIBLE);     //On cache le bouton
        textPick.setVisibility(View.VISIBLE);        //On affiche le texte
        /* Application de l'animation de rotation  */
        RotateAnimation rotate = new RotateAnimation(0, 360, Animation.RELATIVE_TO_SELF, 0.5f, Animation.
```

```
90 RELATIVE_TO_SELF, 0.5f);
91         rotate.setDuration(10000);
92         rotate.setRepeatCount(Animation.INFINITE);
93         imageViewBluetooth2.setAnimation(rotate);
94         /* Appel de la fonction qui affichage les appareils appairés */
95         pairedDevicesList();
96     }
97
98     private void pairedDevicesList() {
99         pairedDevices = myBluetooth.getBondedDevices();
100        ArrayList list = new ArrayList();
101
102        if (pairedDevices.size() > 0) {
103            for (BluetoothDevice bt : pairedDevices) {
104                list.add(bt.getName() + "\n" + bt.getAddress()); //Récupération du nom et de l'adress
105            }
106        } else {
107            Toast.makeText(getApplicationContext(), "No Paired Bluetooth Devices Found.", Toast.LENGTH_LONG).show();
108        }
109
110
111        final ArrayAdapter adapter = new ArrayAdapter(this, R.layout.custom_textview, list);
112        devicelist.setAdapter(adapter);
113        devicelist.setOnItemClickListener(myListClickListener); //Méthode appelée quand on clique sur un élément de la
    liste
114
115    }
116
117    private AdapterView.OnItemClickListener myListClickListener = new AdapterView.OnItemClickListener()
118    {
119        public void onItemClick (AdapterView<?> av, View v, int arg2, long arg3)
120        {
121            //Récupération de l'adress MAC, qui est sur les 17 derniers caractères de la vue
122            String info = ((TextView) v).getText().toString();
123            String address = info.substring(info.length() - 17);
124            //Création de l'intent pour passer à l'activité suivante
125            Intent i = new Intent(interfaceConnection.this, interfacePilotage.class);
126            //Changement d'acitivité
127            i.putExtra(EXTRA_ADDRESS, address);//Envoi de l'adress en paramètre à l'autre activité
128            startActivity(i);
129        }
130    };
131
132 }
133
```

```java
package com.example;

import android.app.Activity;
import android.app.ProgressDialog;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.Intent;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.media.MediaPlayer;
import android.os.AsyncTask;
import android.os.Bundle;
import android.os.Handler;
import android.view.MotionEvent;
import android.view.View;
import android.view.Window;
import android.view.WindowManager;
import android.view.animation.AlphaAnimation;
import android.view.animation.Animation;
import android.view.animation.LinearInterpolator;
import android.widget.CompoundButton;
import android.widget.ImageButton;
import android.widget.ImageView;
import android.widget.SeekBar;
import android.widget.Switch;
import android.widget.TextView;
import android.widget.Toast;
import java.io.IOException;
import java.util.Timer;
import java.util.TimerTask;
import java.util.UUID;


public class interfacePilotage extends Activity implements SensorEventListener {

/************************************************************************************************/
    /* DECLARATION DES VARIABLES */
/************************************************************************************************/

    private MediaPlayer mPlayer = null;         //Media Player

    private SensorManager mSensorManager;       //Sensor manager
    Sensor accelerometer;                       //Accéléromètre

    int previous_dir_value;                     //Ancienne valeur de la barre de direction
    int dir_value,acc_value;                    //Valeurs des barres de commande
    boolean signalRight,signalLeft;             //Etat des clignotants
    boolean isWarning;                          //Etat warning
    private TextView textSeekBarValue;          //Texte d'affichage des valeurs des barres de commande
    private TextView textMessage;

    private SeekBar seekBarDir;                 //Barre de commande de direction
    private SeekBar seekBarAcc;                 //Barre de commande d'accélération

    private Switch switchConduite;              //Switch du mode de conduite
    private Switch switchMarche;                //Switch du mode de conduite

    private ImageView imageViewDirectionCompass;//Image de retour visuel de la barre de direction
    private ImageView rightTurnSignalArrow;     //Image du bouton clignotant droit
    private ImageView leftTurnSignalArrow;      //Image du bouton clignotant gauche
    private ImageView imageView_warning;        //Image du bouton warning

    private ImageButton imagebutton1;           //Image du bouton feux de position
    private ImageButton imagebutton2;           //Image du couton feux de croisement
    private ImageButton imagebutton3;           //Image du bouton feux de route
    private ImageView imagebutton4;             //Image du bouton essuis glace
    private ImageView imagebutton5;             //Image du bouton klaxon

    int etat_sortie[] = new int[7];             //Tableau de booleans sur l'état des sorties binaires
    /* int MARCHE(0,1) | int ACC[0,100] | int DIR[-50,50] | int CLIGNOTANTS(0,1,2,3) | int FEUX(0,1,2,3) | int
ESSUISGLACE(0,1) | int KLAXON(0,1) */
    final Animation flashAnimation = new AlphaAnimation(1, 0);      //Animation de clignotement
    final Animation blinkAnimation = new AlphaAnimation(0.4f,1.0f); //Animation de clignotement

    String address = null;                                                          //
    BluetoothAdapter myBluetooth = null;                                            //
    BluetoothSocket btSocket = null;                //  Variables servants aux protocoles bluetooth
    private ProgressDialog progress;                                                //
    private boolean isBtConnected = false;                                          //
    static final UUID myUUID = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB"); //

/************************************************************************************************/
    /* ON CREATE */
/************************************************************************************************/

    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
90          super.onCreate(savedInstanceState);
91          /* Récupération de l'intent : on récupère l'adresse du véhicule connecté */
92          Intent newint = getIntent();
93          address = newint.getStringExtra(interfaceConnection.EXTRA_ADDRESS);
94
95          /* Enlever la barre d'état du haut pour mettre le layout en fullscreen */
96          this.requestWindowFeature(Window.FEATURE_NO_TITLE);
97          this.getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN, WindowManager.LayoutParams.
    FLAG_FULLSCREEN);
98          setContentView(R.layout.activity_interface_pilotage);
99
100         /* Initialisation et accès à l'accéléromètre */
101         mSensorManager = (SensorManager)getSystemService(SENSOR_SERVICE);
102         accelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
103
104
105         /* Initialisation des accesseurs sur les vues*/
106         textSeekBarValue = (TextView)findViewById(R.id.textViewVitesse);
107         textMessage = (TextView)findViewById(R.id.textViewMessage);
108         seekBarDir = (SeekBar)findViewById(R.id.seekBar_dir);
109         seekBarAcc = (SeekBar)findViewById(R.id.seekBar_acc);
110         switchConduite = (Switch)findViewById(R.id.switch_conduite);
111         switchMarche = (Switch)findViewById(R.id.switch_marche);
112         imageViewDirectionCompass = (ImageView) findViewById(R.id.imageView_directionCompass);
113         rightTurnSignalArrow = (ImageView) findViewById(R.id.turn_signal_right_img);
114         leftTurnSignalArrow = (ImageView) findViewById(R.id.turn_signal_left_img);
115         imageView_warning = (ImageView)findViewById(R.id.imageView_warning);
116         imagebutton1 = (ImageButton) findViewById(R.id.Image_button1);
117         imagebutton2 = (ImageButton) findViewById(R.id.Image_button2);
118         imagebutton3 = (ImageButton) findViewById(R.id.Image_button3);
119         imagebutton4 = (ImageView) findViewById(R.id.Image_button4);
120         imagebutton5 = (ImageView) findViewById(R.id.Image_button5);
121
122         /* Initialisation des valeurs */
123         //Tableau de donnees a envoyer par bluetooth
124         etat_sortie[0] = 0;// int MARCHE(-1,1)
125         etat_sortie[1] = 50;//int ACC[0,100]
126         etat_sortie[2] = 0;//int DIR[-50,50]
127         etat_sortie[3] = 0;//int CLIGNOTANTS(-1,0,1,2)
128         etat_sortie[4] = 0;//int FEUX(0,1,2,3)
129         etat_sortie[5] = 0;//int ESSUISGLACE(0,1)
130         etat_sortie[6] = 0;//int KLAXON(0,1)
131
132         acc_value = 50;
133         dir_value = 0;
134
135         signalLeft = false;
136         signalRight = false;
137         isWarning = false;
138
139         /* Mise au flou des boutons */
140         imagebutton1.setAlpha(0.4f);
141         imagebutton2.setAlpha(0.4f);
142         imagebutton3.setAlpha(0.4f);
143         imagebutton4.setAlpha(0.4f);
144         imagebutton5.setAlpha(0.4f);
145
146         /* Animation pour faire clignoter */
147         flashAnimation.setDuration(500);
148         flashAnimation.setInterpolator(new LinearInterpolator());
149         flashAnimation.setRepeatCount(Animation.INFINITE);
150         flashAnimation.setRepeatMode(Animation.REVERSE);
151         /* Animation pour faire clignoter une fois */
152         blinkAnimation.setDuration(500);
153         blinkAnimation.setInterpolator(new LinearInterpolator());
154         blinkAnimation.setRepeatCount(1);
155         blinkAnimation.setRepeatMode(Animation.REVERSE);
156
157         /* Appel de la class de connection de la voiture par bluetooth */
158         if (address!=null) {
159             new ConnectBT().execute();
160         }
161         else{
162             msg("Aucune voiture a connecter");
163         }
164
165         /* TouchListener pour le bouton klaxon */
166         imagebutton5.setOnTouchListener(new View.OnTouchListener() {
167             public boolean onTouch(View view, MotionEvent event) {
168                 if (event.getAction() == android.view.MotionEvent.ACTION_DOWN) {
169                     etat_sortie[6] = 1;
170                     imagebutton5.startAnimation(blinkAnimation);
171                     playSound(R.raw.klaxon_sound);
172                     imagebutton5.setAlpha(1.0f);
173                 } else if (event.getAction() == android.view.MotionEvent.ACTION_UP) {
174                     etat_sortie[6] = 0;
175                     imagebutton5.setAlpha(0.4f);
176                 }
177                 return true;
178             }
```

```
179         });
180
181         /* Changement du mode de conduite Avant/Arrière */
182         switchMarche.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
183             @Override
184             public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
185                 if(isChecked){
186                     etat_sortie[0]=0;
187                 }else{
188                     etat_sortie[0]=1;
189                 }
190             }
191         });
192
193         /* Changement du mode de conduite Manuelle/Sensor */
194         switchConduite.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
195             @Override
196             public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
197                 if(isChecked){
198                     seekBarAcc.setClickable(true);
199                     seekBarDir.setClickable(true);
200                 }else{
201                     seekBarAcc.setClickable(false);
202                     seekBarDir.setClickable(false);
203                 }
204             }
205         });
206
207         /* Gérer les évènements sur les changements de la barre de direction */
208         seekBarDir.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener(){
209             @Override
210             public void onProgressChanged(SeekBar seekBarDir, int progress, boolean fromUser) {
211                 dir_value = progress - 50;                          //Actualisation de la valeur de la barre de
    commande de direction
212                 textSeekBarValue.setText(
213                         "Acc = " + String.valueOf(acc_value) + "Dir = " + String.valueOf(dir_value));    //Affichage de
     cette valeur
214                 imageViewDirectionCompass.setRotation(dir_value);   //Définir la rotation de l'image en retour visuel
215                 /* Evènements à vérifier si l'un des clignotants est activé */
216                 if(signalLeft == true || signalRight == true)
217                 {
218                     //Détection d'un retour centre pour le clignotant gauche
219                     if(signalLeft == true && previous_dir_value<-15 && dir_value >=-15)
220                     {
221                         //Arrêt de l'animation du clignotant
222                         leftTurnSignalArrow.clearAnimation();
223                         leftTurnSignalArrow.setVisibility(View.INVISIBLE);
224                     }
225                     //Détection d'un retour centre pour le clignotant droit
226                     if(signalRight == true && previous_dir_value>15 && dir_value <=15)
227                     {
228                         //Arrêt de l'animation du clignotant
229                         rightTurnSignalArrow.clearAnimation();
230                         rightTurnSignalArrow.setVisibility(View.INVISIBLE);
231                     }
232                 }
233                 //Actualisation de la valeur précédente de la direction
234                 previous_dir_value = dir_value;
235             }
236
237             @Override
238             public void onStartTrackingTouch(SeekBar seekBarDir) {}
239             @Override
240             public void onStopTrackingTouch(SeekBar seekBarDir) {
241                 /* Remise à zéro de la barre de direction lorsque le doigt de l'utilisateur lache la barre */
242                 dir_value = 0;
243                 seekBarDir.setProgress(50);
244                 textSeekBarValue.setText("Acc = " + String.valueOf(acc_value) + "Dir = " + String.valueOf(dir_value));
245                 imageViewDirectionCompass.setRotation(0);
246             }
247         });
248
249         /* Gérer les évènements sur les changements de la barre d'accélération */
250         seekBarAcc.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener(){
251             @Override
252             public void onProgressChanged(SeekBar seekBarAcc, int progress, boolean fromUser) {
253                 //Actualisation de la valeur de la barre de commande d'accélération
254                 acc_value = progress;
255                 //Affichage de cette valeur
256                 textSeekBarValue.setText("Acc = "+String.valueOf(acc_value)+"Dir = "+String.valueOf(dir_value));
257             }
258             @Override
259             public void onStartTrackingTouch(SeekBar seekBarAcc) {}
260             @Override
261             public void onStopTrackingTouch(SeekBar seekBarAcc) {}
262         });
263     }
264
265 /*********************************************************************************************/
266     /* ON RESUME */
```

```
/*****************************************************************************************************/

    protected void onResume() {
        super.onResume();
        /* Remise en route de l'accelerométre */
        mSensorManager.registerListener(this, accelerometer, SensorManager.SENSOR_DELAY_UI);
    }

/*****************************************************************************************************/
    /* ON PAUSE */
/*****************************************************************************************************/

    protected void onPause() {
        super.onPause();
        /* Mise au repos de l'accéléromètre */
        mSensorManager.unregisterListener(this);
        /* Mise au repos du media player */
        if(mPlayer != null) {
            mPlayer.stop();
            mPlayer.release();
        }
    }

/*****************************************************************************************************/
    /* GERER LES CHANGEMENTS DE VALEURS DES CAPTEURS */
/*****************************************************************************************************/

    public void onAccuracyChanged(Sensor sensor, int accuracy) {  }

    public void onSensorChanged(SensorEvent event) {

        if(switchConduite.isChecked() == false){// Correspond au mode Conduite Sensor
            if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER){
                double pi = 3.141592;
                double g = 9.81;
                //Décomposition sur trois axes de l'accélération
                float x = event.values[0];
                float y = event.values[1];
                float z = event.values[2];
                int Dir,Acc;

                //Loi de commande des barres de commande
                Acc = (int)(100*Math.sin(z*pi/(2*g)));
                Dir = (int)(50*Math.sin(y*pi/(2*g))*Math.cos(z*pi/(2*g)));

                if (acc_value >= 0 && acc_value <= 100 ){
                    //Approche dégressive de l'écart
                    acc_value = acc_value +(Acc-acc_value)/4;
                    //Correction si dépassement
                    if(acc_value < 0){acc_value = 0;}
                    if(acc_value > 100){acc_value = 100;}
                }

                if (dir_value >= -50 && dir_value <= 50 ){
                    //Approche dégressive de l'écart
                    dir_value = dir_value +(Dir-dir_value)/4;
                    //Correction si dépassement
                    if(dir_value < -50){dir_value = -50;}
                    if(dir_value > 50){dir_value = 50;}
                }
                //Actualisation des barres de commande
                seekBarAcc.setProgress(acc_value);
                seekBarDir.setProgress(dir_value+50);
            }
        }

    }

/*****************************************************************************************************/
    /* GERER LES EVENEMENTS DE CLIC SUR LES BOUTONS DE L'INTERFACE */
/*****************************************************************************************************/

    /* Bouton clignotant droit */
    public void Click_rightSignal(View view){
        //Si les warning ne sont pas en cours
        if(etat_sortie[3] != 3) {
            //Si le clignotant est déjà allumé alors on l'éteind
            if (etat_sortie[3] == 1) {
                rightTurnSignalArrow.setVisibility(View.INVISIBLE);
                rightTurnSignalArrow.clearAnimation();
                etat_sortie[3] = 0;
            }
            //Sinon on allume le clignotant correspondant
            else {
                //Si le clignotant opposé est déjà allumé alors on l'éteind
                if (etat_sortie[3] == 2) {
                    leftTurnSignalArrow.setVisibility(View.INVISIBLE);
                    leftTurnSignalArrow.clearAnimation();
                }
                rightTurnSignalArrow.setVisibility(View.VISIBLE);
```

```
                rightTurnSignalArrow.startAnimation(flashAnimation);
                etat_sortie[3] = 1;
            }
        }
    }

    /* Bouton clignotant gauche */
    public void Click_leftSignal(View view){
        //Si les warning ne sont pas en cours
        if(etat_sortie[3]!=3) {
            //Si le clignotant est déjà allumé alors on l'éteind
            if (etat_sortie[3] == 2) {
                leftTurnSignalArrow.setVisibility(View.INVISIBLE);
                leftTurnSignalArrow.clearAnimation();
                etat_sortie[3] = 0;
            }
            //Sinon on allume le clignotant correspondant
            else {
                //Si le clignotant opposé est déjà allumé alors on l'éteind
                if (etat_sortie[3] == 1) {
                    rightTurnSignalArrow.setVisibility(View.INVISIBLE);
                    rightTurnSignalArrow.clearAnimation();
                }
                leftTurnSignalArrow.setVisibility(View.VISIBLE);
                leftTurnSignalArrow.startAnimation(flashAnimation);
                etat_sortie[3] = 2;
            }
        }
    }

    /* Bouton warning */
    public void Click_warning(View view){
        //Si les warning ne sont pas en cours on anime
        if(etat_sortie[3]!=3) {
            etat_sortie[3] = 3;
            imageView_warning.startAnimation(flashAnimation);       //Clignotement du warning
            rightTurnSignalArrow.setVisibility(View.VISIBLE);       //Affichage des deux icones clignotants
            leftTurnSignalArrow.setVisibility(View.VISIBLE);
            rightTurnSignalArrow.startAnimation(flashAnimation);    //Clignotement des icones
            leftTurnSignalArrow.startAnimation(flashAnimation);
        }
        //Sinon on efface les animations
        else{
            etat_sortie[3] = 0;
            imageView_warning.clearAnimation();                     //Arret clignotement warning
            rightTurnSignalArrow.setVisibility(View.INVISIBLE);     //Masquage des deux icones clignotants
            leftTurnSignalArrow.setVisibility(View.INVISIBLE);
            rightTurnSignalArrow.clearAnimation();                  //Arret clignotement des icones
            leftTurnSignalArrow.clearAnimation();
        }
    }

    /* Bouton flèche droite de direction */
    public void Click_rightArrow(View view){
        if (dir_value < 46 ) {                      //Si possible incrémentation de la barre de direction
            dir_value = dir_value + 5;              //Incrémentation
            seekBarDir.setProgress(dir_value + 50); //Actualisation de la barre de direction
        }
    }

    /* Bouton flèche gauche de direction */
    public void Click_leftArrow(View view){
        if (dir_value > -46 ){                      //Si possible incrémentation de la barre de direction
            dir_value = dir_value -5;               //Décrémentation
            seekBarDir.setProgress(dir_value+50);   //Actualisation de la barre de direction
        }
    }

    public void Click_feux_position(View view){
        if (etat_sortie[4] != 1){
            //Actualisation des sorties
            etat_sortie[4] = 1;
            //Actualisation des flous
            imagebutton1.setAlpha(1.0f);
            imagebutton2.setAlpha(0.4f);
            imagebutton3.setAlpha(0.4f);
        }else{
            //Actualisation des sorties
            etat_sortie[4] = etat_sortie[4]-1;
            //Actualisation des flous
            imagebutton1.setAlpha(0.4f);
        }
    }

    public void Click_feux_croisement(View view){
        if (etat_sortie[4] != 2){
            //Actualisation des sorties
            etat_sortie[4] = 2;
            //Actualisation des flous
            imagebutton1.setAlpha(1.0f);
```

```
            imagebutton2.setAlpha(1.0f);
            imagebutton3.setAlpha(0.4f);
        }else{
            //Actualisation des sorties
            etat_sortie[4] = etat_sortie[4]-1;
            //Actualisation des flous
            imagebutton2.setAlpha(0.4f);
        }
    }

    public void Click_feux_route(View view){
        if (etat_sortie[4] != 3){
            //Actualisation des sorties
            etat_sortie[4] = 3;
            //Actualisation des flous
            imagebutton1.setAlpha(1.0f);
            imagebutton2.setAlpha(1.0f);
            imagebutton3.setAlpha(1.0f);
        }else{
            //Actualisation des sorties
            etat_sortie[4] = etat_sortie[4]-1;
            //Actualisation des flous
            imagebutton3.setAlpha(0.4f);
        }
    }

    public void Click_essuis_glace(View view){
        if (etat_sortie[5] == 0){
            //Actualisation des sorties
            etat_sortie[5] = 1;
            //Actualisation des flous
            imagebutton4.setAlpha(1.0f);
        }else{
            //Actualisation des sorties
            etat_sortie[5] = 0;
            //Actualisation des flous
            imagebutton4.setAlpha(0.4f);
        }
    }

    private void playSound(int resId) {
        if(mPlayer != null){                            //On éteind le média player si il est déjà en route
            mPlayer.stop();
            mPlayer.release();
        }
        mPlayer = MediaPlayer.create(this, resId);      //On joue le son passé en paramètre
        mPlayer.start();
    }

    public void Disconnect(View view) {
        if (btSocket!=null) {       //Si la connection est encore valide
            try {
                btSocket.close();   //Arret de la connection
            }
            catch (IOException e) {
                msg("Erreur pendant l'arrêt de la connection bluetooth");
            }
        }
        finish();                   //Arrêt de l'activité
    }

    private class ConnectBT extends AsyncTask<Void, Void, Void> // UI thread
    {
        private boolean ConnectSuccess = true;
        @Override protected void onPreExecute() {
            progress = ProgressDialog.show(interfacePilotage.this, "Connection...", "Veulliez patienter");
        }
        @Override protected Void doInBackground(Void... devices)
        {
            try
            {
                if (btSocket == null || !isBtConnected) {
                    myBluetooth = BluetoothAdapter.getDefaultAdapter();
                    BluetoothDevice dispositivo = myBluetooth.getRemoteDevice(address);
                    btSocket = dispositivo.createInsecureRfcommSocketToServiceRecord(myUUID);
                    BluetoothAdapter.getDefaultAdapter().cancelDiscovery();
                    btSocket.connect();
                }
            }
            catch (IOException e)
            {
                ConnectSuccess = false;
            }
            return null;
        }
        @Override
        protected void onPostExecute(Void result)
        {
            super.onPostExecute(result);
            if (!ConnectSuccess) {
```

```
                msg("Echec de connection ! L'appareil sélectionné est-il bien celui que vous voulez connecter ?");
                finish();
            }
            else {
                msg("Connection établie !");
                isBtConnected = true;
            }
            progress.dismiss();
        }
    }

    private void msg(String s)
    {
        //Affichage d'un texte passé en paramètre
        Toast.makeText(getApplicationContext(),s,Toast.LENGTH_LONG).show();
    }

    public void sendByBluetooth(View view)
    {

        etat_sortie[1] = acc_value;
        etat_sortie[2] = dir_value + 50;//int DIR[-50,50]

        String accString;
        String dirString;
        if(etat_sortie[1]<10){
            accString = "00"+String.valueOf(etat_sortie[1]);
        }else if(etat_sortie[1]!=100){
            accString = "0"+String.valueOf(etat_sortie[1]);
        }else{
            accString = "100";
        }

        if(etat_sortie[2]<10){
            dirString = "00"+String.valueOf(etat_sortie[2]);
        }else if(etat_sortie[2]!=100){
            dirString = "0"+String.valueOf(etat_sortie[2]);
        }else{
            dirString = "100";
        }

        // "M,ACC,DIR,C,F,E,K"
        String message = String.valueOf(etat_sortie[0])
                + accString
                + dirString
                + String.valueOf(etat_sortie[3])
                + String.valueOf(etat_sortie[4])
                + String.valueOf(etat_sortie[5])
                + String.valueOf(etat_sortie[6]);

        textMessage.setText(message);

        if (btSocket!=null) {
            try {
                //Envoie de données par bluetooth
                btSocket.getOutputStream().write(message.getBytes());
                //btSocket.getOutputStream().write(String.valueOf(acc_value).getBytes());
                msg("Message en cours d'envoi !"+message);
            }
            catch (IOException e) {
                msg("Erreur d'envoi des donnees !");
            }
        }
        else{
            msg("Rien à envoyer vous n'êtes pas connecté !");
        }
    }

    public void defineClock(View view)
    {
        //Création d'un timer pour envoyer les données régulièrement
        final Handler ha=new Handler();
        ha.postDelayed(new Runnable() {

            @Override
            public void run() {

                etat_sortie[1] = acc_value;
                etat_sortie[2] = dir_value + 50;//int DIR[-50,50]
                String accString;
                String dirString;

                if(etat_sortie[1]<10){ accString = "00"+String.valueOf(etat_sortie[1]);
                }else if(etat_sortie[1]!=100){ accString = "0"+String.valueOf(etat_sortie[1]);
                }else{accString = "100";}

                if(etat_sortie[2]<10){ dirString = "00"+String.valueOf(etat_sortie[2]);
                }else if(etat_sortie[2]!=100){dirString = "0"+String.valueOf(etat_sortie[2]);
                }else{ dirString = "100";}

```

```
627                 // "M,ACC,DIR,C,F,E,K"
628                 String message = String.valueOf(etat_sortie[0])
629                         + accString
630                         + dirString
631                         + String.valueOf(etat_sortie[3])
632                         + String.valueOf(etat_sortie[4])
633                         + String.valueOf(etat_sortie[5])
634                         + String.valueOf(etat_sortie[6]);
635
636                 textMessage.setText(message);
637
638                 if (btSocket!=null) {
639                     try {
640                         //Envoie de données par bluetooth
641                         btSocket.getOutputStream().write(message.getBytes());
642                     }
643                     catch (IOException e) {
644                         //msg("Erreur d'envoi des donnees !");
645                     }
646                 }else{
647                     //msg("Rien à envoyer vous n'êtes pas connecté !");
648                 }
649                 ha.postDelayed(this, 50);
650             }
651         }, 50);
652     }
653
654 }
655
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/white"
    android:columnCount="1">


    <ImageButton
        android:layout_row="0"
        android:layout_column="0"
        android:src="@mipmap/logout"
        android:layout_gravity="top|right"
        android:layout_margin="10dp"
        android:layout_width="40dp"
        android:layout_height="50dp"
        android:scaleType="fitCenter"
        android:background="@color/white"
        android:onClick="quitter" />

    <ImageView
        android:src="@mipmap/logo_ecl_roue"
        android:layout_row="0"
        android:layout_column="0"
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="left|top"
        android:layout_marginTop="10dp"
        android:layout_marginLeft="10dp"/>

    <ImageView
        android:src="@mipmap/logo_ecl_forme"
        android:layout_row="0"
        android:layout_column="0"
        android:id="@+id/imageView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="left|top"
        android:layout_marginTop="10dp"
        android:layout_marginLeft="10dp"/>

    <ImageView
        android:src="@mipmap/logo_ecl_texte"
        android:layout_row="0"
        android:layout_column="0"
        android:id="@+id/imageView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="right|bottom"
        android:layout_marginRight="30dp"/>

    <ImageView
        android:scaleType="fitCenter"
        android:src="@mipmap/titre_appli"
        android:layout_height="150dp"
        android:layout_row="1"
        android:layout_column="0"
        android:layout_margin="10dp"
        android:fontFamily="sans-serif-condensed"
        android:id="@+id/imageView_title"
        android:layout_gravity="center"/>

    <Space
        android:layout_row="2"
        android:layout_column="0"
        android:layout_rowSpan="2"
        android:layout_height="40dp"/>



    <ImageButton
        android:background="@drawable/radiusbutton"
        android:src="@mipmap/bouton_appairage_texte"
        android:text="Connecter un véhicule"
        android:scaleType="fitCenter"
        android:layout_height="120dp"
        android:layout_marginRight="30dp"
        android:layout_marginLeft="30dp"
        android:onClick="demarrerConnection"
        android:layout_row="3"
        android:layout_column="0"
        android:id="@+id/button1"
        android:layout_gravity="center_horizontal" />


    <ImageView
        android:src="@mipmap/logo_bee"
        android:layout_row="5"
```

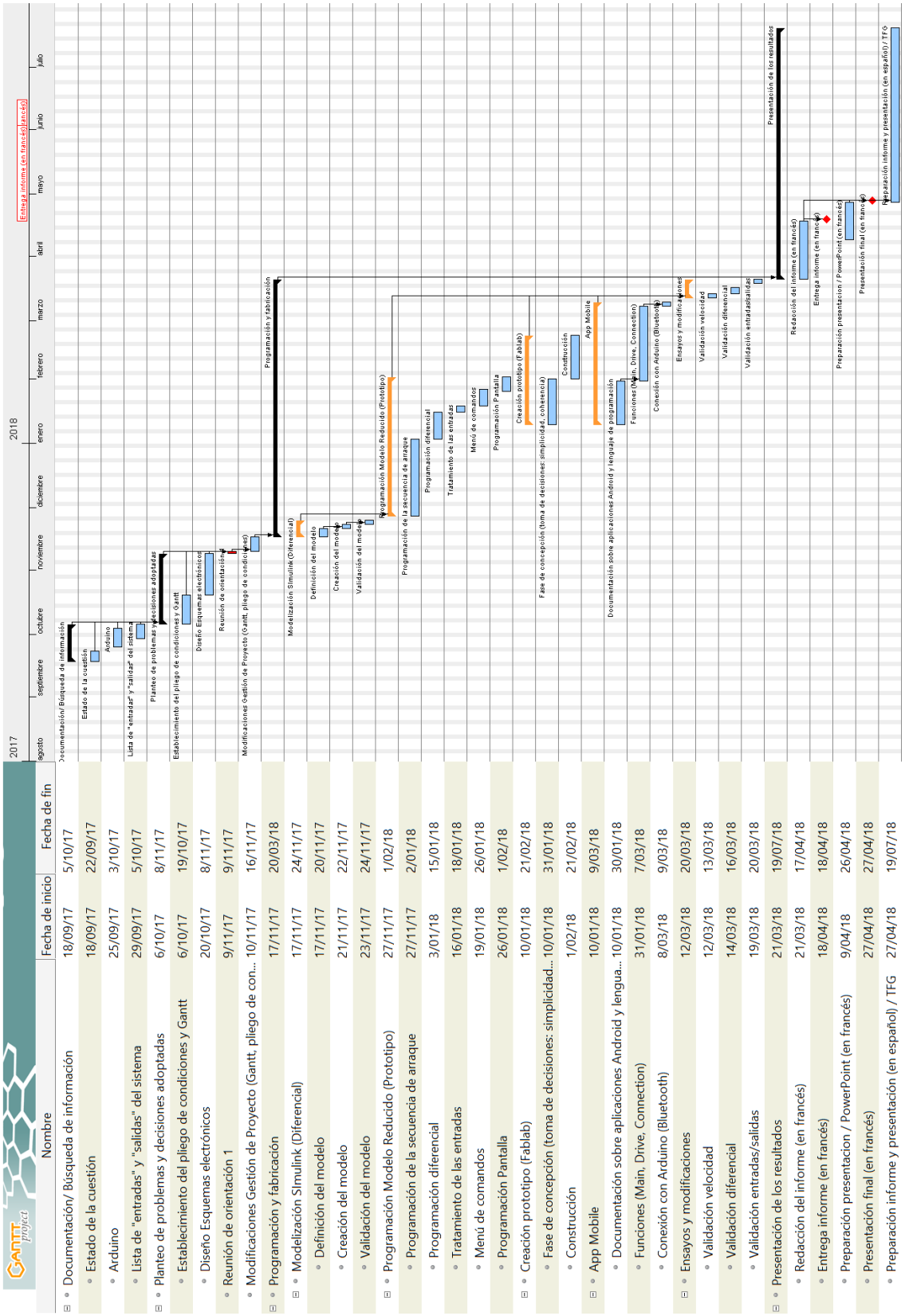
```
91          android:onClick="startPilotage"
92          android:layout_column="0"
93          android:id="@+id/imageView4"
94          android:layout_gravity="bottom|center"/>
95
96  </GridLayout>
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/white"
    android:columnCount="1">


    <GridLayout
        android:layout_row="0"
        android:columnCount="1"
        android:layout_marginTop="20dp"
        android:layout_gravity="center"
        android:layout_column="0">

        <ImageView
            android:src="@mipmap/logo_ecl_roue"
            android:layout_width="120dp"
            android:layout_height="120dp"
            android:scaleType="fitCenter"
            android:id="@+id/imageViewConnect2"
            android:layout_gravity="center"
            android:layout_row="0"
            android:layout_column="0" />

        <ImageView
            android:src="@mipmap/bluetooth"
            android:layout_width="100dp"
            android:layout_height="100dp"
            android:scaleType="fitCenter"
            android:id="@+id/imageViewConnect1"
            android:onClick="startConnection"
            android:layout_gravity="center"
            android:layout_row="0"
            android:layout_column="0" />

        <Button
            android:text="Commencer"
            android:textColor="@color/red_centrale"
            android:onClick="startConnection"
            android:layout_row="1"
            android:layout_column="0"
            android:id="@+id/buttonConnect"
            android:layout_marginTop="10dp"
            android:layout_gravity="center"/>

        <TextView
            android:id="@+id/textViewPick"
            android:visibility="invisible"
            android:layout_row="1"
            android:layout_column="0"
            android:layout_marginTop="10dp"
            android:text="Veulliez choisir votre voiture..."
            android:textColor="@color/red_centrale"
            android:textStyle="bold"
            android:layout_gravity="center"/>

    </GridLayout>

    <ListView
        android:id="@+id/listViewConnect"
        android:layout_row="1"
        android:layout_column="0"
        android:cacheColorHint="@color/red_centrale"/>

</GridLayout>
```

Parte III:

Anexos

A. Cronograma



Nombre	Fecha de inicio	Fecha de fin
Documentación/ Búsqueda de información	18/09/17	5/10/17
Estado de la cuestión	18/09/17	22/09/17
Arduino	25/09/17	3/10/17
Lista de "entradas" y "salidas" del sistema	29/09/17	5/10/17
Planteo de problemas y decisiones adoptadas	6/10/17	8/11/17
Establecimiento del pliego de condiciones y Gantt	6/10/17	19/10/17
Diseño Esquemas electrónicos	20/10/17	8/11/17
Reunión de orientación 1	9/11/17	9/11/17
Modificaciones Gestión de Proyecto (Gantt, pliego de con...	10/11/17	16/11/17
Programación y fabricación	17/11/17	20/03/18
Modelización Simulink (Diferencial)	17/11/17	24/11/17
Definición del modelo	17/11/17	20/11/17
Creación del modelo	21/11/17	22/11/17
Validación del modelo	23/11/17	24/11/17
Programación Modelo Reducido (Prototipo)	27/11/17	1/02/18
Programación de la secuencia de arraque	27/11/17	2/01/18
Programación diferencial	3/01/18	15/01/18
Tratamiento de las entradas	16/01/18	18/01/18
Menú de comandos	19/01/18	26/01/18
Programación Pantalla	26/01/18	1/02/18
Creación prototipo (Fablab)	10/01/18	21/02/18
Fase de concepción (toma de decisiones: simplicidad...	10/01/18	31/01/18
Construcción	1/02/18	21/02/18
App Mobile	10/01/18	9/03/18
Documentación sobre aplicaciones Android y lengua...	10/01/18	30/01/18
Funciones (Main, Drive, Connection)	31/01/18	7/03/18
Conexión con Arduino (Bluetooth)	8/03/18	9/03/18
Ensayos y modificaciones	12/03/18	20/03/18
Validación velocidad	12/03/18	13/03/18
Validación diferencial	14/03/18	16/03/18
Validación entradas/salidas	19/03/18	20/03/18
Presentación de los resultados	21/03/18	19/07/18
Redacción del informe (en francés)	21/03/18	17/04/18
Entrega informe (en francés)	18/04/18	18/04/18
Preparación presentacion / PowerPoint (en francés)	9/04/18	26/04/18
Presentación final (en francés)	27/04/18	27/04/18
Preparación informe y presentación (en español) / TFG	27/04/18	19/07/18

Figura 50: Diagrama de Gantt

B. Imágenes del vehículo Mobee'City

Figura 51: Estado actual del vehículo Mobee'City

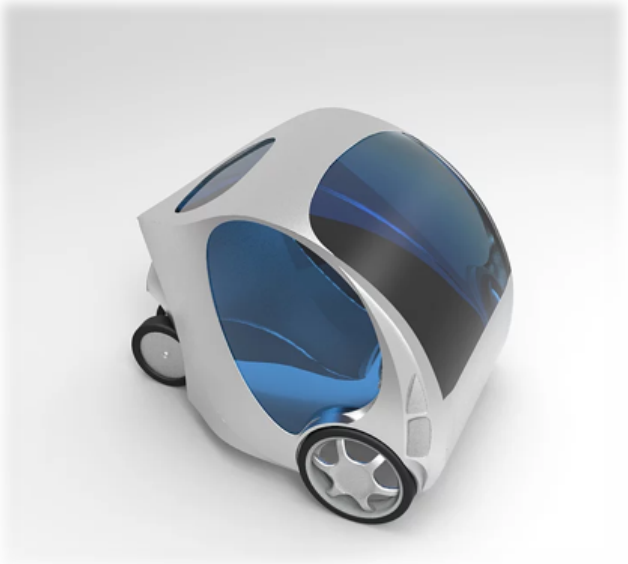


Figura 52: Diseño del Vehículo Mobee'City

C. Modelo Simulink Completo

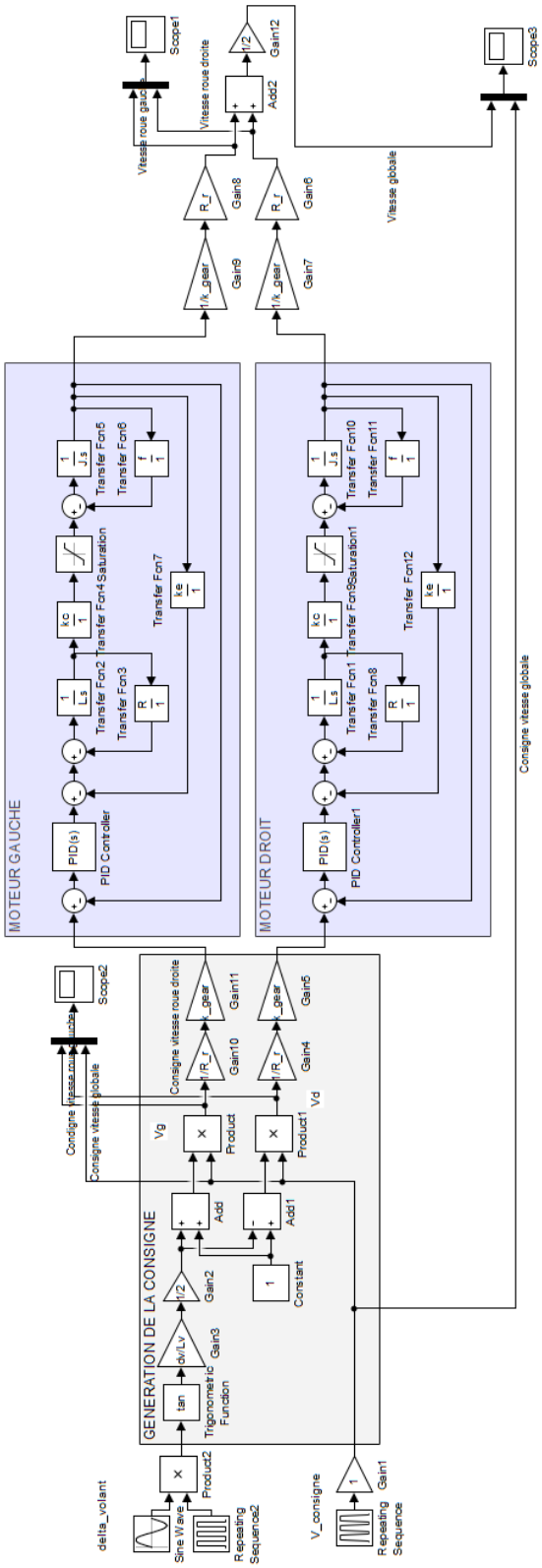


Figura 53: Modelo Simulink completo

D. Pinout placa Arduino Mega

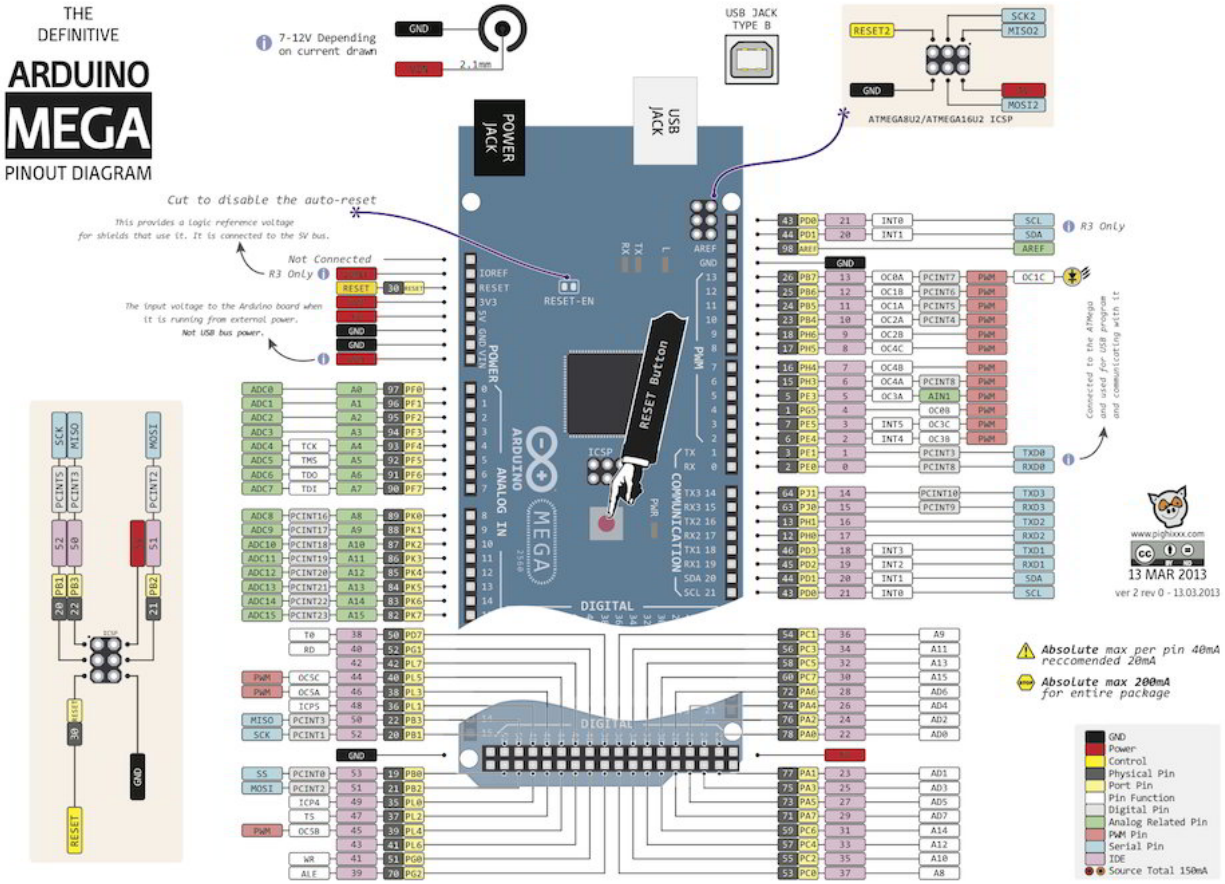


Figura 54: Pinout placa Arduino Mega

E. Vector param

Variable	Signification	Valeurs possibles	Explication des valeurs
param[0]	Vitesse	[0, 99]	Vitesse réelle du véhicule en km/h
param[1]	Consigne de vitesse	[0 , 255]	Vitesse de consigne mesurée grâce à la pédale
param[2]	Frein	{0, 1}	Intensité du freinage mesuré grâce à la pédale
param[3]	Volant	[0, 255]	Angle de virage mesuré grâce au volant
param[4]	Frein à main	{0, 1}	0 : Frein à main déverrouillé 1 : Frein à main verrouillé
param[5]	Clef de contact	{0, 1}	0 : La clef n'est pas tournée 1 : La clef est pas tournée
param[6]	Ceinture de sécurité	{0, 1}	0 : La ceinture n'est pas mise 1 : La ceinture est mise
param[7]	Présence conducteur	{0, 1}	0 : Le conducteur n'est pas détecté 1 : Le conducteur est détecté
param[8]	Sens de circulation	{0, 1, 2}	0 : Véhicule au point mort 1 : Véhicule en marche avant 2 : Véhicule en marche arrière
param[9]	Éclairage intérieur	{0, 1}	0 : Éclairage éteint 1 : Éclairage allumé
param[10]	Feux	{0, 1, 2}	0 : Les feux sont éteints 1 : Feux de position 2 : Feux de croisement
param[11]	Essuie-glaces	{0, 1}	0 : Eteint 1 : Allumés
param[12]	Klaxon	{0, 1}	0 : Eteint 1 : Allumés
param[13]	Clignotants	{0, 1, 2, 3}	0 : Les clignotants sont éteints 1 : Clignotant gauche allumé 2 : Clignotant droit allumé 3 : Feux de détresse
param[14]			
param[15]	Température	[-99, 99]	Température mesurée
param[16]	Niveau de la batterie	[0, 7]	0 : Batterie complètement déchargée 7 : Batterie complètement chargée
param[17]	Mode d'affichage	{0, 1, 2}	0 : Affichage des paramètres de conduite 1 : Affichage du logo Mobee'City 2 : Affichage du logo Centrale Lyon
param[18]			
param[19]			

F. Hacer parpadear los intermitentes

Para evitar utilizar la función `delay()`, que obligaría al programa a pararse momentáneamente, la solución alternativa elegida ha sido utilizar la función `millis()`. Esta función envía el valor del reloj interno del Arduino en ms. Cada vez que se ejecuta la función, se realiza la siguiente operación:

$$\texttt{millis()} \ \% \ 666 > 333 \tag{20}$$

Si el resultado de esa operación booleana es `true`, entonces se enciende la luz intermitente. Si el resultado es `false`, la luz se apaga.

Sin embargo, este método tiene sus límites. En efecto, si se enciende esa luz de manera demasiado ocasional, el intermitente puede adoptar un comportamiento caótico y no respetar el criterio de frecuencia establecido en el pliego de condiciones.

La Figura 55 muestra el resultado obtenido para una frecuencia de 20 Hz, y la figura 56 para 2Hz.

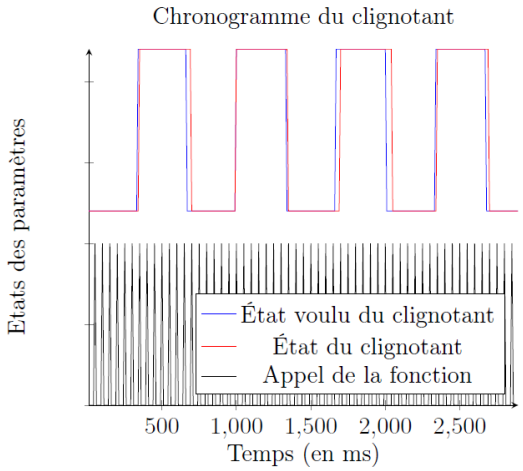


Figura 55: Cronograma de un intermitente para una frecuencia de llamada de 20 Hz

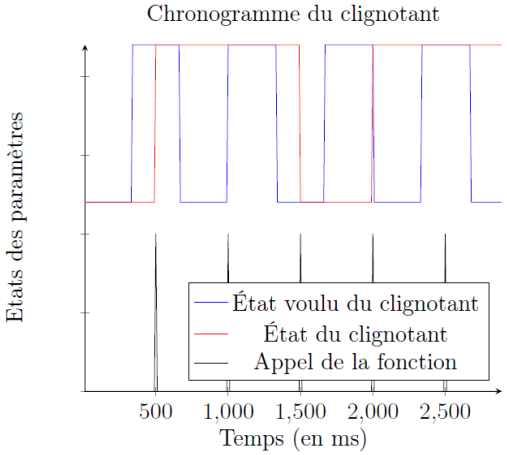


Figura 56: Cronograma de un intermitente para una frecuencia de llamada de 2 Hz

De esta manera, se puede observar que para ciertos valores de frecuencia, el intermitente funcionará correctamente (por ejemplo 20 Hz). Para otros valores (2 Hz), el intermitente no tiene un comportamiento adecuado.

Pasando por un algoritmo que permite evaluar el error cometido entre una señal totalmente rectangular de periodo de 666 ms y la señal obtenida, se puede trazar el error en función de la frecuencia especificada en la función. El resultado se presenta en la Figura 57.

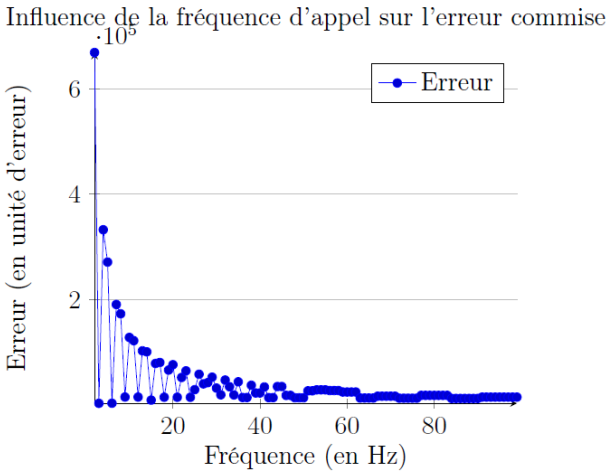


Figura 57: Influencia de la frecuencia de llamada en el error cometido

El número de unidades de error aceptable se sitúa alrededor de 50000. A ese número, el ojo humano no percibe las variaciones temporales en el parpadeo de la luz. Se establecerá un tiempo inferior a 37 ms para llamar a la función `set_digital_output`.

G. El panel de control para el banco de ensayo

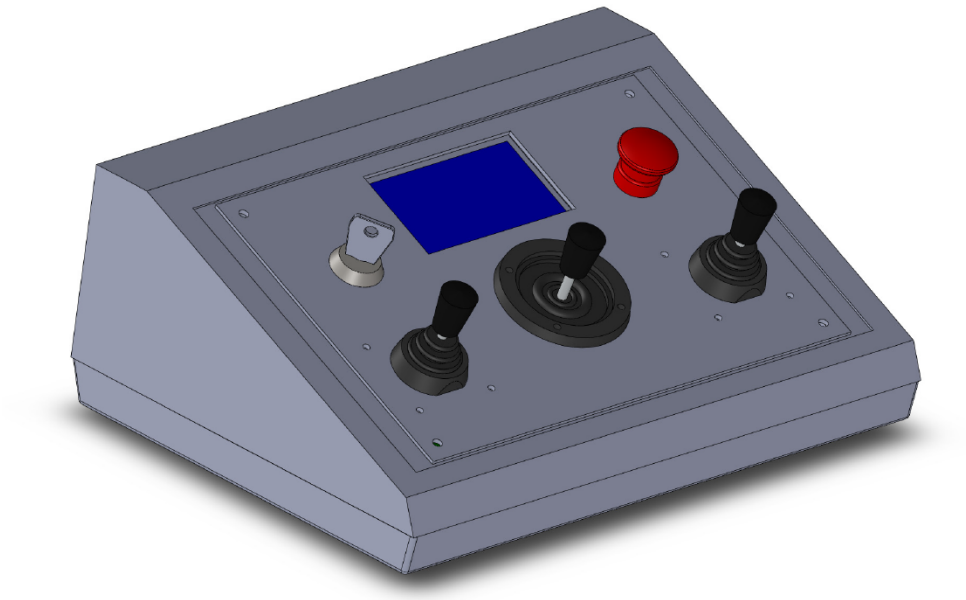


Figura 58: Panel de control

Se ha decidido simular el cuadro de mando con un panel de control, diseñado por M. Perrard. EL panel tiene una pantalla LCD, dos joysticks laterales que funcionan como interruptores de 4 estados y el joystick central tiene dos potenciómetros en el eje horizontal y el vertical. Detrás tiene tres entradas para poder conectar los distintos elementos del coche. En el interior se colocará la placa Arduino y los dispositivos electrónicos necesarios.

El curso siguiente, un alumno de la École Centrale de Lyon se encargará de la construcción de este panel de control y la implantación de la electrónica en él. Podrá utilizar como base el código Arduino que se ha compezado a desarrollar este año, en el que se definen las entradas de los diferentes joysticks en la placa (ver el extracto de código 19) y las funciones necesarias para el control del coche.

```
1  //-------------------------- PIN DECLARATION--------------------------------------
2  // INPUT PINS
3  int pin_turn_signal_left  = 48;          // Left Joystick Left
4  int pin_highlight         = 46;          // Left Joystick TOP
5  int pin_turn_signal_right = 47;          // Left Joystick Right
6  int pin_warnings          = 45;          // Left Joystick Bottom
7
8  int pin_direction         = A15;         // Middle Joystick X axis
9  int pin_speed             = A14;         // Middle Joystick Y axis
10 int pin_break             = A14;         // Middle Joystick Y axis
11
12 int pin_wipers            = 44;          // Right Joystick Left
13 int pin_klaxon            = 43;          // Right Joystick TOP
14 int pin_int_light         = 42;          // Right Joystick Right
15 int pin_41                = 41;          // Right Joystick Bottom
16
17 int pin_tech_button_left  = 51;          // Technician module button left
```

```
18 int pin_tech_button_right = 50;          // Technician module button right
19 int pin_tech_button_top   = 53;          // Technician module button top
20 int pin_tech_button_bottom= 52;          // Technician module button bottom
21 int pin_tech_button_enter = 49;          // Technician module button enter
22
23 int pin_sensor_temperature= A8;          // Temperature sensor pin
24 int pin_sensor_light      = A9;          // Light sensor pin
```

Código 19: Declaración de los PINs de entrada Arduino del panel de control

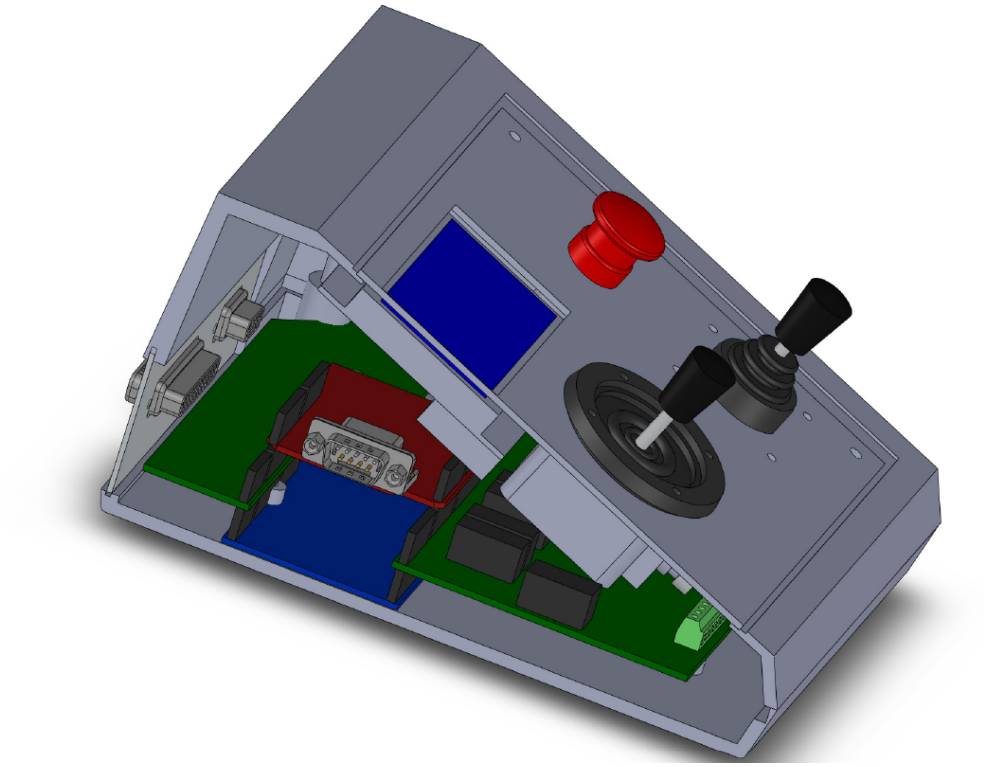
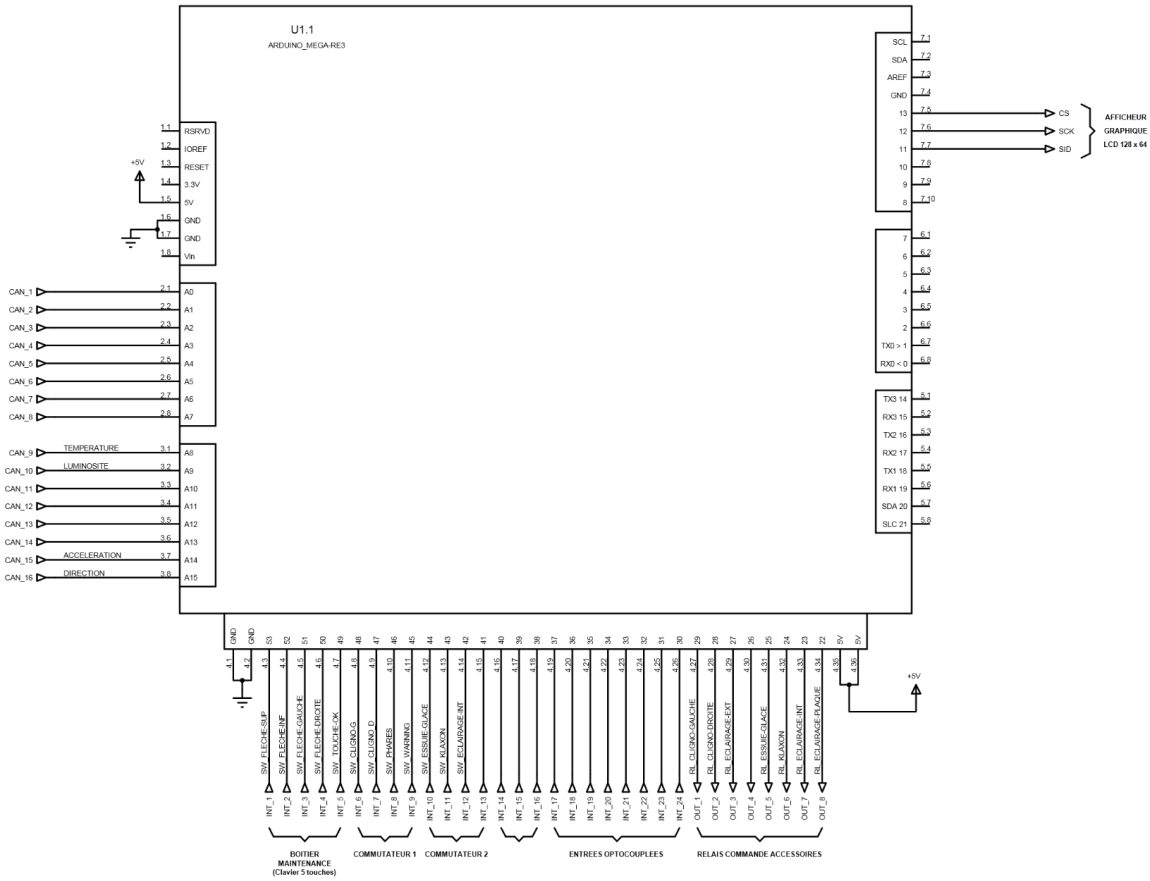


Figura 59: Corte vertical del Panel de control

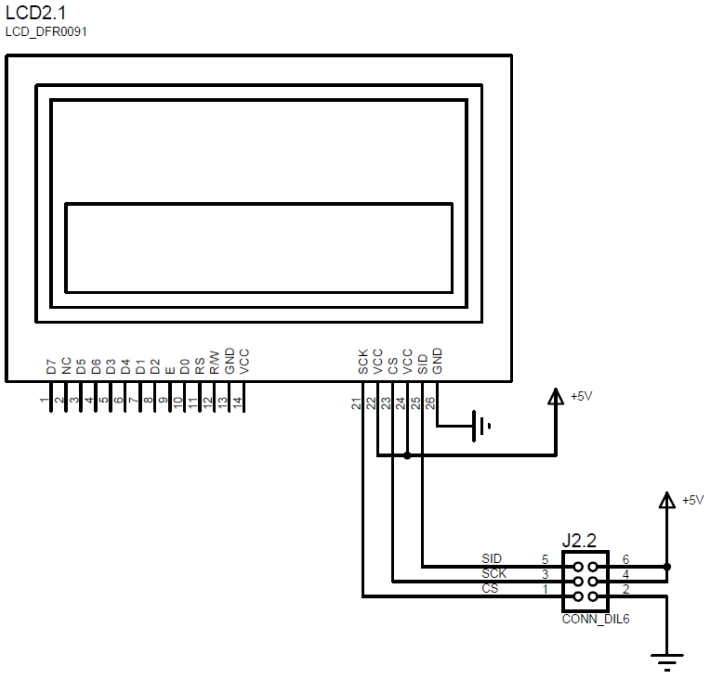
DOCUMENTO 2. PLANOS

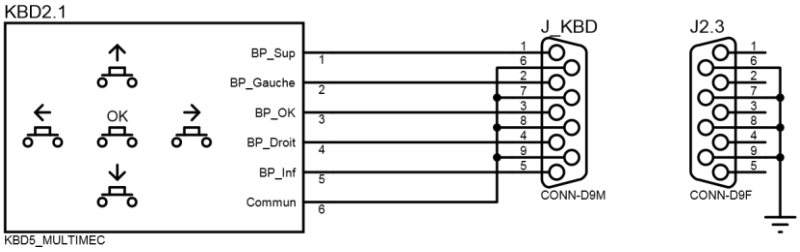
Lista de planos

- Diagrama Modulo Arduino MEGA-RE3
- Diagrama Módulo pantalla LCD
- Diagrama Entradas optoacopladas
- Diagrama Entradas no optoacopladas
- Diagrama salidas relés

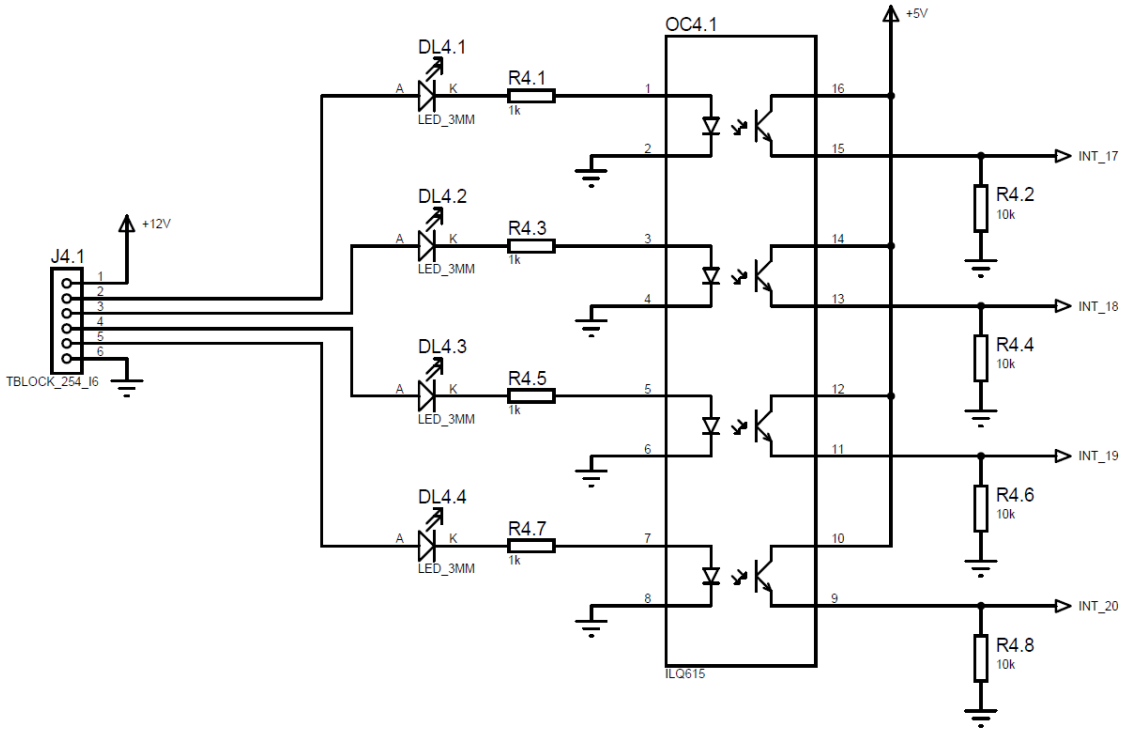


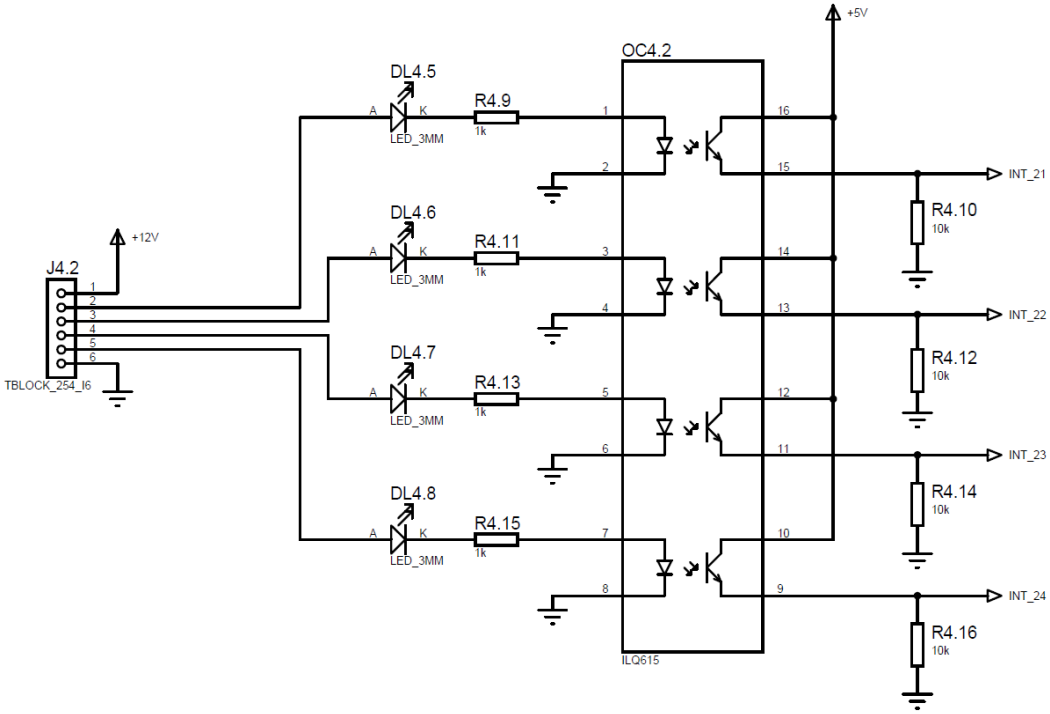
MATERIAL	-		*Poste de pilotage CI « MBC _PC01 »*
TOLERANCIA	-		
	NOMBRE	FECHA	*Modulo Arduino MEGA-RE3*
DIBUJADO	Proyecto Mobee'City	24/10/2017	*Esquema principal*
COMPROBADO	Proyecto Mobee'City	27/02/2018	
ESCALA: *S/E*	FIRMA: P MVC	*I.C.A.I.*	Nº DE LÁMINA: *1*



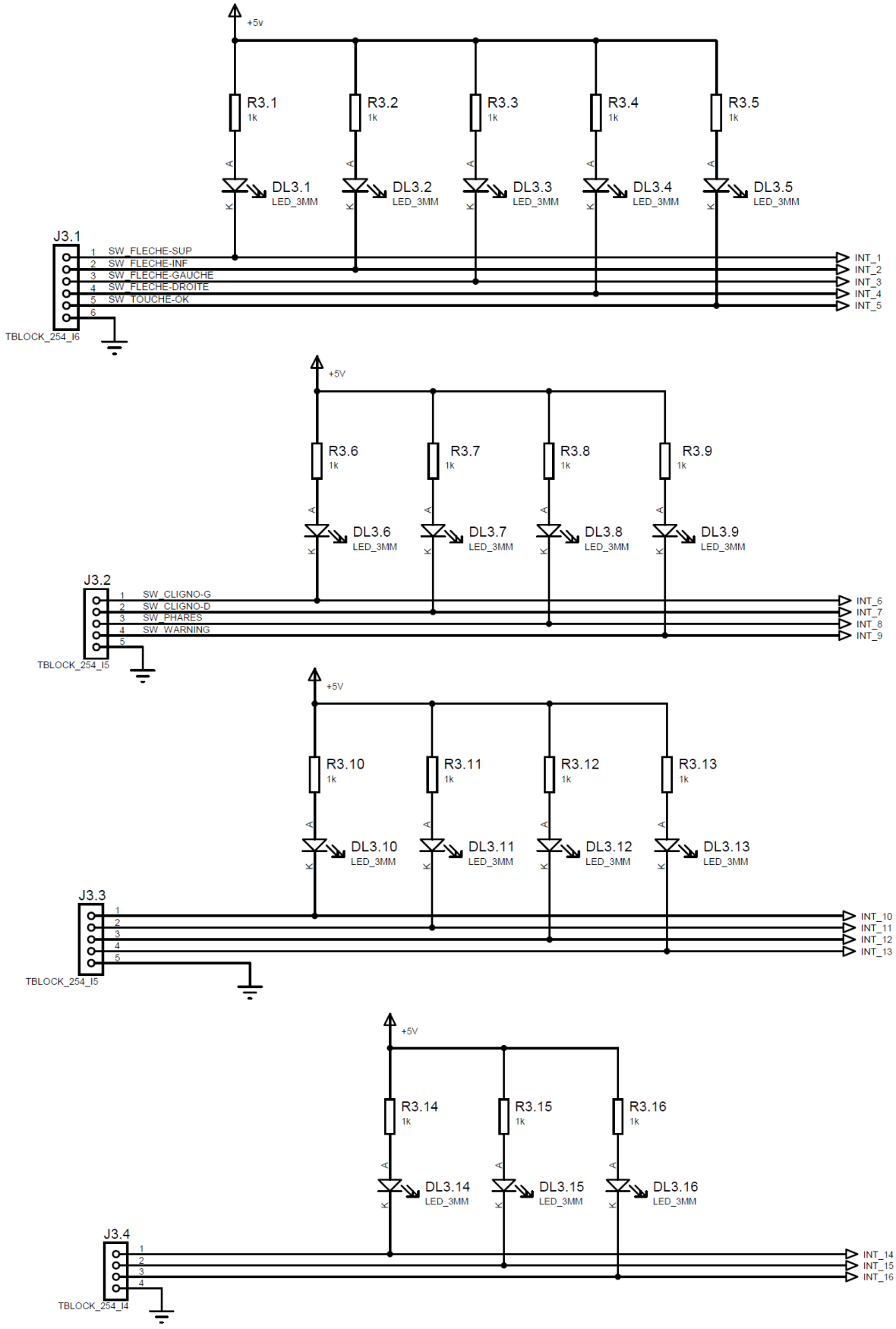


MATERIAL	-		*Poste de pilotage CI « MBC _PC01 »*
TOLERANCIA	-		
	NOMBRE	FECHA	*Modulo Pantala LCD*
DIBUJADO	Proyecto Mobee'City	24/10/2017	*Diagrama*
COMPROBADO	Proyecto Mobee'City	27/02/2018	
ESCALA: *S/E*	FIRMA: P MVC	*I.C.A.I.*	Nº DE LÁMINA: *1*

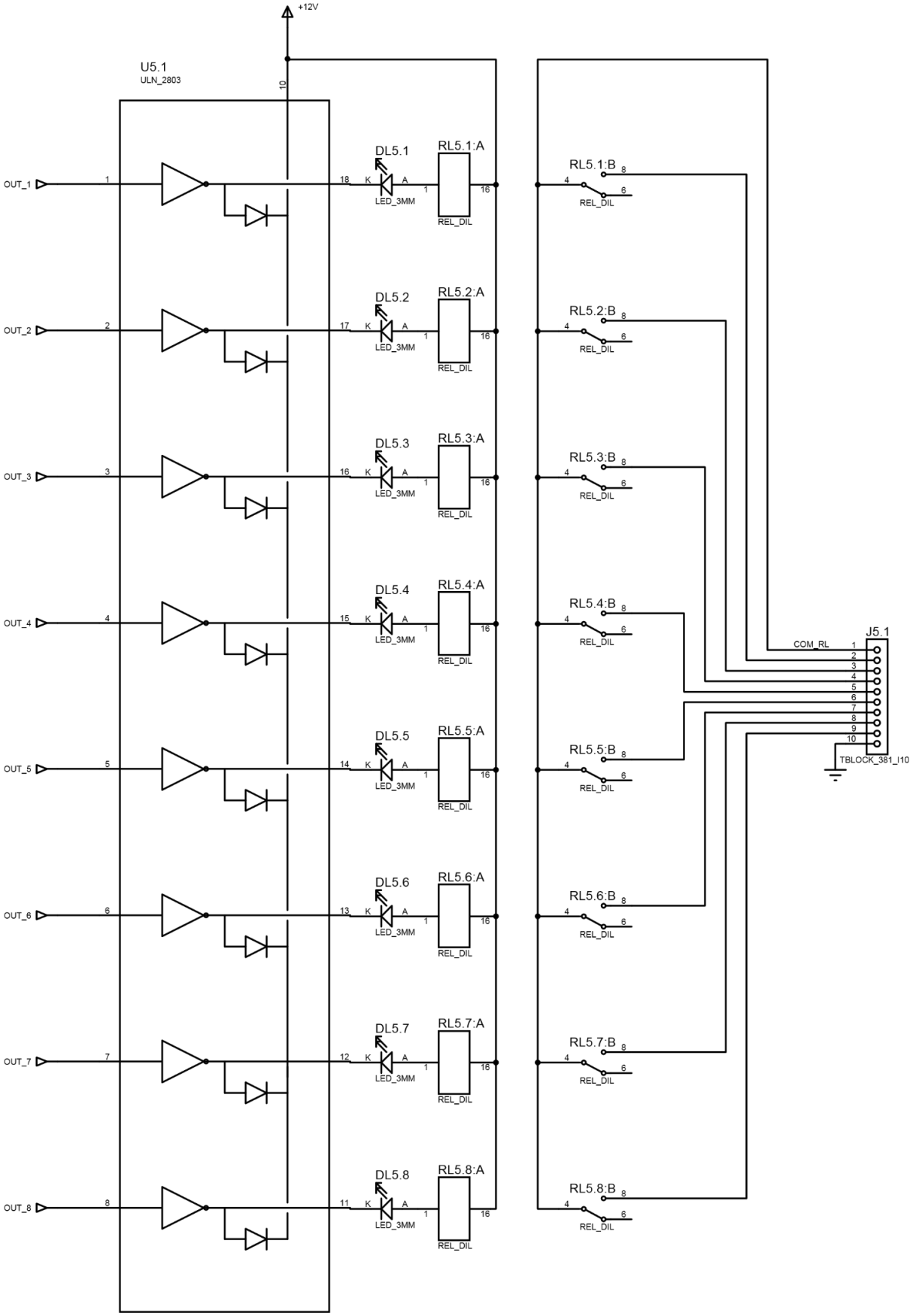




MATERIAL	-		**Poste de pilotage CI « MBC _PC01 »**
TOLERANCIA	-		
	NOMBRE	FECHA	**Entradas optoacopladas Diagrama**
DIBUJADO	Proyecto Mobee'City	24/10/2017	
COMPROBADO	Proyecto Mobee'City	27/02/2018	
ESCALA: **S/E**	FIRMA: P MVC		**I.C.A.I.** Nº DE LÁMINA: **3**



MATERIAL	-		***Poste de pilotage CI « MBC _PC01 »***
TOLERANCIA	-		
	NOMBRE	FECHA	***Entradas no optoacopladas Diagrama***
DIBUJADO	Proyecto Mobee'City	24/10/2017	
COMPROBADO	Proyecto Mobee'City	27/02/2018	
ESCALA: ***S/E***	FIRMA: P MVC	***I.C.A.I.***	Nº DE LÁMINA: ***4***



MATERIAL	-		***Poste de pilotage CI « MBC _PC01 »***
TOLERANCIA	-		
	NOMBRE	FECHA	***Salidas relés***
DIBUJADO	Proyecto Mobee'City	24/10/2017	***Diagrama***
COMPROBADO	Proyecto Mobee'City	27/02/2018	
ESCALA: ***S/E***	FIRMA: P MVC	***I.C.A.I.***	Nº DE LÁMINA: ***5***