Universidad de Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN.

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

ÁREA DE INGENIERÍA TELEMÁTICA

TRABAJO FIN DE MÁSTER Nº 202010

SISTEMA MULTISENSORIAL DE ASISTENCIA A LA CONDUCCIÓN EN VEHÍCULOS INDUSTRIALES

D. MONTES ANTUÑA, Javier
TUTOR: D. GARCÍA FERNÁNDEZ, Roberto
TUTOR: D. CORCOBA MAGAÑA, Víctor

FECHA: Julio 2020

Agradecimientos

Al grupo de investigación SMIOT por darme la oportunidad de participar de forma activa en un proyecto de este tipo; a Roberto y Xicu por ofrecerme realizar el TFM con el grupo, y a Víctor y Alejandro por su ayuda.

A mis compañeros del máster Carlos Tristán González, Alejandor Casanova y Enrique Álvarez-Cofiño por soportarme estos dos años de máster y ayudarme siempre que pudieron.

A mis compañeros de grado Borja Imaz, Alberto Álvarez, David Álvarez, Jaime Suárez, Miguel Ángel Solís, Miguel Fernández, Marcos Riestra por soportar mis abundantes quejas durante estos dos años.

A mi familia, por estar ahí siempre que los necesité.

Y por último pero no menos importante, a mis amigos de siempre, Andrés Arbesú, Andrés Vivo, Jorge Vigil, Pelayo Montequín, Jaime Alonso, Eduardo García, Juan Álvarez, Diego Alonso, Patricia Fernández, Carmen Vigil, Laura García, Irene Coco por preocuparse por mi e intentar subirme el ánimo siempre que podían.

Índice general

Índice de figuras

Lista de scripts

1. Introducción

Este TFM está realizado como una memoria de prácticas en el grupo de investigación SMIOT perteneciente al departamento de Informática. Dentro del grupo se han realizado trabajos en dos enfoques:

- Enfoque comercial: se ha desarrollado una placa de sensores para recoger datos del interior de vehículos comerciales. Además, se añadirá una banda de pulsaciones para monitorizar al conductor y un conector OBD2 para recoger datos del vehículo.
- Enfoque industrial: se ha investigado un sistema de detección de obstáculos y localización de vehículos industriales. Además, se ha diseñado una interfaz gráfica doble (pantalla y LEDs). Por último, se ha desarrollado un estudio para conocer la mejor forma de mostrar la información al usuario.

1.1.- Tecnologías usadas

En ambos casos las tecnologías usadas son comunes:

- Python: como lenguaje de programación se ha usado python, en concreto la versión 3.7, puesto que es la que viene por defecto en el sistema operativo de la Raspberry (Raspbian) y es uno de los lenguajes más fáciles de usar con ella. Además, todos los sensores usados con la Raspberry Pi tienen sus librerías desarrolladas en Python, haciendo más sencillo el desarrollo de programas para la recogida de datos.

Figura 1.1.- Python

- SQLite3: para el almacenamiento de datos en la raspberry pi se ha usado SQLite3 por la poca capacidad de cómputo que se necesita para usar este motor de base de datos. Otras opciones que se han tenido en cuenta son MySQL o MariaDB, pero se ha heredado este motor del proyecto tal y como estaba con la Raspberry Pi 3.

Figura 1.2.- SQLite3

- SQL: como motor de base de datos en el servidor remoto se ha usado SQL que está instalado por defecto en los servidores Windows 10. Este motor ha sido elegido por el grupo desde el principio del proyecto.

Figura 1.3.- SQL

- Matlab: como programa para el análisis de datos se ha usado Matlab debido a la facilidad para realizar gráficas y la posibilidad de obtener los datos automáticamente a través del archivo .db que genera SQLite3.

Figura 1.4.- Matlab

- Eagle: para el diseño de la placa de circuito impreso (PCB) del proyecto se ha usado el programa Eagle, propiedad de Autodesk. Se ha utilizado por la gran cantidad de librerías de componentes de la que dispone, familiaridad en la utilización y por ser gratuito para estudiantes.

Figura 1.5.- Eagle

1.2.- Equipamiento utilizado

El equipamiento utilizado en este proyecto, sin contar con los elementos de la placa explicados en el capítulo 2, se explican a continuación:

- Raspberry Pi: como dispositivo embarcado para el procesamiento de datos para ambos proyectos se ha usado una Raspberry Pi 4 de 4 Gigas. Es un PC en «miniatura». Presenta todas las funcionalidades de un PC de sobremesa y tiene como sistema operativo Raspbian, una distribución más «ligera» de Linux, pero con unas funciones casi idénticas. Se ha usado por la compatibilidad con las GPIOs con la Raspberry Pi 3, de la que ya se disponía de una placa construida, y por la mayor capacidad de computación que aporta en comparación con la 3 (pasa de 1 GB de RAM a 4GB). Además, el consumo que tiene la Raspberry es pequeño y se puede alimentar a través del mechero del coche o de una batería portátil.

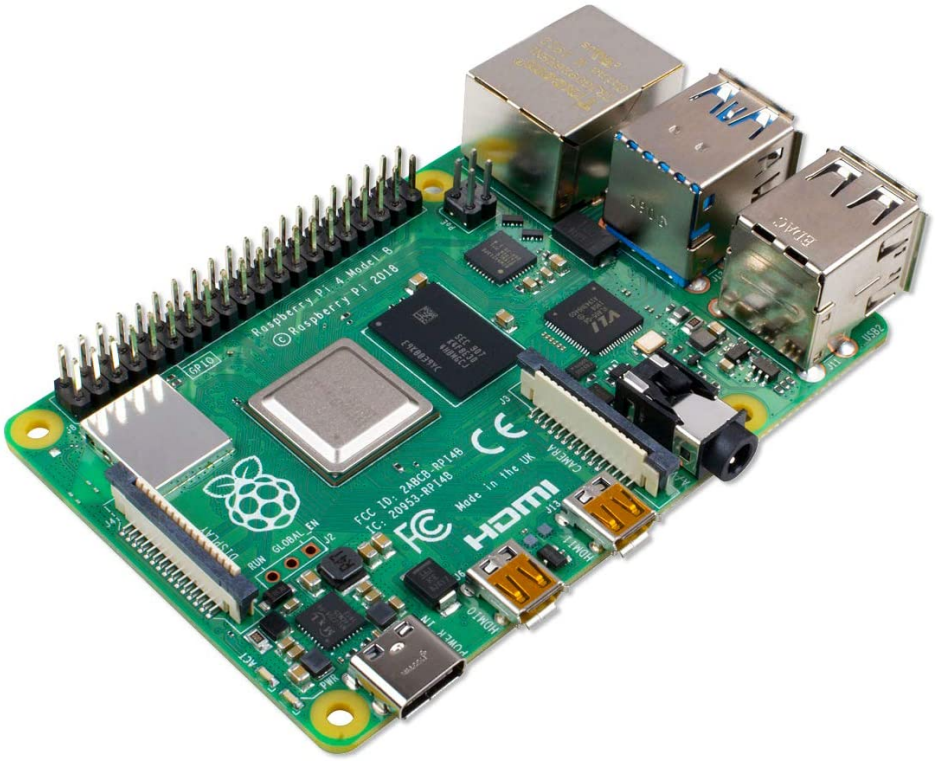

Figura 1.6.- Raspberry Pi 4

- Banda pectoral Polar H10: para medir las pulsaciones del conductor se ha usado una banda Polar H10. Se ha usado esta banda por la sencillez de programación y la API abierta que tiene para sacar la información tanto de las pulsaciones como del intervalo R-R.

Figura 1.7.- Polar H10

- Conector OBD2: para conseguir la información del vehículo se ha utilizado un conector OBD2 cable link. El hecho de que sea una conexión por cable lo hace más molesto, pero no se ha conseguido que las alternativas Bluetooth funcionasen de manera consistente.

Figura 1.8.- Conector OBD2 bluetooth

- Tiras LED: para mostrar al usuario industrial las alertas de una forma más visual y sencilla se usarán tiras LED como las que se muestran en la Figura 1.9.

Figura 1.9.- Tiras LED usadas en la interfaz gráfica

2. Parte comercial

En este capítulo se hablará del proyecto de ámbito comercial del grupo de investigación. Se empezará por una explicación del proyecto en conjunto, siguiendo con el diseño de la PCB y se continuará por el desarrollo de un script que agiliza el análisis de datos, juntando todos los datos en el servidor remoto en una única tabla con la misma marca de tiempo. Por último se mostrarán los datos y las gráficas de una pruebas de viaje.

2.1.- Introducción

En este proyecto se pretende diseñar y fabricar una placa para la recogida de datos del entorno dentro de un vehículo comercial, del propio conductor y del vehículo en sí. Estos datos serán analizados para crear un perfil de conducción, único por persona. Con estos perfiles y un conjunto de reglas diseñadas dentro del grupo de investigación, se harán recomendaciones al conductor con el objetivo de optimizar la conducción y hacerla más segura.

Entre los datos que se recogerán del entorno están: la temperatura, la humedad, la presión, la calidad del aire (partículas por millón de CO2), aceleraciones y ángulos en todos los ejes, nivel de ruido y geolocalización del vehículo; del conductor se recogerán datos sobre sus pulsaciones y sus intervalos R-R; y del vehículo se recogerán datos a través del conector OBD, que está conectado a la centralita del vehículo (ECU). Los datos recogidos del OBD serán, entre otros, las revoluciones por minuto (RPM), la velocidad, la posición de los pedales de freno y aceleración, el nivel de combustible, el ratio de consumo de combustible o la presión del combustible.

2.2.- PCB de sensores

En esta sección se explicará cómo se ha diseñado la PCB de los sensores, hablando primeramente qué elementos se han utilizado, siguiendo por sus conexiones en un esquemático y terminando por cómo se ha diseñado la PCB.

2.2.1.- Sensores utilizados

Para la realización de la PCB se han usado un total de 5 sensores y un GPS, además de un conversor A/D, 2 conectores USB-A, 2 condensadores de tipo SMD y un conector tipo HAT para la Raspberry Pi.

2.2.1.1.- Sensor de ruido

Para el sensor de ruido se ha usado el Sparkfun Sound Detector (Figura 2.1). Es un sensor de tipo analógico que está alimentado a 3.3V y devuelve la envolvente del ruido que capta. Se conectada a un conversor A/D, un MCP3008 (Figura 2.2), que se conecta a los pines MOSI MISO (parte de protocolo síncrono SPI integrado) de la Raspberry Pi. Este sensor necesita de una calibración previa usando un sonómetro como el mostrado en la Figura 2.3. Para las primeras pruebas se ha usado un teléfono móvil para calibrarlo. Se ha elegido este sensor en vez de otro micrófono digital por la sencillez de programación y por recomendaciones de una empresa con la que trabaja el grupo de investigación que implementan un sistema similar al de este proyecto.

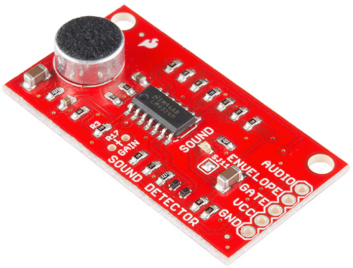

Figura 2.1.- Sensor de ruido Sparkfun Sound Detector

Figura 2.2.- Conversor A/D MCP3008

Figura 2.3.- Sonómetro usado para la calibración futura del sensor

2.2.1.2.- GPS

Para la posición del vehículo se ha usado el GY-NEO6MV2 (Figura 2.4), a partir del cual se pueden obtener la latitud, la longitud y la altitud. El GPS se alimenta a 3.3V y está conectado a los pines TX y RX de la Raspberry Pi, cruzados, es decir, el Tx del GPS al Rx de la Raspberry, y el Rx del GPS al Tx. Se ha elegido este sensor por la sencillez de programación y por el bajo precio. En comparación con otros sensores más caros, no presenta ninguna ventaja significativa que justifique la diferencia de precio.

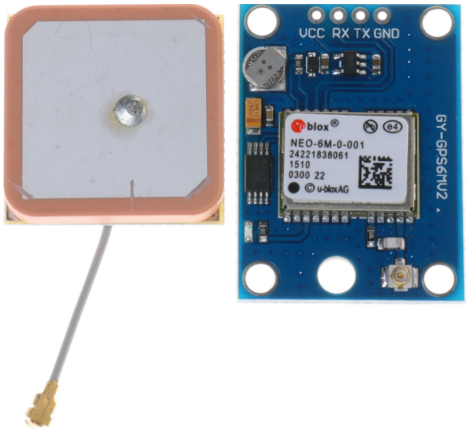

Figura 2.4.- GPS GY-NEO6MV2

2.2.1.3.- Acelerómetro

Como acelerómetro se ha seleccionado el Adafruit LSM9DS1 (Figura 2.5), que devuelve valores de aceleración en metros por segundo cuadrado, además de ser magnetómetro y giroscopio. Es un sensor I2C, conectado a los pines SCL y SDA de la Raspberry Pi, y alimentado a 3.3V. Se ha elegido este acelerómetro porque, además de devolver las aceleraciones en todos los ejes, devuelve los ángulos gracias a su giroscopio. De esta forma, se podrá analizar si el vehículo está subiendo o bajando una cuesta, o si se ha encontrado con algún bache.

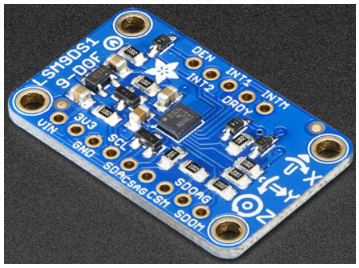

Figura 2.5.- Acelerómetro LSM9DS1

2.2.1.4.- Sensor de calidad de aire

Para evaluar la calidad del aire se ha usado el sensor CCS811 de Adafruit (figura 2.6), un sensor I2C que devuelve los valores de CO2 en partículas por millón (ppm) y TVOC en partículas por billón. Al ser un sensor I2C, también está conectado a los pines SDA y SCL de la Raspberry Pi, y está alimentado a 3.3V. Se ha usado un sensor de calidad de aire digital en vez de uno analógico porque el analógico necesita de una calibración y, una vez calibrado, los valores suelen no ser tan exactos como con un sensor digital.

Figura 2.6.- Sensor de calidad de aire CCS811 de Adafruit

2.2.1.5.- Sensor de luz

Como sensor de luz se ha usado el BH1750 (Figura 2.7), un sensor I2C que funciona a 3.3V y devuelve el valor actual de luz en lúmenes. Se ha usado este sensor en vez de una foto resistencia al uso porque se quieren medir los valores de luz exactos. Con una foto resistencia se necesitaría, primero, calibrarla y, como se ha comprobado con la placa anterior del proyecto, los valores que devuelve pueden ser completamente erróneos (como datos de 20000 lúmenes estables durante un periodo apreciable de tiempo).

Figura 2.7.- Sensor de Luz BH1750 de AZ Delivery

2.2.1.6.- Sensor de Temperatura, Humedad y Presión

Para conseguir los valores de temperatura, humedad y presión se ha usado el sensor BME280 (Figura 2.8), un sensor I2C que funciona a 3.3V y devuelve valores de temperatura en grados centígrados (ºC), de humedad relativa (HR) en tanto por

ciento (%) y presión en hectopascales (hPa). Se ha elegido este sensor por su precio, su sencillez de programación y la variedad de datos que aporta. La diferencia con un sensor más caro no es significativa, sólo tendría una sensibilidad algo menor y, en términos de temperatura, por ejemplo, 0,05 ºC no son significativos dentro de un vehículo.

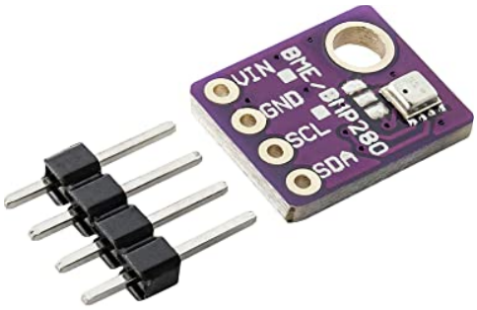

Figura 2.8.- Sensor de temperatura, humedad y presión BME280 de AZ Delivery

2.2.1.7.- Circuito de alimentación

Para alimentar el circuito se ha usado un conector USB-A (Figura 2.9), el regulador 7833 de 3.3V (Figura 2.10) y 2 condensadores, de 0.33 uF y de 0.1 uF, conectados de la forma en que se muestra en la Figura 2.11. El otro conector USB-A se usa para alimentar la propia Raspberry Pi. Cualquiera de los dos conectores se puede usar indistintamente para alimentar la placa o la Raspberry Pi. Mencionar que la corriente máxima que da este regulador es de 1 amperio, que es suficiente para alimentar todos los sensores a 3.3V.

Se ha elegido esta opción de alimentación externa en vez de una alimentación desde la Raspberry Pi para intentar descargarla de cualquier consumo dispensable y que pueda centrar todos sus recursos en la pura computación. Además, con un regulador cuya salida tiene una corriente de 0,5 A, hay potencia más que suficiente para alimentar todos los sensores. Por último, se han usado conectores USB-A por lo comunes que son y por la resistencia mecánica que aportan en comparación con conectores micro USB, que son de montaje superficial (SMD) y tienen una resistencia mucho menor. Estos se

probaron en una iteración previa de la placa y se tuvieron que pegar con cola térmica para evitar que se movieran, complicando su soldado.

Figura 2.9.- Conector USB tipo A

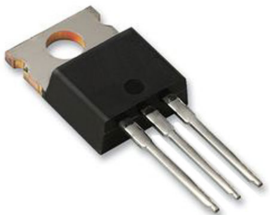

Figura 2.10.- Regulador lineal de 3.3V 7833

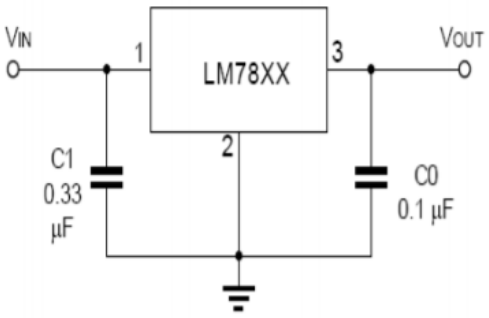

Figura 2.11.- Conexionado regulador lineal 7833

2.2.2.- Esquemático

El circuito esquemático al completo se recoge en la Figura 2.12.

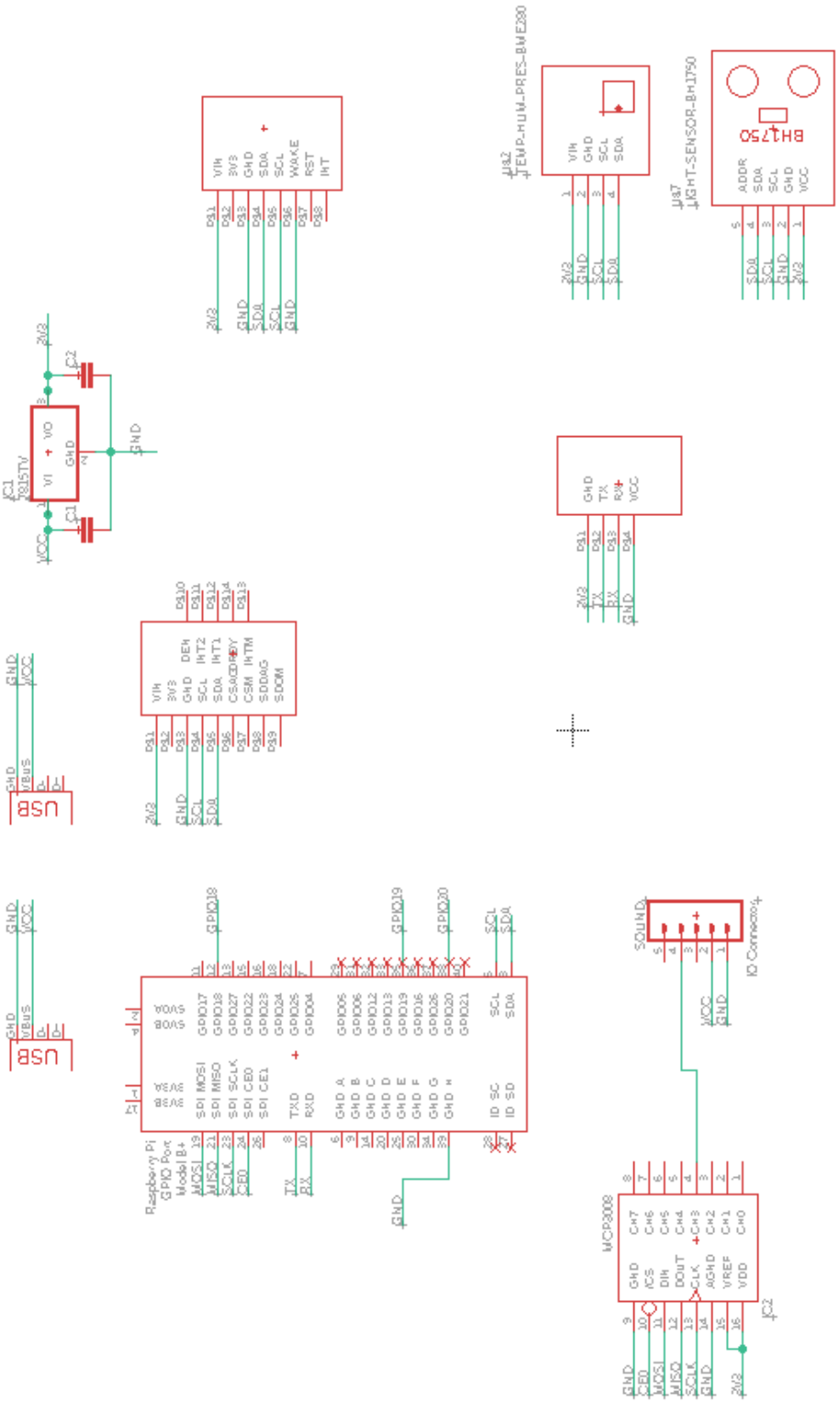

Figura 2.12.- Esquemático general de la placa

La alimentación de todos los sensores se hace a través de la entrada exterior y se divide en 2 circuitos de alimentación: uno de 5V, conectado directamente a la entrada VCC de los USB-A; y otro de 3.3V, conectado a la salida del regulador lineal. El condensador C1 tiene un valor de 0.33 uF y C2, de 0.1 uF. Todos los sensores y el GPS son alimentados a la red de 3.3V excepto el sensor de ruido, que funciona a 5V. Además, la masa de ambas redes está conectada entre sí y a la masa de la Raspberry a través de su pin 39.

Los sensores I2C (sensor de temperatura, humedad y presión, sensor de luz, acelerómetro y sensor de gas) se conectan al pin SDA (pin 3 de la Raspberry) y al pin SCL (puerto 5 de la Raspberry). Además, el sensor de gas tiene una entrada negada, WAKE, que se debe conectar también a GND para que funcione. Podría conectarse a algún pin de la Raspberry para controlar si está funcionando o no. Sin embargo, la intención de este HAT es que recoja todos los datos en todo momento, así que se ha optado por esta opción que simplifica la placa.

El sensor de ruido está conectado al conversor A/D, el MCP3008, a través de su pin *Envelope* al pin número 3 del conversor. El conversor A/D está conectado, por un lado, a la tensión de referencia de la red de 3.3V, y por otro lado a los pines CEO, MOSI, MISO y SCLK de la Raspberry, que son los pines 24, 19, 21 y 23 respectivamente.

Por último, el GPS está conectado a los pines Tx y Rx de la Raspberry, que equivalen a los pines 8 y 10, respectivamente.

2.2.3.- PCB

La PCB se muestra en la Figura 2.13, y su distribución en la Figura 2.14. En rojo se muestran las pistas que irían por la capa *top* y en rojo, las que irían por la capa *bottom*. Además, no se muestra el plano de masa, que iría en la capa *bottom*, y la huella que aparece en el GPS está al revés, es decir, el GPS quedaría hacia la zona de afuera de la placa, al igual que el sensor de ruido. En ambos casos la razón es que, debido al tamaño de los sensores, el ponerlos dentro de la placa aumentaría excesivamente su

tamaño, aumentando a su vez su precio de una manera no necesaria. En el resto de los casos los sensores quedarán dentro de la placa.

Todos los sensores irán soldados a la placa. Además, se necesitará sujetar de alguna forma los conectores USB-A para mantenerlos fijados y evitar posibles roturas debido a la conexión y desconexión de la alimentación. Esta alimentación irá a través de la alimentación de 5V del coche o, en caso de que no se pueda usar, a través de una batería externa de 5V.

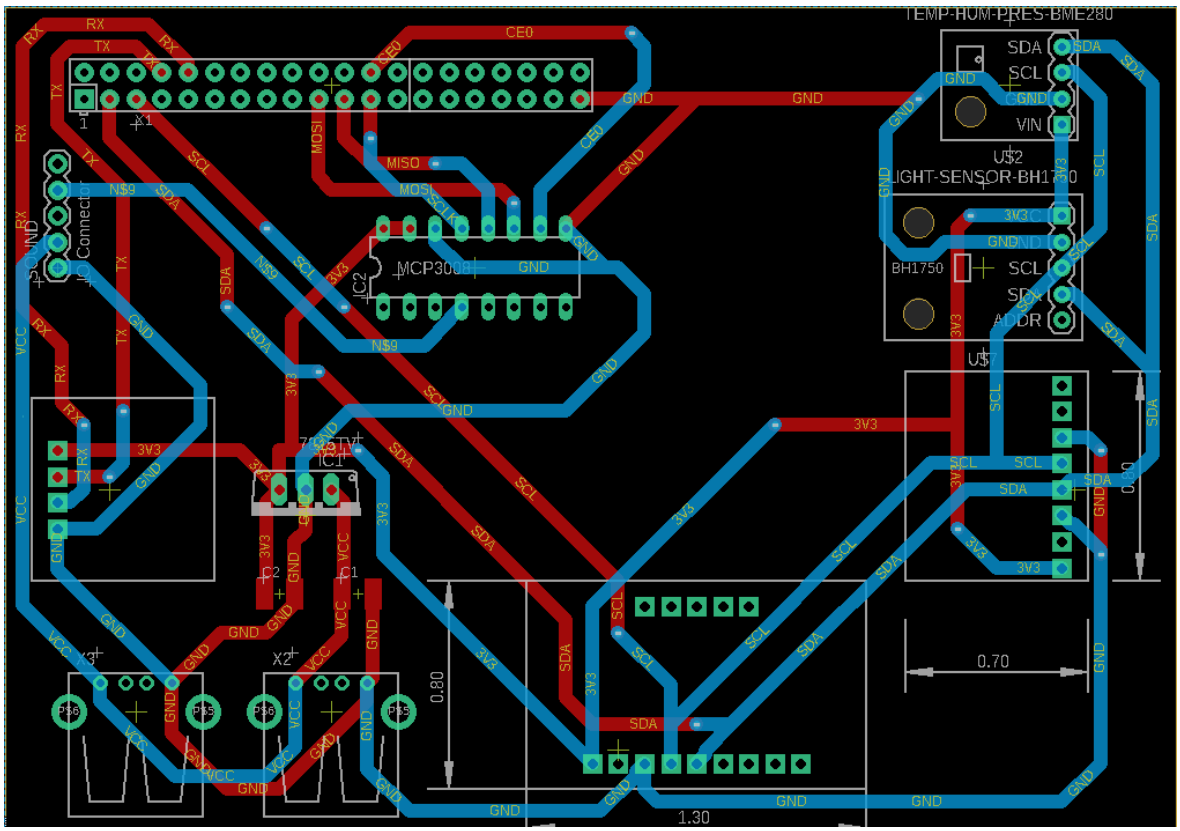

Figura 2.13.- Layout de la PCB

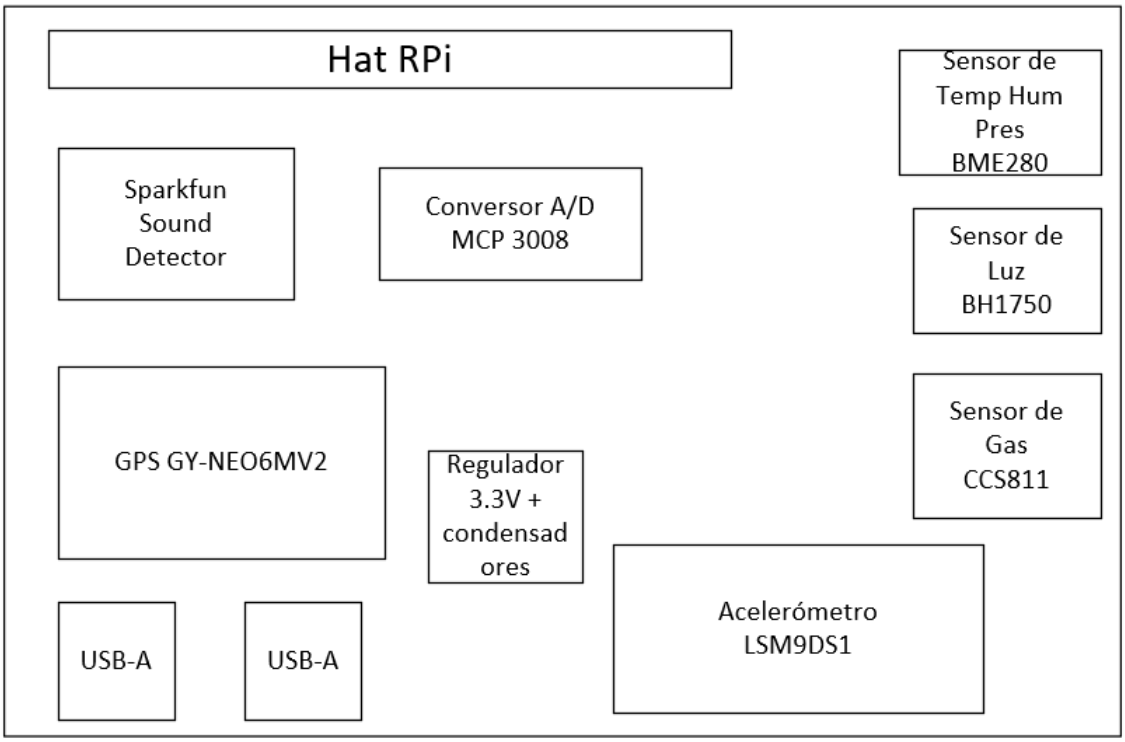

Figura 2.14.- Distribución de la PCB

2.2.4.- Presupuesto por placa completa

En este apartado se recogerá el presupuesto de realización de cada placa, incluyendo el coste de la Raspberry Pi, de la banda H10, del OBD2 y de la caja contenedora:

Descripción	**Precio por unidad (€)**	**Unidades**	**Precio (€)**
Raspberry Pi 4 4GB	49,59	1	49,59
Tarjeta SD 32GB	6,19	1	6,19
PCB	2,56	1	2,56
Conector HAT	2,09	1	2,09
Sensor de ruido SEN12642	18,55	1	18,55
Conversor A/D MCP3008	1.97	1	1.97
GPS GY-NEO6MV2	6,25	1	6,25
Acelerómetro LSM9DS1	19,17	1	19,17
Sensor de calidad de Aire Adafruit CCS811	30,24	1	30,24
Sensor de Luz AZ BH1750	2,31	1	2,31
Sensor de temp, hum y press AZ BME280	4,05	1	4,05
Conector USB-A	0,71	2	1,42
Regulador de tensión lineal 7833	0,69	1	0,69
Condensador 0,33uF	0,46	1	0,46
Condensador 0,11uF	0,09	1	0,09
Polar H10	78,38	1	78,38
Conector OBD2	34,64	1	34,64
Cable USB-C	5.78	1	5,78
Cable USB-A a USB-A	4.94	1	4,94
Caja impresión 3D	4.13	1	4,13
TOTAL SIN IVA			261,82
IVA (21 %)			54,98
TOTAL			316,80

2.3.- Scripts recogida de datos

En esta sección se explicarán los scripts relativos a la recogida de datos de la Raspberry Pi, primeramente mostrando un esquema de cómo será la ejecución de los mismos, y después entrando en detalle del funcionamiento de cada uno por separado. Además, se explicará el funcionamiento del almacenamiento y envío de los mismos.

Mencionar que tanto la base de datos como los scripts de recogida de pulsaciones y RR de la polar h10 y el de envío de datos al servidor ya pertenecían al proyecto, con lo que, aún explicados, no forman parte del trabajo realizado.

2.3.1.- Funcionamiento general

Dentro de este apartado se explicará cómo funciona el sistema de almacenamiento, se realizará un esquema de comunicación entre programa principal y los diferentes scripts y se explicará el programa principal en detalle.

2.3.1.1.- Sistema de almacenamiento

El sistema de almacenamiento local de los datos recogidos por la placa está basado en SQLite. Se ha optado por esta opción puesto que aligera la carga en comparación con un sistema clásico como podría ser MySQL y, en el caso de funcionamiento en el que se encuentra el proyecto, con Raspberries Pi 3 de 1 GigaByte de RAM resultaría más crítico.

Las diferentes tablas siguen la misma disposición:

keyword	unix	datestamp	value
TEXT	REAL	TEXT	TEXT

La columna *keyword* define el sensor de recogida de datos, o, en el caso del acelerómetro, la dirección del espacio en la que se recogen datos; la columna *unix* es la conversión de la columna *datestamp* a unix; y la columna *value*, el valor recogido por el sensor.

Por otro lado, mencionar que existen 2 bases de datos separadas: DATOS_P1, donde se guardan todos los datos referentes a los sensores, la banda de pulsaciones y el GPS; y DATOS_P2, donde se guardan todos los datos recogidos por el OBD.

2.3.1.2.- Esquema de funcionamiento

El funcionamiento general del programa consistirá en un programa principal que lanzará los diferentes scripts de recogida de datos en hilos. Una vez estos scripts recojan datos, los colocarán en la cola correspondiente, y el programa principal los almacenará en la tabla correspondiente, tal y como se muestra en la figura 2.15. Mencionar que, en todos los casos menos en el OBD, la cola la ocuparán solo datos; en caso del OBD, la cola estará formada por una palabra clave que indique qué dato se está pasando y el dato. Esto es debido a la variedad de datos que puede recoger el OBD dependiendo del modelo y año del coche.

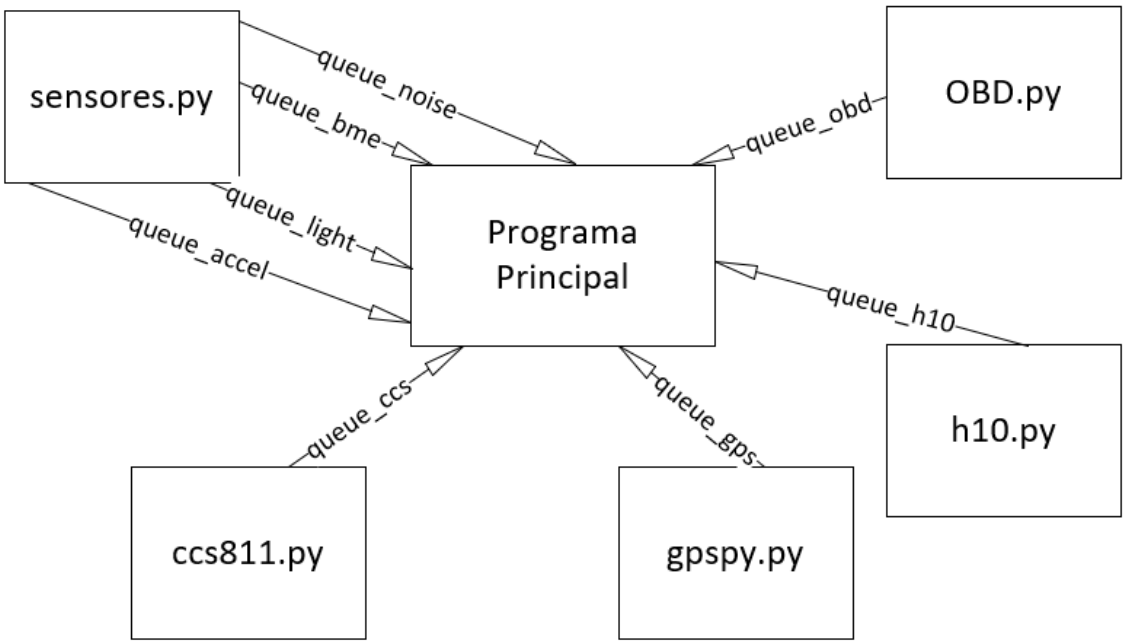

Figura 2.15.- Esquema de funcionamiento

2.3.1.3.- Funcionamiento del programa principal

El programa principal se muestra en el Cuadro 2.1.

```
import threading
```

```
import time
from datetime import datetime
import queue
import sensores
import ccs811
import gpspy
import os
import sqlite3
import hdiez_definitivo
import obd_new

def create_table_P1(curs):
    curs.execute('CREATE TABLE IF NOT EXISTS Temperatura(keyword
        TEXT, unix REAL, datestamp TEXT, value REAL)')
    curs.execute('CREATE TABLE IF NOT EXISTS Humedad(keyword
        TEXT, unix REAL, datestamp TEXT, value REAL)')
    curs.execute('CREATE TABLE IF NOT EXISTS Presion(keyword
        TEXT, unix REAL, datestamp TEXT, value REAL)')
    curs.execute('CREATE TABLE IF NOT EXISTS Luz(keyword TEXT,
        unix REAL, datestamp TEXT, value REAL)')
    curs.execute('CREATE TABLE IF NOT EXISTS Calidad_Aire(
        keyword TEXT, unix REAL, datestamp TEXT, value REAL)')
    curs.execute('CREATE TABLE IF NOT EXISTS Ruido(keyword TEXT,
        unix REAL, datestamp TEXT, value REAL)')
    curs.execute('CREATE TABLE IF NOT EXISTS Acelerometro(
        keyword TEXT, unix REAL, datestamp TEXT, value REAL)')
    curs.execute('CREATE TABLE IF NOT EXISTS Magnetometro(
        keyword TEXT, unix REAL, datestamp TEXT, value REAL)')
    curs.execute('CREATE TABLE IF NOT EXISTS Giroscopio(keyword
        TEXT, unix REAL, datestamp TEXT, value REAL)')
```

```
    curs.execute('CREATE TABLE IF NOT EXISTS LAT_GPS(keyword
        TEXT, unix REAL, datestamp TEXT, value REAL)')
    curs.execute('CREATE TABLE IF NOT EXISTS LONG_GPS(keyword
        TEXT, unix REAL, datestamp TEXT, value REAL)')
    curs.execute('CREATE TABLE IF NOT EXISTS ALTITUD_GPS(keyword
        TEXT, unix REAL, datestamp TEXT, value REAL)')
    curs.execute('CREATE TABLE IF NOT EXISTS SPEED_GPS(keyword
        TEXT, unix REAL, datestamp TEXT, value REAL)')
    curs.execute('CREATE TABLE IF NOT EXISTS Pulso_H10(keyword
        TEXT, unix REAL, datestamp TEXT, value REAL)')
    curs.execute('CREATE TABLE IF NOT EXISTS RR_H10(keyword TEXT
        , unix REAL, datestamp TEXT, value REAL)')

def create_table_P2(curs):
    curs.execute('CREATE TABLE IF NOT EXISTS FUEL_STATUS(keyword
        TEXT, unix REAL, datestamp TEXT, value TEXT)')
    curs.execute('CREATE TABLE IF NOT EXISTS ENGINE_LOAD(keyword
        TEXT, unix REAL, datestamp TEXT, value TEXT)')
    curs.execute('CREATE TABLE IF NOT EXISTS FUEL_PRESSURE(
        keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
    curs.execute('CREATE TABLE IF NOT EXISTS INTAKE_PRESSURE(
        keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
    curs.execute('CREATE TABLE IF NOT EXISTS RPM(keyword TEXT,
        unix REAL, datestamp TEXT, value TEXT)')
    curs.execute('CREATE TABLE IF NOT EXISTS SPEED(keyword TEXT,
        unix REAL, datestamp TEXT, value TEXT)')
    curs.execute('CREATE TABLE IF NOT EXISTS TIMING_ADVANCE(
        keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
    curs.execute('CREATE TABLE IF NOT EXISTS MAF(keyword TEXT,
        unix REAL, datestamp TEXT, value TEXT)')
```

```
curs.execute('CREATE TABLE IF NOT EXISTS THROTTLE_POS(
    keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS RUN_TIME(keyword
    TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS
    FUEL_RAIL_PRESSURE_VAC(keyword TEXT, unix REAL, datestamp
    TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS FUEL_LEVEL(keyword
    TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS ABSOLUTE_LOAD(
    keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS
    RELATIVE_THROTTLE_POS(keyword TEXT, unix REAL, datestamp
    TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS THROTTLE_POS_B(
    keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS THROTTLE_POS_C(
    keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS ACCELERATOR_POS_D(
    keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS ACCELERATOR_POS_E(
    keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS ACCELERATOR_POS_F(
    keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS THROTTLE_ACTUATOR(
    keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS FUEL_INJECT_TIMING(
    keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS FUEL_RATE(keyword
    TEXT, unix REAL, datestamp TEXT, value TEXT)')
```

```
def dynamic_data_entry(sensor, valor, tabla, unix, date, curs,
    conn):
    keyword = sensor
    value = valor
    curs.execute("INSERT INTO " + tabla +" (keyword, unix,
        datestamp, value) VALUES(?, ?, ?, ?)", (keyword, unix,
        date, value))
    conn.commit()


if __name__ == "__main__":
  conn_P1=sqlite3.connect('DATOS_P1.db')
  curs_P1=conn_P1.cursor()
  conn_P2=sqlite3.connect('DATOS_P2.db')
  curs_P2=conn_P2.cursor()
  create_table_P1(curs_P1)
  create_table_P2(curs_P2)
  queue_accel = queue.Queue()
  queue_bme = queue.Queue()
  queue_light = queue.Queue()
  queue_noise = queue.Queue()
  queue_ccs = queue.Queue()
  queue_gps = queue.Queue()
  queue_h10 = queue.Queue()
  queue_rr = queue.Queue()
  queue_obd = queue.Queue()
  try:
    t1 = threading.Thread(target=sensores.main,args=(
        queue_accel,queue_bme,queue_light,queue_noise))
    t2 = threading.Thread(target=ccs811.main,args=(queue_ccs,)
        )
    t3 = threading.Thread(target=gpspy.main,args=(queue_gps,))
```

```
t4 = threading.Thread(target=hdiez_definitivo.main, args=(
    queue_h10, queue_rr))
t5 = threading.Thread(target=obd_new.main, args=(queue_obd
    ,))
t1.setDaemon(True)
t2.setDaemon(True)
t3.setDaemon(True)
t4.setDaemon(True)
t5.setDaemon(True)
t1.start()
t2.start()
t3.start()
t4.start()
t5.start()
while True:
  #print("Main")
  #os.system('clear')
  date = datetime.now()
  unix = date.timestamp()*1000
  date = date.strftime('%Y-%m-%d %H:%M:%S.%f')[:-3]
  #Accel queue
  try:
    accel = queue_accel.get(timeout=0.25)
    #print("Accel")

    if (accel != "" and accel[0]!= None):
      print("Accelerometer: ", accel[0:3])
      print("Magnetometer: ", accel[3:6])
      print("Giroscope: ", accel[6:8])
      dynamic_data_entry('X', accel[0], "Acelerometro", unix
          , date, curs_P1, conn_P1)
```

```
            dynamic_data_entry('Y', accel[1],"Acelerometro",unix
                ,date,curs_P1,conn_P1)
            dynamic_data_entry('Z', accel[2],"Acelerometro",unix
                ,date,curs_P1,conn_P1)
            dynamic_data_entry('X', accel[3],"Magnetometro",unix
                ,date,curs_P1,conn_P1)
            dynamic_data_entry('Y', accel[4],"Magnetometro",unix
                ,date,curs_P1,conn_P1)
            dynamic_data_entry('Z', accel[5],"Magnetometro",unix
                ,date,curs_P1,conn_P1)
            dynamic_data_entry('X', accel[6],"Giroscopio",unix,
                date,curs_P1,conn_P1)
            dynamic_data_entry('Y', accel[7],"Giroscopio",unix,
                date,curs_P1,conn_P1)
            dynamic_data_entry('Z', accel[8],"Giroscopio",unix,
                date,curs_P1,conn_P1)
            queue_accel.queue.clear()
        except Exception as e:
            print(e)
        #BME queue
        try:
            bme = queue_bme.get(timeout=0.25)
            #print("BME")
            if (bme !=""):
                print("Temperature: ",bme[0] )
                print("Humidity: ",bme[1])
                print("Pressure: ",bme[2])
                if (bme[0]<100 and bme[0]>-20):
                    dynamic_data_entry('Temperatura (Grados C)', bme
                        [0],"Temperatura",unix,date,curs_P1,conn_P1)
                if (bme[2]<100 and bme[2]>0):
```

```
            dynamic_data_entry('Humedad (%)', bme[2],"Humedad"
                ,unix,date,curs_P1,conn_P1)
        if (bme[1]<2000 and bme[1]>0):
            dynamic_data_entry('Presion (hPa)', bme[1],"
                Presion",unix,date,curs_P1,conn_P1)
        queue_bme.queue.clear()
    except Exception as e:
        print(e)
    #Light queue
    try:
        light = queue_light.get(timeout=0.25)
        #print("Light")
        if (light!="" and light <10000):
            print("Light: ",light)
            #print(type(light))
            dynamic_data_entry('Nivel de luz (lux)', light,"Luz"
                ,unix,date,curs_P1,conn_P1)
            queue_light.queue.clear()
    except Exception as e:
        print(e)
    #Noise queue
    try:
        noise = queue_noise.get(timeout=0.25)
        if (noise !=""):
            print("Noise",noise)
            #print(noise)
            dynamic_data_entry('Nivel de ruido (dB)',noise,"
                Ruido",unix,date,curs_P1,conn_P1)
            queue_noise.queue.clear()
    except Exception as e:
        print(e)
```

```
#CCS queue
try:
    ccs = queue_ccs.get(timeout=0.25)
    #print("CCS")
    if(ccs[0]<1000 and ccs[0] > 0):
        print("CO2: ",ccs)
        dynamic_data_entry('Calidad aire (ppm)',ccs[0],"
            Calidad_aire",unix,date,curs_P1,conn_P1)
        #dynamic_data_entry('TVOC (ppb)',ccs[1],"
            Calidad_aire",unix,date)
    queue_ccs.queue.clear()
except Exception as e:
    #t2.kill()
    #t2.start()
    print(e)
#GPS queue
try:
    gps = queue_gps.get(timeout=0.25)
    #print("GPS")
    if(gps[0]!=0 and gps[0] != None):
        print("Lat: ",gps[0])
        print("Lon: ",gps[1])
        print("Alt: ",gps[2])
        print("Speed: ",gps[3])
        dynamic_data_entry('Latitud modulo GPS',gps[0], "
            LAT_GPS", unix, date,curs_P1,conn_P1)
        dynamic_data_entry('Longitud modulo GPS',gps[1], "
            LONG_GPS", unix, date,curs_P1,conn_P1)
        dynamic_data_entry('Altitud modulo GPS',gps[2], "
            ALTITUD_GPS", unix, date,curs_P1,conn_P1)
```

```
            dynamic_data_entry('Velocidad modulo GPS',gps[3], "
                SPEED_GPS", unix, date,curs_P1,conn_P1)
            queue_gps.queue.clear()
        except Exception as e:
            print(e)
        try:
            h10 = queue_h10.get(timeout=0.25)
            if(h10!=None and h10>0):
                print("Pulso: ",h10)
                dynamic_data_entry('Ritmo cardiaco (ppm)',h10,"
                    Pulso_H10", unix, date,curs_P1,conn_P1)
        except Exception as e:
            print(e)
        try:
            rr = queue_rr.get(timeout=0.25)
            if(rr!=None and rr>0):
                #print("-------------------------------------------------")
                print("RR: ",rr)
                #print("-------------------------------------------------")
                dynamic_data_entry('Intervalo RR',rr,"RR_H10",unix,
                    date,curs_P1,conn_P1)
            queue_h10.queue.clear()
        except Exception as e:
            print(e)
        try:
            obd = queue_obd.get(timeout=0.25) #lista_buena[i],str(
                res.value)
            for i in range (0,round(len(obd)/2)):
                print("-------------------------------------------------")
                print(obd[i])
                print(obd[i+1])
```

```
            print("-----------------------------------------------------")
            dynamic_data_entry(obd[i],obd[i+1],obd[i],unix,date,
                curs_P2,conn_P2)
        except Exception as e:
            print(e)
        print(date)
        time.sleep(0.5)
    except KeyboardInterrupt:
        print("Stop")
```

Cuadro 2.1.- Script main.py

El funcionamiento del programa se basa en hilos y colas. Primeramente, se crean las diferentes colas mencionadas en la Figura 2.15. Una vez creadas, se lanzan los hilos con las colas como argumentos.

Además, presenta 2 funciones: la primera, **create_table**, que sirve para crear, en caso de que no existan, todas las tablas en la base de datos sqlite3 DATOS_P1; y la segunda, **dynamic_data_entry**, que sirve para guardar todos los datos en su tabla correspondiente.

En el bucle infinito se comprueban los datos cada medio segundo. Se supone que los datos recogidos se cogen en el mismo *timestamp*, siendo éste como mucho de 0.25 segundos. Una vez recuperados los datos de las colas, se llama a la función **dynamic_data_entry** y se guardan en su correspondiente tabla.

Como comentario, se asegura que los datos sean reales, por ejemplo, se comprueba que la calidad de aire no sea mayor de 30000 o de 0.

2.3.2.- Explicación detallada de cada script

En esta sección se explicará en detalle los scripts desarrollados durante el período de prácticas y, de manera más general, los scripts que se están usando pero no se han desarrollado.

2.3.2.1.- Script sensores.py

El script *sensores.py* se muestra en la Cuadro 2.2.

```
import time
import RPi.GPIO as GPIO
import datetime
import sys
import os
import bme280
import lum
import busio
import sqlite3
import adafruit_mcp3xxx.mcp3008 as MCP
from adafruit_mcp3xxx.analog_in import AnalogIn
import digitalio
import board
import adafruit_lsm9ds1
import numpy as np
import queue
from board import *

# create the spi bus
spi = busio.SPI(clock=board.SCK, MISO=board.MISO, MOSI=board.
    MOSI)

# create the cs (chip select)
cs = digitalio.DigitalInOut(board.D22)

# create the mcp object
mcp = MCP.MCP3008(spi, cs)
```

```
chan0 = AnalogIn(mcp, MCP.P3)

def getNoise():
  try:
    divValue = chan0.voltage
    #print(divValue)
    time.sleep(0.1)
    noise = 41.8518 * np.log(divValue) + 124.7257
    #print("Max: "+ str(noise))
    return noise
  except Exception as e:
    print (e)
    return None



i2c_bus = busio.I2C(SCL, SDA)
#(chip_id, chip_version) = bme280.readBME280ID()

def main(queue_accel=None,queue_bme=None,queue_light=None,
    queue_noise=None):
#conn = sqlite3.connect('/home/pi/DATOS_P1.db')
  i2c = busio.I2C(board.SCL, board.SDA)
  sensor = adafruit_lsm9ds1.LSM9DS1_I2C(i2c)
  try:
    while(True):
      try:
        accel_x, accel_y, accel_z = sensor.acceleration
        mag_x, mag_y, mag_z = sensor.magnetic
        gyro_x, gyro_y, gyro_z = sensor.gyro
        #temp = sensor.temperature
        # Print values.
```

```
        '''print('Acceleration (m/s^2): ({0:0.3f},{1:0.3f
            },{2:0.3f})'.format(accel_x, accel_y, accel_z))
        print('Magnetometer (gauss): ({0:0.3f},{1:0.3f},{2:0.3
            f})'.format(mag_x, mag_y, mag_z))
        print('Gyroscope (degrees/sec): ({0:0.3f},{1:0.3f
            },{2:0.3f})'.format(gyro_x, gyro_y, gyro_z))'''
        queue_accel.queue.clear()
        queue_accel.put((accel_x,accel_y,accel_z,mag_x, mag_y,
             mag_z,gyro_x, gyro_y, gyro_z))
        #map(queue_accel.put(),[accel_x,accel_y,accel_z,mag_x,
             mag_y, mag_z,gyro_x, gyro_y, gyro_z])
    except Exception as e:
        print(e)
    try:
        temperature,pressure,humidity = bme280.readBME280All()
        '''print ("Temperature : ", temperature, "C")
        print ("Pressure : ", pressure, "hPa")
        print ("Humidity : ", humidity, "%")
        dynamic_data_entry('Temperatura (Grados C)',
            temperature,"Temperatura",unix,date)
        dynamic_data_entry('Humedad (%)', humidity,"Humedad",
            unix,date)
        dynamic_data_entry('Presion (hPa)', pressure,"Presion
            ",unix,date)'''
        queue_bme.queue.clear()
        queue_bme.put((temperature,pressure,humidity))
        #list(map(queue_bme.put(),[temperature,pressure,
            humidity]))
    except Exception as e:
        print(e)
```

```
        #En cuanto tengas esto enchufado y funcionando hablamos
            de como van las URLs
        try:
            light=lum.readLight()
            #print("Light Level : " , light , " lx")
            queue_light.queue.clear()
            queue_light.put(light)
            #dynamic_data_entry('Nivel de luz (lux)', light,"Luz",
                unix,date)
        except Exception as e:
            print(e)
        try:
            noise = getNoise()
            #print("Nivel de ruido (dB): ", noise )
            queue_noise.queue.clear()
            queue_noise.put(noise)
        except Exception as e:
            print(e)
        time.sleep(1)
        #os.system('clear')
    except KeyboardInterrupt:
        print("Finalizando")
    except Exception as e:
        print(e)
    finally:
        GPIO.cleanup()

if __name__ == "__main__":
    queue_accel = queue.Queue()
    queue_bme = queue.Queue()
    queue_light = queue.Queue()
```

```
queue_noise = queue.Queue()
main(queue_accel,queue_bme,queue_light,queue_noise)
```

Cuadro 2.2.- Script sensores.py

El script tiene 1 función, **getNoise**, que primeramente obtiene el valor del ruido en dB del conversor A/D al que se le aplica una fórmula matemática extrapolada de la calibración del sensor y devuelve este valor.

El programa principal es un bucle infinito, con un *try...except* general para evitar que, aunque se produzca un fallo no controlado, no pare de ejecutarse. Dentro de éste se encuentran 4 *try...except*, el primero para el acelerómetro; el segundo, para el sensor BME280; el tercero para el sensor de luz; y el cuarto para el sensor de ruido. El funcionamiento de todos es parecido, primeramente pidiendo los valores a sus librerías (caso de BME, acelerómetro y luz) o a la función getNoise (caso sensor de ruido) y, posteriormente añadiéndolos a su correspondiente cola para enviárselo al programa principal. Además, se encuentran *prints* como forma de depuración para comprobar el correcto funcionamiento de los sensores.

2.3.2.2.- Script ccs811.py

El script *ccs811.py* se muestra en la Cuadro 2.3.

```
import time
import board
import busio
import adafruit_ccs811
import queue

def main(queue_ccs=None):
  while True:
    try:
```

```
        i2c = busio.I2C(board.SCL, board.SDA)
        ccs811 = adafruit_ccs811.CCS811(i2c)
        break
    except Exception as e:
        print(e)
        time.sleep(0.1)
# Wait for the sensor to be ready
while not ccs811.data_ready:
    pass

print("Sensor ready")

while True:
    #print("CO2: {} PPM, TVOC: {} PPB".format(ccs811.eco2,
        ccs811.tvoc))
    try:
        queue_ccs.queue.clear()
        queue_ccs.put((ccs811.eco2,ccs811.tvoc))
        #map(queue_ccs.put(),[ccs811.eco8,ccs811.tvoc])
        time.sleep(1)
    except Exception as e:
        print(e)
        time.sleep(1)

if __name__ == "__main__":
    queue = queue.Queue()
    main(queue)
```

Cuadro 2.3.- Script ccs811.py

El script de recogida de datos de gas es sencillo y se apoya en la librería de Adafruit. Como los scripts de los sensores, recoge los datos de la función de la librería y los envía a una cola para que se almacenen en la base de datos a través del programa principal.

2.3.2.3.- Script gpspy.py

El script *gpspy.py* se muestra en la Cuadro 2.4.

```
#! /usr/bin/python3
# Written by Dan Mandle http://dan.mandle.me September 2012
# License: GPL 2.0

import os
from gps import *
from time import *
import time
import threading
import requests
import json
import datetime
import queue

gpsd = None #seting the global variable

os.system('clear') #clear the terminal (optional)

class GpsPoller(threading.Thread):
  def __init__(self):
    threading.Thread.__init__(self)
    global gpsd #bring it in scope
    gpsd = gps(mode=1) #starting the stream of info
```

```
    self.current_value = None
    self.running = True #setting the thread running to true

  def run(self):
    global gpsd
    while gpsp.running:
      gpsd.next() #this will continue to loop and grab EACH
          set of gpsd info to clear the buffer

def main(queue_gps=None):
  global gpsp
  gpsp = GpsPoller() # create the thread
  try:
    gpsp.start() # start it up
    while True:
      #It may take a second or two to get good data
      #print gpsd.fix.latitude,', ',gpsd.fix.longitude,'  Time
          : ',gpsd.utc
      queue_gps.queue.clear()
      #os.system('clear')
      unix=time.time()
      lat = gpsd.fix.latitude
      lon = gpsd.fix.longitude
      alt = gpsd.fix.altitude
      speed = gpsd.fix.speed
      #print(gpsd.fix)
      timeGPS = gpsd.utc,' + ', gpsd.fix.time
      '''print("Lat: ",lat)
      print("lon: ",lon)
      print("Alt: ", alt)
      print("Speed: ",speed)'''
```

```
        queue_gps.put((lat,lon,alt,speed))
        #map(queue_gps.put(),[lat,lon,alt,speed])
        #print(str(datetime.datetime.fromtimestamp(unix).
            strftime('%Y-%m-%d %H:%M:%S')))
        time.sleep(1) #set to whatever

    except (KeyboardInterrupt, SystemExit): #when you press ctrl
        +c
        print ("\nKilling Thread...")
        gpsp.running = False
        gpsp.join() # wait for the thread to finish what it's
            doing
    print ("Done.\nExiting.")

if __name__ == '__main__':
    queue_gps = queue.Queue()
    main(queue_gps)
```

Cuadro 2.4.- Script gpspy.py

En el caso del GPS, el funcionamiento está basado en un programa ya hecho que crea la clase **GpsPoller**. Una vez creado este objeto, en el bucle infinito se piden los datos de latitud, longitud, altitud y velocidad para colocarlo en la cola correspondiente. El programa principal será el encargado de almacenarlos.

2.3.2.4.- Script obd.py

El script *obd.py* se muestra en la Cuadro 2.5.

```
import obd
import time
import sqlite3
```

```
import sys
import datetime
import subprocess
import os
import queue

def main(queue):
    #create_table(curs)
    #print(con.status())
    lista_buena=['FUEL_STATUS', 'ENGINE_LOAD', 'FUEL_PRESSURE', '
        INTAKE_PRESSURE', 'RPM', 'SPEED', 'TIMING_ADVANCE', 'MAF', '
        THROTTLE_POS', 'RUN_TIME', 'FUEL_RAIL_PRESSURE_VAC', '
        FUEL_LEVEL', 'ABSOLUTE_LOAD', 'RELATIVE_THROTTLE_POS', '
        THROTTLE_POS_B', 'THROTTLE_POS_C', 'ACCELERATOR_POS_D', '
        ACCELERATOR_POS_E', 'ACCELERATOR_POS_F', 'THROTTLE_ACTUATOR
        ', 'FUEL_INJECT_TIMING', 'FUEL_RATE']
    cmd_list = [obd.commands.FUEL_STATUS, obd.commands.
        ENGINE_LOAD, obd.commands.FUEL_PRESSURE, obd.commands.
        INTAKE_PRESSURE, obd.commands.RPM, obd.commands.SPEED, obd.
        commands.TIMING_ADVANCE, obd.commands.MAF, obd.commands.
        THROTTLE_POS, obd.commands.RUN_TIME, obd.commands.
        FUEL_RAIL_PRESSURE_VAC, obd.commands.FUEL_LEVEL, obd.
        commands.ABSOLUTE_LOAD, obd.commands.RELATIVE_THROTTLE_POS
        , obd.commands.THROTTLE_POS_B, obd.commands.THROTTLE_POS_C,
        obd.commands.ACCELERATOR_POS_D, obd.commands.
        ACCELERATOR_POS_E, obd.commands.ACCELERATOR_POS_F, obd.
        commands.THROTTLE_ACTUATOR, obd.commands.
        FUEL_INJECT_TIMING, obd.commands.FUEL_RATE]
    while True:
        try:
            con = obd.OBD()
```

```
i = 0
unix=time.time()
date=str(datetime.datetime.fromtimestamp(unix).
    strftime('%Y-%m-%d %H:%M:%S'))
for cmd in cmd_list:
  try:
    #print(str(cmd))
    res=con.query(cmd)
    resval=str(res.value)
    #print(resval)
    if (resval!="None"):
      #print("VALOR")
      #dynamic_data_entry(lista_buena[i],str(res.value
          ),lista_buena[i],unix,date,curs)
      queue.put((lista_buena[i],str(res.value)))
    else:
      print(con.status())
      if(con.status()!=obd.OBDStatus.CAR_CONNECTED):
        con.close()
        con = obd.OBD()
      #res_ful[contador]=0
  except Exception as e:
    print(e)
    time.sleep(1)
    #con = connectCar()
    #log.logSensors("OBD", e)
  #time.sleep(1)

  i = i + 1
print(date)
con.close()
```

```
        #time.sleep(1)
    except Exception as e:
        print(e)
        #log.logSensors("OBD", e)
        time.sleep(1)
        if(con.status!=obd.OBDStatus.CAR_CONNECTED):
          time.sleep(1)
          con = obd.OBD()
        '''try:
            subprocess.call(["sudo rm /home/pi/DATOS_P2.db-
                journal"], shell=True)
            subprocess.call(["sudo cp /home/pi/DATOS_P42.db
                /home/pi/DATOS_P2.db"], shell=True)
        except:
            pass
        pass'''


  con.close()

if __name__ == "__main__":
  main()
```

Cuadro 2.5.- Script obd.py

La conexión con el OBD se realiza a través del USB con el uso del constructor de la librería OBD. Una vez conectado, se realizan peticiones usando la librería obd al dispositivo conectado con el bus CAN y devuelve valores de velocidad, RPM, ... Estos datos se añaden a la cola correspondiente de la forma (Nombre de la variable, Valor de la variable) y el programa principal los almacena en la base de datos.

2.3.2.5.- Script h10.py

El script *h10.py* se muestra en la Cuadro 2.6.

```
import time
import pexpect
import subprocess
import queue

def hexStrToInt(hexstr):
    val = int(hexstr[0:2],16) + (int(hexstr[3:5],16)<<8)
    if ((val&0x8000)==0x8000): # treat signed 16bits
        val = -((val^0xffff)+1)
    return val

def main(queue_h10=None, queue_rr=None):
  f=open("mac.conf","r")
  mac=f.read()
  f.close()
  #print("Running gatttool...")
  command="sudo gatttool -t random -b "+mac+" --char-write-req
      --handle=0x0011 --value=0100 --listen"
  #print(command)
  child=pexpect.spawn(command)
  tries=0
  while True:
    try:
      queue_h10.queue.clear()
      child.expect("Notification handle = 0x0010 value: ",
          timeout=10)
      child.expect("\r\n", timeout=10)
```

```
        pulso=int(child.before[2:5],16)
        #print("Pulso:  ",pulso)
        queue_h10.put(pulso)
        if(len(child.before)>15):
            rr=hexStrToInt(child.before[12:17])
            print("RR: "+str(rr))
            queue_rr.put(rr)
        time.sleep(0.3)
        first=False

    except:
        print("Failed!")
        time.sleep(0.3)
        subprocess.call("sudo rfkill block bluetooth",shell=True
            )
        time.sleep(0.1)
        subprocess.call("sudo rfkill unblock bluetooth",shell=
            True)
        time.sleep(0.1)
        child=pexpect.spawn(command)
        tries=tries+1

if __name__ == "__main__":
  queue = queue.Queue()
  main(queue)
```

Cuadro 2.6.- Script h10.py

En el caso del script de recogida de datos de pulso y RR de la polar h10, el funcionamiento es algo diferente. Primeramente, se realiza la conexión de la polar por bluetooth con su dirección MAC, guardada en un archivo de configuración. Posteriormente, se obtienen los datos del pulso y del RR a través de los datos

hexadecimales devueltos por la polar, y se añaden a su cola correspondiente, que serán almacenados por el programa principal.

2.3.2.6.- Script send_database.py

El script *send_database.py* se muestra en la Cuadro 2.7.

```
import sqlite3 as db
import logging
import requests
import csv
import sys
import gzip
import zlib
import yaml
import json
import time
import datetime
import io



logging.basicConfig(filename='envio.log', level=logging.DEBUG)



N = 20
MAP= {
    "ALTITUD_GPS": "gps_alt",
    "LAT_GPS": "gps_lat",
    "LONG_GPS": "gps_lon",
    "Humedad": "humidity",
    "Calidad_Aire": "air_quality",
```

```
    "Luz": "light",
    "Pulso_H10": "hr",
    "RR_H10": "rr",
    "Time_GPS": "gps_time",
    "Temperatura": "temperature",
    "Ruido": "noise",
    "Acelerometro": "acelerometer"
}


def create_table(database):
    try:
        with db.connect(database) as conn:
            cur = conn.cursor()
            sql = 'CREATE TABLE IF NOT EXISTS LASTID(tablename
                TEXT PRIMARY KEY, lastIdSent INT, lastIdRcvd
                INT)'
            cur.execute(sql)
        return True
    except db.Error as e:
        logging.error(e)
        return False


def update_table(database, table, lastIdSent,lastIdRcvd):
    try:
        with db.connect(database) as conn:
            cur = conn.cursor()
            cur.execute("INSERT OR REPLACE INTO LASTID (
                tablename,lastIdSent, lastIdRcvd) VALUES(?, ?,
                ?)",(table, lastIdSent, lastIdRcvd))
```

```
            conn.commit()
        return True
    except db.Error as e:
        logging.error(e)
        return False


def get_lastId(database,table):
    try:
        with db.connect(database) as conn:
            cur = conn.cursor()
            cur.execute("SELECT lastIdRcvd FROM LASTID WHERE
                tablename=(?);",(table,))
            lastIdRcvd=tables = cur.fetchone()
        if lastIdRcvd is None:
            lastIdRcvd=1
        else:
            lastIdRcvd=lastIdRcvd[0];
        return lastIdRcvd
    except db.Error as e:
        logging.error(e)

def import_yaml(file):
    """Function to read de Config from a YAML file"""
    with open(file) as f:
        var = yaml.load(f)
        return var['Config']


def ReadDatabaseTables(database):
    """Read the names of the tables in a database"""
```

```
    try:
        with db.connect(database) as conn:
            cur = conn.cursor()
            sql = "SELECT name FROM sqlite_master WHERE type='
               table' ORDER BY name;"
            cur.execute(sql)
            tables = cur.fetchall() #Returns the data as a
               tuple
            tablesList = []
            data = [0]
            for table in tables:
                tablesList.append(''.join(table)) #Tuple to
                   list of strings
            tablesList.remove('LASTID')
    except db.Error as e:
        logging.error(e)
        return False
    except Exception as e:
        logging.error(e)
        return False
    else:
        return tablesList


def remove_sended_records(database, tableName, n):
    """Remove the N first rows of a table and reset the rowid
       for further iterations"""
    try:
        with db.connect(database) as conn:
            cur = conn.cursor()
            sql = 'delete from ' + tableName + \
```

```
                '␣where␣rowid␣IN␣(SELECT␣rowid␣from␣'+
                    tableName + \
                '␣order␣by␣rowid␣asc␣limit␣' + str(N)+')'
            print(sql)
            cur.execute(sql)
            conn.commit() #Save the changes
            sql = 'vacuum' #Reset the rowId
            cur.execute(sql)
    except db.Error as e:
        return False
    else:
        return True


def query_db(database, tableName, lastIdRcvd,n, username, one=
    False):
    """Get the N first rows of a table and format a JSON
       object to send it to the server"""
    try:
        with db.connect(database) as conn:
            cur = conn.cursor()
            sql = "SELECT␣*␣FROM␣"+tableName + \
                "␣WHERE␣("+"rowid>"+str(lastIdRcvd)+"␣and␣
                    rowid<"+str(lastIdRcvd+N+1)+")"
            cur.execute(sql)
            r = [dict((cur.description[i][0], value)
                      for i, value in enumerate(row)) for row
                         in cur.fetchall()] #Create a dict
                         with the columns of the table as keys
                          and the values as, well, values
            dictionary = []
```

```
            for j in range(0, len(r)):
                r[j]["username"] = username
                if(database == 'DATOS_P2.db'):
                    r[j]["type"] = tableName.lower() #Modify
                        the name of the tables to match the
                        names present in the server DB
                else:
                    r[j]["type"] = MAP[tableName]
                r[j]["ts"] = r[j].pop('datestamp')
                dictionary.append({"h2data": r[j]})
    except db.Error as e:
        print('Error en la BBDD')
        sys.exit()
    else:
        return dictionary


def login(host, username, mac):
    """Check if the device is registered in the server
       database"""
    try:
        dict = {"username": username, "idDevice": str(mac)}
        requests_body = json.dumps(dict)
        method = host+'login'
        print('Login: ' + method)
        #print(requests_body)
        headers = {'Content-type': 'application/json'}
        response = requests.post(method, data=requests_body,
           headers=headers)
        print(response)
```

```
        while response.status_code != requests.codes.ok: #
            Retries forever until 200OK
            response = requests.post(method, data=
                requests_body, headers=headers)
            print('Retrying')
        #user = response.json()["username"]
        logging.info(response)
        #print(user)
        return username
    except Exception as e:
        print(e)
        time.sleep(5)
        return login(host, username, mac)


def send_process(host, data,lastIdRcvd,N,database,table):
    """Send data to the server"""
    try:
        requests_body = data
        method = host+'postData'
        headers = {'Content-type': 'application/json', '
            Content-Encoding': 'gzip'}
        response = requests.post(method, data=requests_body,
            headers=headers)
        while response.status_code != requests.codes.ok:
            response = requests.post(method, data=
                requests_body, headers=headers)
        print(response)
        update_table(database,table,lastIdRcvd,lastIdRcvd+N)
        return True
    except Exception:
```

```
        time.sleep(5)
        print("Error. Retrying.")
        return send_process(host, data,lastIdRcvd,N,database)


def main(argv):
    time.sleep(10)
    config = import_yaml('config.yaml')
    rPiMac = config['ID']
    user = config['User']
    db1 = 'DATOS_P1.db'
    db2 = 'DATOS_P2.db'
    create_table(db1)
    create_table(db2)
    tables1 = ReadDatabaseTables(db1)
    tables2 = ReadDatabaseTables(db2)
    host = 'http://156.35.171.153/api/'
    username = login(host, user, rPiMac)
    k = 0
    while True:
        try:
            for table in tables1:
                print(table)
                lastIdRcvd=get_lastId(db1,table)
                print(lastIdRcvd)
                list = query_db('DATOS_P1.db', table,
                    lastIdRcvd,N, username,db1)
                json_output = json.dumps(list, indent=1)
                print(len(list))
                if(len(list) == N):
                    print("sendPROCESS")
```

```
                    send_process(host, json_output,lastIdRcvd,
                        N,db1,table)
                    print("OUT")
                    k = 0
                else:
                    k = k+1
                if(k == 12): #If DATOS_P1.db doesnt have
                    enough data, wait 2 minutes. DATOS_P1.db
                    always has more data than DATOS_P2.db
                    k = 0
                    print('Alakazam uso hipnosis!')
                    time.sleep(120)
                    print()

                time.sleep(1)
            for table in tables2:
                lastIdRcvd=get_lastId(db2,table)
                list = query_db('DATOS_P2.db', table,
                    lastIdRcvd, N, username,db2)
                json_output = json.dumps(list, indent=1)
                if(len(list) == N):
                    send_process(host, json_output,lastIdRcvd,
                        N,db2,table)
        except Exception as e:
            print(e)


if __name__ == "__main__":
    try:
        main(sys.argv[:1])
    except KeyboardInterrupt as e:
```

```
        print(e)
```

Cuadro 2.7.- Script send_database.py

El script send_database lee ambas bases de datos, DATOS_P1 y DATOS_P2 y las envía al servidor remoto donde se almacenan. En primer lugar, realiza una conexión con el servidor donde envía el nombre de la placa y el identificador asignado a la misma y, una vez reciba una respuesta satisfactoria del servidor, envía los datos en tuplas de 20. En el servidor se almacenan estos datos junto con el nombre de usuario de la placa.

2.3.3.- Script tabla_sinc

El script *tabla_sinc* sirve para juntar todos los datos en una única tabla. Los datos se almacenan en diferentes tablas y, al enviarlos al servidor, se mantiene esta distribución. Además, existen datos antiguos de una otra versión de la placa cuyos valores a veces no están sincronizados. Por estas razones, se crea el siguiente script que junta los datos según la marca de tiempo.

Cabe mencionar varias cosas de este script:

- La marca de tiempo se coge de la tabla latitud. Esto es debido a que, en la anterior versión de la placa, la raspberry carecía de conexión a Internet, con lo que obtenía la hora del GPS. Es decir, cuando no tenía GPS, la hora de la raspberry no era correcta. Si se cogen los datos de la tabla latitud, se asegura que la hora sea correcta.

- Primeramente se intenta coger los datos comprendidos entre 1 segundo anterior y posterior de la marca de tiempo sacada de la tabla latitud. Sin embargo, si no existieran datos para ese rango, se cogería el anterior valor sacado de la tabla correspondiente, siempre y cuando no hubiesen pasado más de 100 segundos.

- El unix del tiempo se guarda en formato VARCHAR debido a que si se guarda en formato REAL, sql simplifica el número a notación científica, perdiendo mucha exactitud.

- De no existir ningún dato que se cumpla las dos restricciones anteriores, se guardará -9999, para indicar que este valor no es viable.

El script se recoge en el Cuadro 2.8.

```
import logging
import requests
import csv
import sys
import gzip
import zlib
import yaml
import json
import time
import io
import numpy as np
from datetime import datetime, timedelta
import pyodbc

username="xandru"

try:
  time_unix = query_db_time()
except:
  time_unix = datetime.utcfromtimestamp(1000202290).strftime('
      %Y-%m-%d %H:%M:%S')

def create_table():
  try:
```

```
    with pyodbc.connect('Driver={ODBC Driver 17 for SQL Server
        };''Server=localhost;''Database=SMIOT;''
        Trusted_Connection=yes;') as conn:
      cur = conn.cursor()
      sql = 'CREATE TABLE SMIOT.dbo.data_all(unix VARCHAR(150)
          , datestamp TEXT, username VARCHAR(150), 
          absolute_load VARCHAR(150), accelerator_pos_d VARCHAR
          (150), accelerator_pos_e VARCHAR(150), 
          accelerator_pos_f VARCHAR(150), acelerometer_x 
          VARCHAR(150), acelerometer_y VARCHAR(150), 
          acelerometer_z VARCHAR(150), air_quality VARCHAR(150)
          , engine_load VARCHAR(150), fuel_rate VARCHAR(150), 
          gps_alt VARCHAR(150), gps_lat VARCHAR(150), gps_lon 
          VARCHAR(150), gps_time VARCHAR(150), hr REAL, 
          humidity VARCHAR(150), intake_pressure VARCHAR(150), 
          light VARCHAR(150), maf VARCHAR(150), noise VARCHAR
          (150), relative_accel_pos VARCHAR(150), 
          relative_throttle_pos VARCHAR(150), rpm VARCHAR(150), 
          rr VARCHAR(150), run_time VARCHAR(150), speed 
          VARCHAR(150), temperature VARCHAR(150), throttle_pos 
          VARCHAR(150), throttle_pos_b VARCHAR(150), 
          throttle_pos_c VARCHAR(150))'
      cur.execute(sql)
    return True
  except Exception as e:
    logging.error(e)
    return False

def update_table(unix, datestamp, values):
  try:
    global username
```

```
    with pyodbc.connect('Driver={ODBC Driver 17 for SQL Server
        };'
                        'Server=localhost;'
                        'Database=SMIOT;'
                        'Trusted_Connection=yes;') as conn:
      cur = conn.cursor()
      cur.execute("INSERT INTO data_all (unix, datestamp, 
          username, absolute_load, accelerator_pos_d, 
          accelerator_pos_e, accelerator_pos_f, 
          acelerometer_x, acelerometer_y, acelerometer_z, 
          air_quality, engine_load, fuel_rate, gps_alt, 
          gps_lat, gps_lon, gps_time, hr, humidity, 
          intake_pressure, light, maf, noise, 
          relative_accel_pos, relative_throttle_pos, rpm, rr
          , run_time, speed, temperature, throttle_pos, 
          throttle_pos_b, throttle_pos_c) VALUES
        (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?
        ,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)",str(unix), datestamp,
    username, values[0], values[1], values[2], values[3],
   values[4], values[5], values[6], values[7], values[8],
   values[9], values[10], values[11],values[12], values[13],
   values[14], values[15], values[16], values[17], values[18],
    values[19], values[20], values[21], values[22], values
   [23], values[24], values[25], values[26], values[27],
   values[28], values[29])
      conn.commit()
    return True
  except Exception as e:
    logging.error(e)
    return False
```

```
def ReadDatabaseTables():
  """Read the names of the tables in a database"""
  try:
    with pyodbc.connect('Driver={ODBC Driver 17 for SQL Server
       };''Server=localhost;''Database=SMIOT;''
       Trusted_Connection=yes;') as conn:
      cur = conn.cursor()
      sql = "SELECT Distinct TABLE_NAME FROM
         information_schema.TABLES;"
      cur.execute(sql)
      tables = cur.fetchall() #Returns the data as a tuple
      tablesList = []
      data = [0]
      for table in tables:
        tablesList.append(''.join(table)) #Tuple to list of
           strings
      #tablesList.remove('LASTID')
  except Exception as e:
    print(e)
    return False
  except Exception as e:
    logging.error(e)
    return False
  else:
    return tablesList


def query_db(tableName, lastTime,keyword=""):
  """Get the N first rows of a table and format a JSON object
     to send it to the server"""
  try:
    global username
```

```
    with pyodbc.connect('Driver={ODBC Driver 17 for SQL Server
        };''Server=localhost;''Database=SMIOT;''
        Trusted_Connection=yes;') as conn:
      cur = conn.cursor()
      sql = "SELECT * FROM "+tableName + \
        " WHERE (datecol>=TRY_PARSE(''"+str(lastTime- timedelta
            (seconds=1))+"' AS DATETIME) and datecol<=TRY_PARSE
            (''"+str(lastTime+ timedelta(seconds=1))+"' AS
            DATETIME) and username='"+username+"');"
      #print(sql)
      cur.execute(sql)
      #print("TBA")
      if (tableName == "acelerometer"):
        r = cur.fetchall()
      else:
        r = cur.fetchone()
      return r
  except Exception as e:
    print(e)
    return None
    #sys.exit()

def query_db_time():
  """Get the N first rows of a table and format a JSON object
     to send it to the server"""
  try:
    global username
    with pyodbc.connect('Driver={ODBC Driver 17 for SQL Server
        };''Server=localhost;''Database=SMIOT;''
        Trusted_Connection=yes;') as conn:
      cur = conn.cursor()
```

```
        sql = "SELECT datestamp FROM data_all WHERE( username='"
            +username+"') ORDER BY datestamp desc;"
        #print(sql)
        cur.execute(sql)
        r = cur.fetchone()
        #print(r)
        if (r != None):
          return r[0]
        else:
          global time_unix
          return time_unix
    except Exception as e:
      print(e)
      sys.exit()

def query_db_time_table(time):
    """Get the N first rows of a table and format a JSON object
       to send it to the server"""
    try:
      global username
      with pyodbc.connect('Driver={ODBC Driver 17 for SQL Server
          };''Server=localhost;''Database=SMIOT;''
          Trusted_Connection=yes;') as conn:
        cur = conn.cursor()
        sql = "SELECT datecol FROM gps_lat WHERE (datecol>
            TRY_PARSE('"+str(time)+"' AS DATETIME) and username='
            "+username+"');"
        #sql = "SELECT unix, username FROM acelerometer WHERE (
            unix >1364202290) ORDER BY UNIX asc"
        #print (sql)
        cur.execute(sql)
```

```
        r = cur.fetchone()
        #print(r)
        return r[0]
    except Exception as e:
        print(e)
        sys.exit()

def query_db_data():
    """Get the N first rows of a table and format a JSON object
       to send it to the server"""
    try:
        global username
        with pyodbc.connect('Driver={ODBC Driver 17 for SQL Server
            };''Server=localhost;''Database=SMIOT;''
            Trusted_Connection=yes;') as conn:
            cur = conn.cursor()
            sql = "SELECT * FROM data_all WHERE (username='"+
                username+"') ORDER BY datestamp desc ;"
            #print(sql)
            cur.execute(sql)
            r = cur.fetchone()
            print(r)
            return r
    except Exception as e:
        print(e)
        sys.exit()

def main(argv):
    #time.sleep(10)
    #create_table()
    tables1 = ReadDatabaseTables()
```

```
t = ["absolute_load" , "accelerator_pos_d" , "
    accelerator_pos_e" , "accelerator_pos_f" , "acelerometer"
     , "air_quality" , "engine_load" , "fuel_rate" , "gps_alt
    " , "gps_lat" , "gps_lon" , "gps_time" , "hr" , "humidity
    " , "intake_pressure" , "light" , "maf" , "noise" , "
    relative_accel_pos" , "relative_throttle_pos" , "rpm" , "
    rr" , "run_time" , "speed" , "temperature" , "
    throttle_pos" , "throttle_pos_b" , "throttle_pos_c" ]
#print(tables1)
timeLast = 0
data = [-9999, -9999, -9999, -9999, -9999, -9999, -9999,
    -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999,
    -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999,
    -9999, -9999, -9999, -9999, -9999, -9999, -9999]
try:
  aux = query_db_data()
  data = np.asarray(aux[3:])
  timeLast = aux[1]
  #timeLast = datetime.strptime(timeLast,'%Y-%m-%d %H:%M:%S
      ')
  #print(data)
except Exception as e:
  print(e)
  timeLast = datetime.utcfromtimestamp(1).strftime('%Y-%m-%d
     %H:%M:%S')
  timeLast = datetime.strptime(timeLast,'%Y-%m-%d %H:%M:%S')
while True:
  index = 0
  global time_unix
  #print("TREU")
  #print(time_unix)
```

```
try:
    time_unix = query_db_time()
    #print(time_unix)
    #time_unix = datetime.strptime(time_unix,'%Y-%m-%d %H:%M
        :%S')
    #print(time_unix)
except Exception as e:
    print(e)
    print("DATA_ALL_EMPTY")
try:
    time_unix = query_db_time_table(time_unix)
    #time_unix = datetime.strptime(time_unix,'%Y-%m-%d %H:%M
        :%S')
    #print(time_unix)
    #print(timeLast)
    #print(time_unix - timeLast)
    if ((time_unix - timeLast).seconds > 100):
      data = [-9999, -9999, -9999, -9999, -9999, -9999,
          -9999, -9999, -9999, -9999, -9999, -9999, -9999,
          -9999, -9999, -9999, -9999, -9999, -9999, -9999,
          -9999, -9999, -9999, -9999, -9999, -9999, -9999,
          -9999, -9999, -9999, -9999, -9999, -9999, -9999,
          -9999, -9999, -9999]

    #print(time_unix)
except Exception as e:
    print(e)

try:
    for table in tables1:
      try:
        if table in t:
```

```
        #print(table)

        list = query_db(table, time_unix)
        #print (list)
        if table == "acelerometer":
          try:
            for i in range(0,3):
              #print(list[i][3])
              #print(list[i])
              if (list!= None):
                data[index] = list[i][3]
              index+=1
          except Exception as e:
            index = 7
            print(e)
        else:
          #print(list[3])
          #print(list)
          if (list!= None):
            data[index]=(list[3])
          index+=1
        #time.sleep(1)
      except Exception as e:
        print(table)
        print(e)
        time.sleep(1)
    #print(data)
    print(time.mktime(time_unix.timetuple()))
    #print(datetime.utcfromtimestamp(time_unix).strftime('%Y
        -%m-%d %H:%M:%S'))
```

```
        update_table(time.mktime(time_unix.timetuple()),
            time_unix, data)
        timeLast = time_unix
      except Exception as e:
        #print(table)
        print(e)


if __name__ == "__main__":
  try:
    main(sys.argv[:1])
  except KeyboardInterrupt as e:
    print(e)
```

Cuadro 2.8.- Script tabla_sinc.py

El script presenta las siguientes funciones:

- **create_table**: función usada para crear la tabla data_all, donde se incluyen los datos, la marca de tiempo, el unix, y el usuario que recogió los datos.
- **update_table**: función para añadir datos a la tabla data_all.
- **query_db**: función que devuelve, de existir, el dato comprendido un segundo antes y un segundo después de la marca de tiempo que se le pase por argumento y asignado al nombre de usuario correspondiente. Si la tabla de la que se recogen los datos es la del acelerómetro se devuelven todos los datos recogidos, debido a que en esta tabla se guardan las aceleraciones en X, Y y Z. Para el resto de casos se devuelve sólo 1 valor.
- **query_db_time**: devuelve la última marca de tiempo guardada en la tabla data_all de existir.
- **query_db_time_table**: devuelve la siguiente marca de tiempo a la que se le pasa por argumento, de la tabla gps_lat.

- **query_db_data**: devuelve la última fila guardada en la tabla data_all, si existe.

El programa principal primero crea un *array* con el nombre de las tablas cuyos datos se quiere guardar en la tabla conjunta. Posteriormente, crea un *array* de 23 elementos con el valor -9999 llamado data.

A continuación, llama a la función **query_db_data** y comprueba si existen valores almacenados en la tabla. De existir, se guardan los valores en el *array* data y se coge la marca de tiempo del valor segundo del *array* devuelto por la función en el valor *timeLast*. En caso de que no existan datos, se coge una marca de tiempo suficientemente antigua para asegurar que no existan datos de ese momento.

Después se entra en el bucle infinito. Dentro, se llama a la función **query_db_time** para sacar el valor del tiempo y se almacena en la variable *time_unix*. Posteriormente, se llama a la función **query_db_time_table** con el tiempo sacado de la anterior función y se guarda en *time_unix*. Por último, se comprueba que el tiempo que pasó entre el último tiempo almacenado en la tabla *data_all* y el tiempo que se va a analizar a continuación no estén separados más de 100 segundos. En el caso de que sí estén separados más de 100 segundos, se reseteará el *array* data a los valores -9999 mencionados anteriormente.

Lo último que se hace en el bucle es llamar a la función **query_db** con todas las tablas del *array* creado al principio del programa principal y la marca de tiempo que se está analizando, y guardar los valores, si existen, en el *array* data. Una vez recorrido el *array* de las tablas entero, se llamará a la función **update_table** para que estos valores se guarden en la base de datos.

Mencionar que el *unix* se guarda en formato de texto porque, si se hace en formato real, SQL lo reducía a su expresión científica, perdiendo precisión (6 dígitos). Al guardarlo como texto, SQL no simplifica nada y ayuda a la rapidez de análisis de los datos.

2.4.- Prueba de viaje

Dentro de este apartado se mostrarán los datos de un viaje realizado entre Gijón y Pola de Siero. Primeramente, se mostrará el trazado del viaje, recogido en la Figura 2.16.

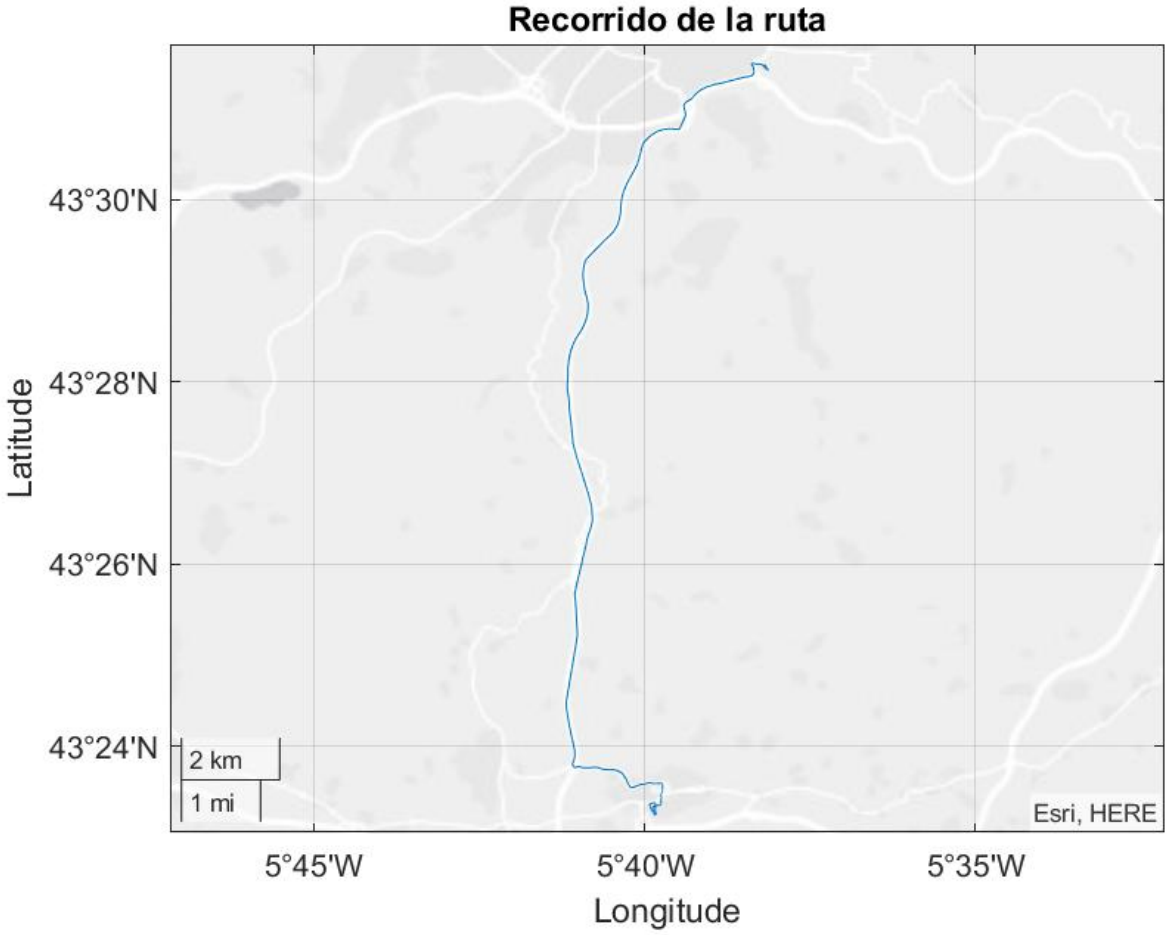

Figura 2.16.- Ruta del viaje de prueba

A continuación se mostrarán gráficas de la altitud, la calidad del aire, la humedad, el nivel de luz, la presión, la velocidad y la temperatura en el viaje, y del pulso y el intervalo R-R del conductor. No se incluyen los resultados del OBD porque no funciona en el vehículo de prueba y del ruido por no haber conseguido calibrarlo correctamente.

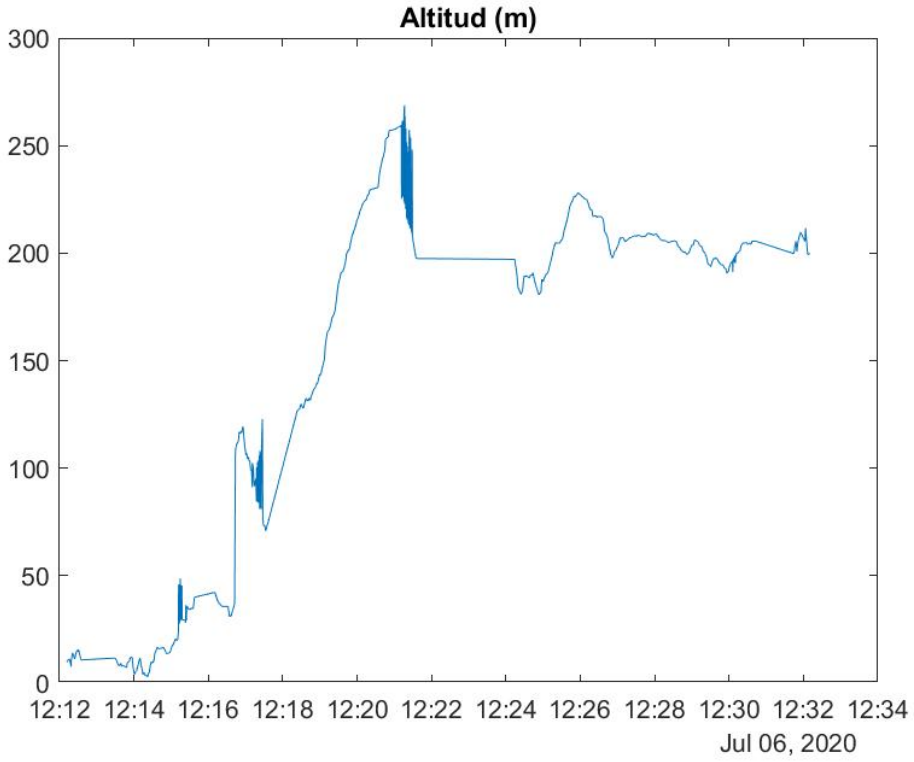

Figura 2.17.- Altitud del viaje de prueba

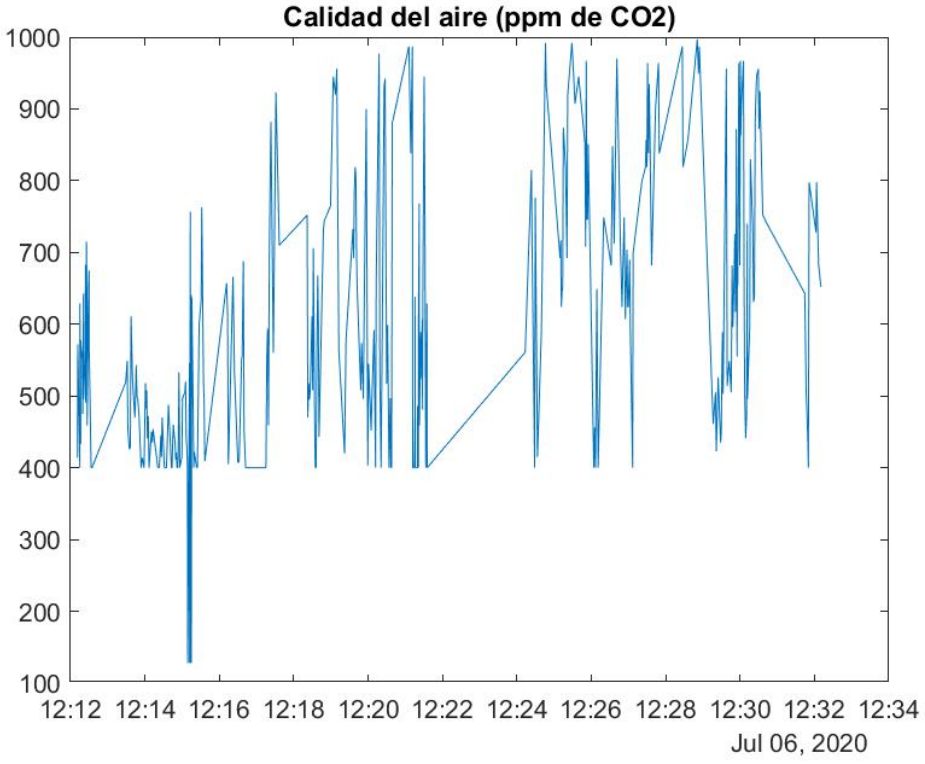

Figura 2.18.- Calidad del aire en el viaje de prueba

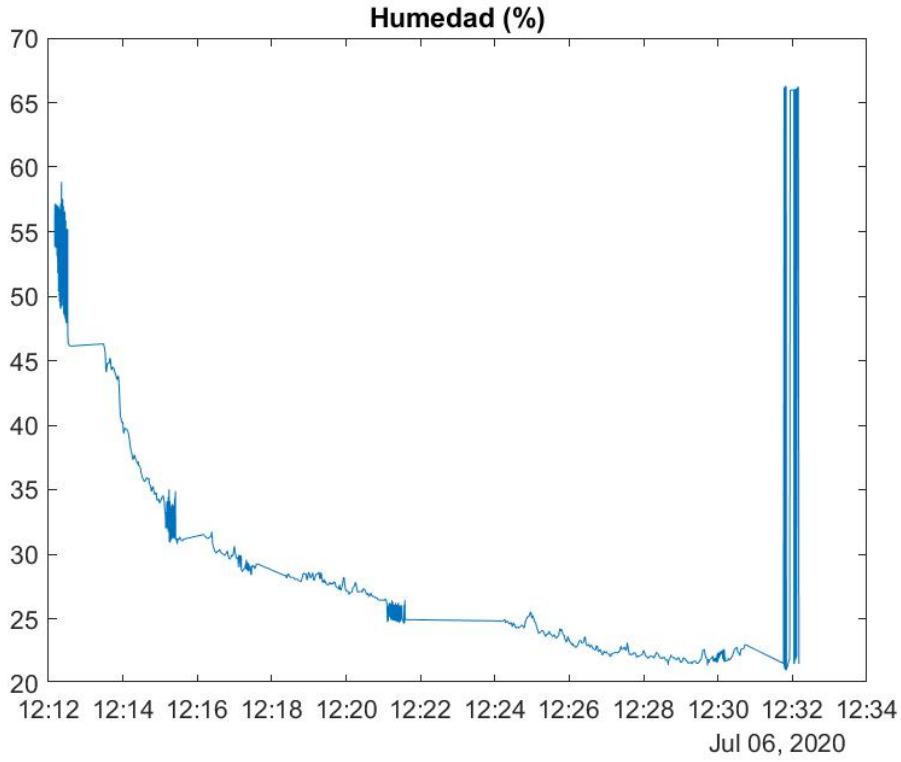

Figura 2.19.- Humedad en el viaje de prueba

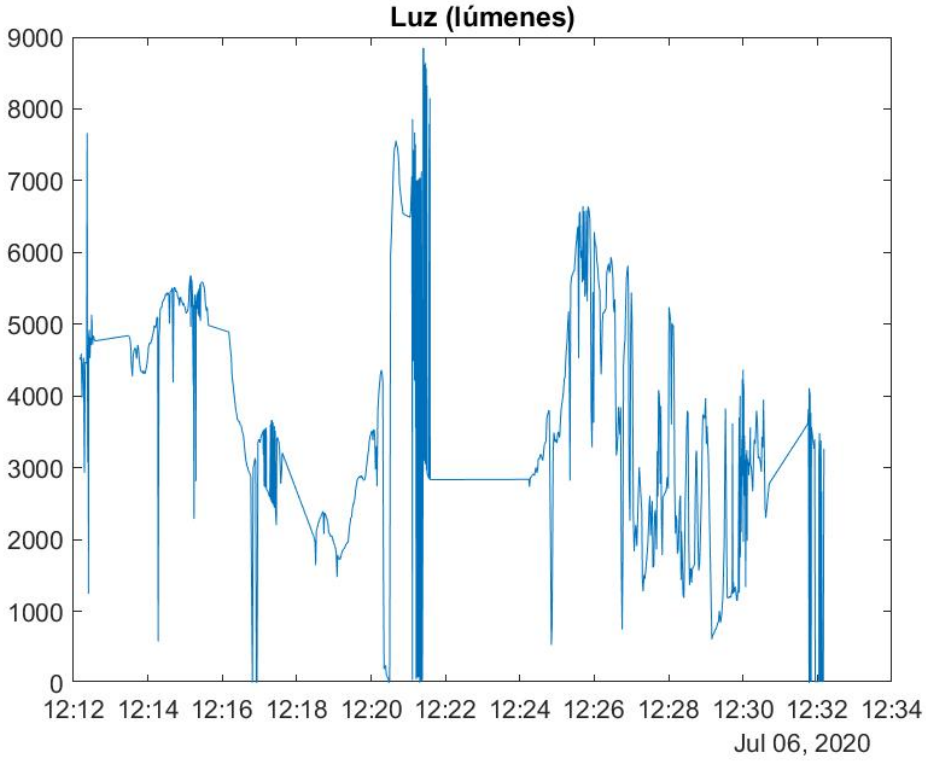

Figura 2.20.- Nivel de luz en el viaje de prueba

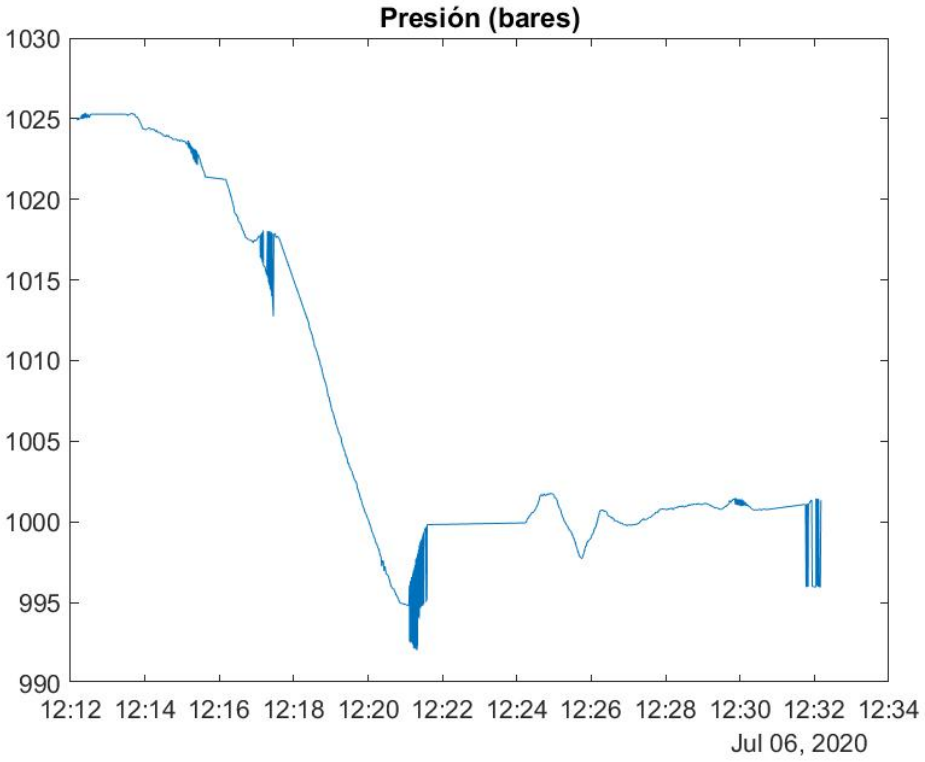

Figura 2.21.- Presión en el viaje de prueba

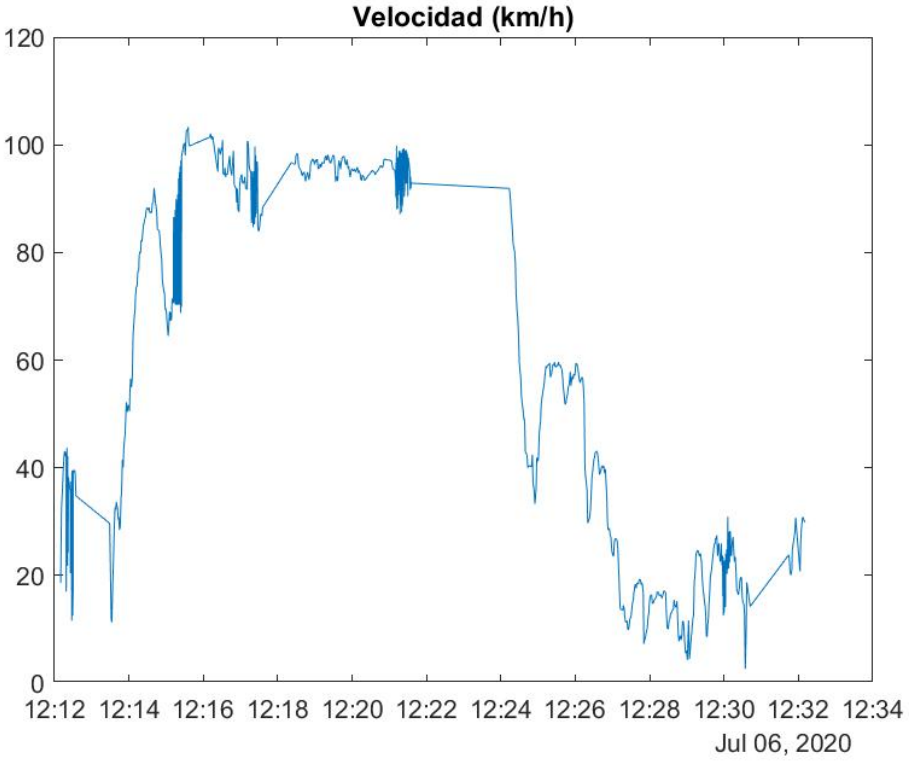

Figura 2.22.- Velocidad del viaje de prueba

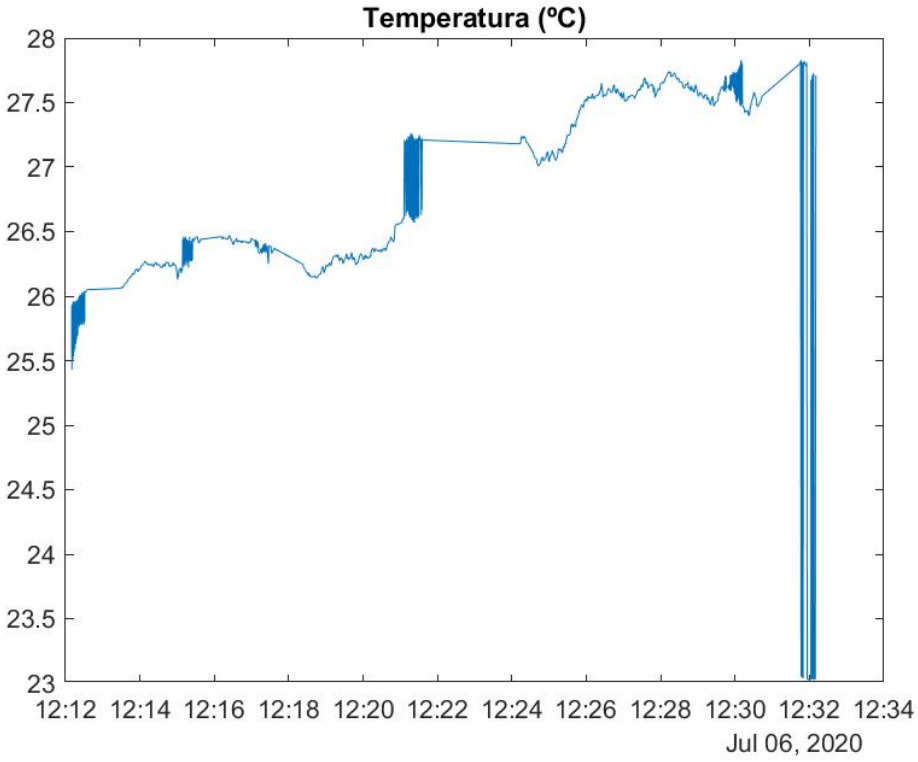

Figura 2.23.- Temperatura en el viaje de prueba

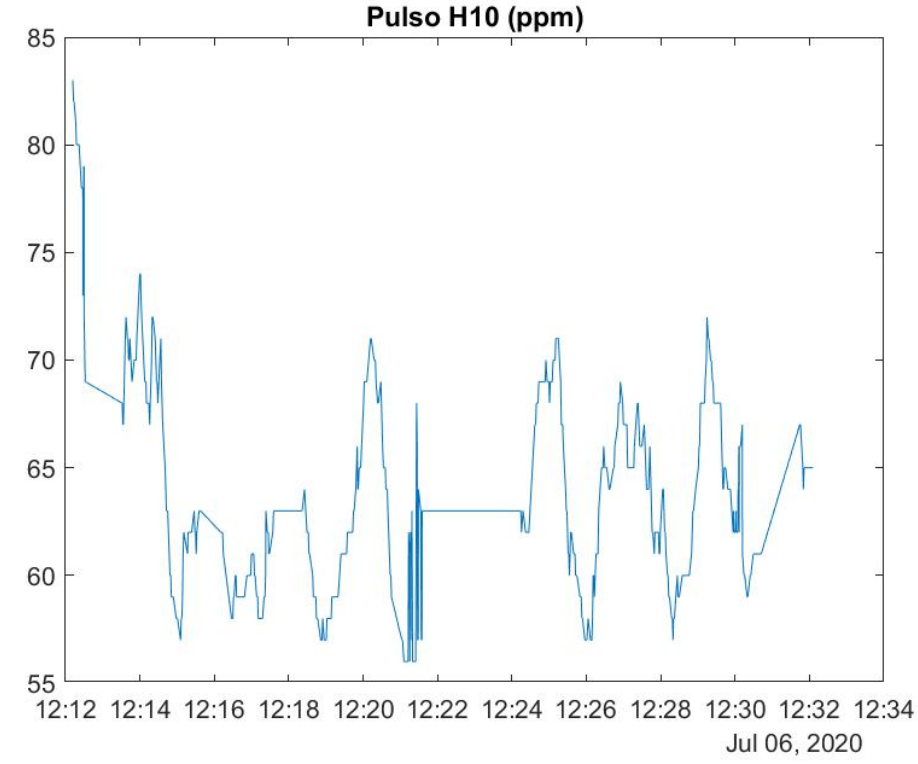

Figura 2.24.- Pulso del conductor en el viaje de prueba

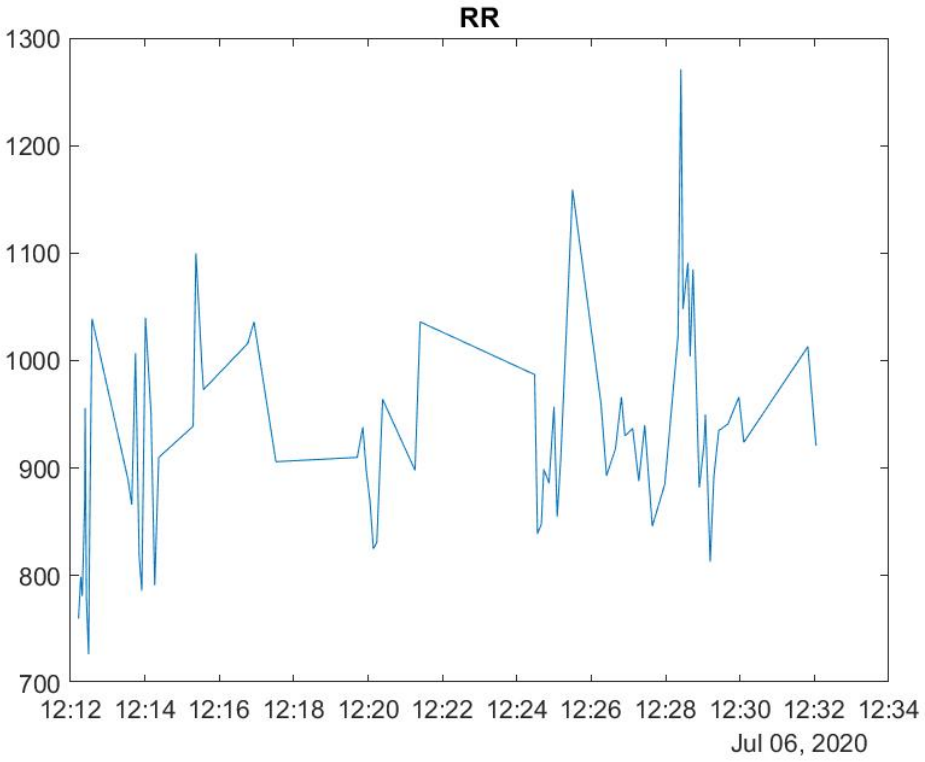

Figura 2.25.- Intervalo R-R del conductor en el viaje de prueba

3. Parte industrial

En esta sección se explicará el desarrollo del proyecto industrial, empezando por el prototipo y sus requisitos y, posteriormente, explicando el estudio que se planea hacer en un futuro.

3.1.- Prototipo

En esta sección se explicará el diseño de un posible prototipo para cubrir los requisitos de diseño expuestos por diferentes empresas asturianas. Además, se explicará la interfaz desarrollada para este prototipo.

3.1.1.- Requisitos de diseño

Para los requisitos de diseño del prototipo se ha consultado a múltiples empresas asturianas con vehículos de carácter industrial como carretillas elevadoras, palas mecánicas, camiones, etc.

Cada empresa tenía unos requisitos propios, pero los comunes a todas eran dos: por un lado, se requiere que los vehículos estuviesen localizados en todo momento tanto en interiores como en exteriores. Los responsables comentaban que se perdía una gran cantidad de tiempo intentando localizar los vehículos para su uso, puesto que no se encontraban en las zonas designadas para su estacionamiento debido a que los trabajadores usaban estos vehículos y los dejaban, sin llevarlos de vuelta a estas zonas.

El otro punto importante que destacaban era la necesidad que tenían los conductores de aumentar la seguridad. En una fábrica existen ángulos muertos, giros de 90 grados sin visión, etc, que producen un peligro potencial tanto al conductor como a los trabajadores que estén trabajando allí. El mecanismo que pretendían instalar era uno basado en la detección, distinguiendo trabajadores de otras máquinas industriales.

A estos dos aspectos, se les ha añadido un tercero, que consiste en la señalización de tendido eléctrico peligroso. En las fábricas puede haber tendido eléctrico que, conduciendo de una manera normal no implique peligro pero, al subir y bajar una carretilla elevadora, pueda tocarse y producir una descarga mortal.

3.1.2.- Posible diseño con sensores

Se ha diseñado una posible solución para el sistema con un formato simplificado, en la que se prescinde de la diferenciación entre personal y obstáculos fijos. Se podrá conocer la posición de las máquinas mediante el sistema de localización.

El sistema de detección de obstáculos se basa en la utilización de sensores ultrasónicos (Figura 3.1) con una raspberry pi, dispuestos de manera que cubra la totalidad o al menos la gran mayoría de los ángulos de la carretilla. Los sensores se colocarían tal y como se muestra en la Figura 3.2.

Figura 3.1.- Sensores ultrasónicos HC-SR04

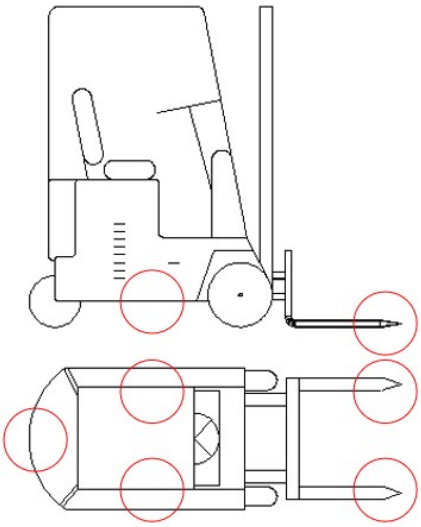

Figura 3.2.- Colocación de los sensores en la carretilla elevadora

Estos sensores miden, teóricamente hasta una distancia de 10 metros. Sin embargo, a partir de 4 o 5 metros, sus medidas contienen bastante más errores. Este sistema se ha diseñado para entornos industriales interiores, teniendo en cuenta las limitaciones de velocidades (10 km/h como máximo). De esta forma, los 4/5 metros de resolución que nos da el sistema cuando avanza la carretilla es suficiente.

Por otro lado, mencionar que cuanto mayor sea el alcance de los sensores, menor es el haz de detección. Como para giros rectos según la regulación la carretilla debería estar parada, la resolución necesaria es de 2 metros, con un sensor en cada lado conseguimos cubrir la gran mayoría de zonas alrededor de la carretilla. De igual forma ocurre en la parte trasera. En la parte delantera, al necesitar más resolución en distancia, se colocarán dos sensores para cubrir el haz necesario.

Para terminar con este sistema, se muestra un diagrama de bloques muy sencillo del funcionamiento (Figura 3.3), donde la información que recogen los sensores ultrasónicos se pasa a la Raspberry Pi que se la muestra al usuario a través de dos interfaces que se explicarán a continuación.

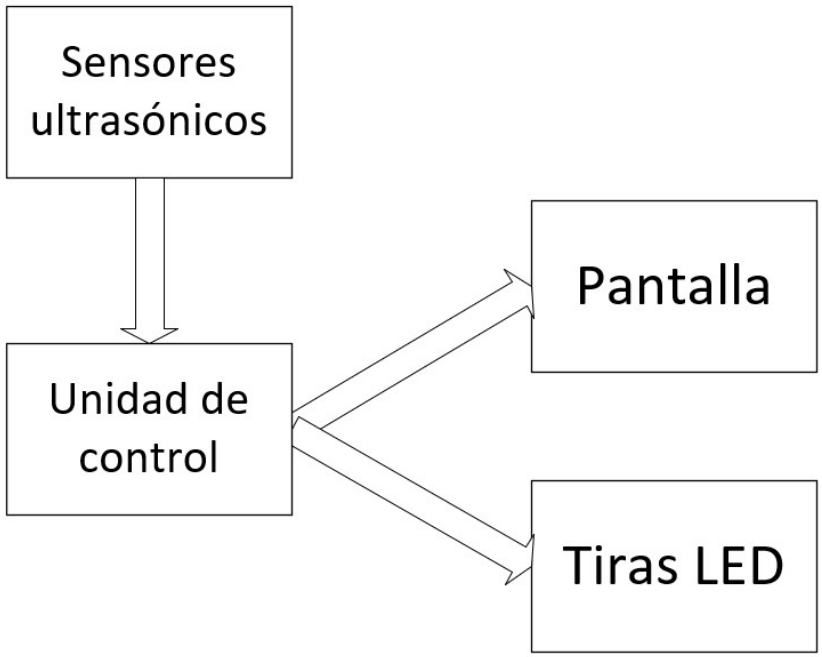

Figura 3.3.- Esquema de funcionamiento del sistema

Con respecto a la localización, se ha optado por la opción de un GPS para exteriores, y para interiores se está barajando la opción de la utilización de un sistema híbrido de acelerómetro y balizas WiFi, si bien se encuentra aún en desarrollo.

En un principio se quería usar sólo un acelerómetro para este aspecto. Sin emabrgo, debido a la fuente deriva de sus medidas, es compleja su correcta calibración. Esta información será enviada a un sistema central que repartirá esta información a los distintos vehículos en uso. Esta información será usada por el sistema de detección de obstáculos para informar al usuario de la presencia de otra maquinaria. La información de localización deberá ser precisa ya que será crítica a la hora de detectar la presencia o no de otro vehículo.

Por último, mencionar que, además de los sensores utilizados se tendrá que diseñar un encapsulado que sea resistente a posibles golpes o movimientos bruscos de la maquinaria industrial.

3.1.3.- Interfaz con el usuario

La interfaz con el usuario se ha diseñado de forma que se muestre a través de una pantalla y de unas tiras LED. La información completa se mostrará en la pantalla y las tiras LED sólo avisarán del peligro inminente.

3.1.3.1.- Interfaz de pantalla

La interfaz se ha desarrollado con PyQT. Se ha valorado otra alternativa, el *framework* kivy de Python, pero su consumo excesivo de recursos (requería del 100 % de recursos de la Raspberry para funcionar) la hacían una opción poco viable.

Para el diseño se ha utilizado la interfaz gráfica, que genera un archivo UI. Mediante PyQT se genera el archivo de interfaz a partir del ui, y se importa al programa principal.

Dentro del programa principal existen una clase **MainWindow**, con un método **init** asociado. En este método se colocan todas las imágenes en los recuadros adecuados de la interfaz y que lanza varios timers para que se actualicen las imágenes. En cada método asociado a los timers se comprobará la existencia o no de peligros y se mostrará o no las flechas y el número adecuado. La interfaz se muestra en la Figura 3.4. El script asociado se muestra en el Cuadro 3.1.

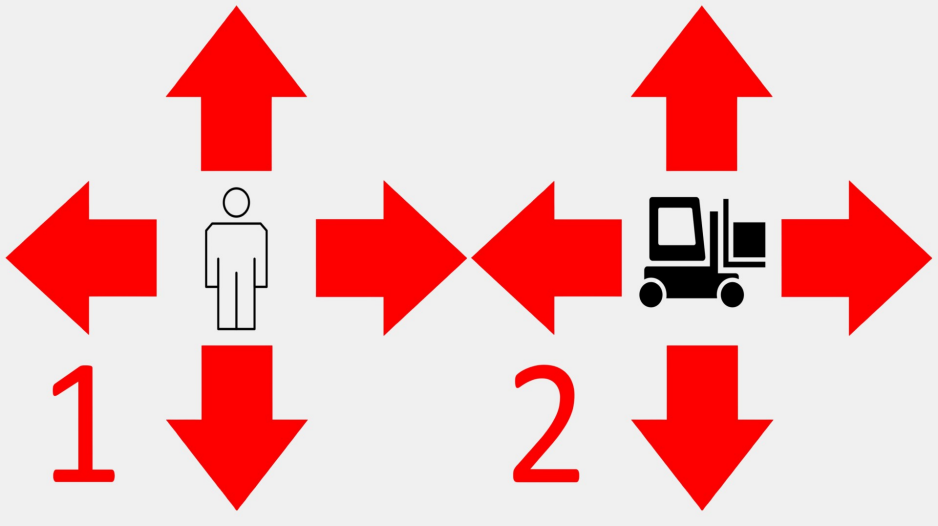

Figura 3.4.- Interfaz gráfica desarrollada

```
from interface_ui import *
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *
#import RPi.GPIO as GPIO
import pygame
import math
import sys
from ultrasonicsensor import ultrasonicRead
from ultrasonicsensor_r import ultrasonicRead_r
import time
n=0
first_r = False
first_f = False
image = ""
```

```
#GPIO.setmode(GPIO.BOARD)
TRIG_F = 16
ECHO_F = 24
TRIG_D = 18
ECHO_D = 40
#GPIO.setup(TRIG_F, GPIO.OUT)
#GPIO.setup(ECHO_F, GPIO.IN)
#GPIO.setup(TRIG_D, GPIO.OUT)
#GPIO.setup(ECHO_D, GPIO.IN)


class MainWindow(QtWidgets.QMainWindow, Ui_MainWindow):
  def __init__(self, *args, **kwargs):
    #QtWidgets.QMainWindow.__init__(self, *args, **kwargs)
    super(MainWindow, self).__init__(*args, **kwargs)
    self.setupUi(self)

    #global image
    #image = "./persona.png"

    #Persona
    pixmap = QtGui.QPixmap("./persona.png")
    self.persona.setPixmap(pixmap)
    self.persona.show()
    self.persona.setScaledContents(True)

    #Flecha arriba
    pixmap = QtGui.QPixmap("./arriba.png")
    self.arriba_p.setPixmap(pixmap)
    self.arriba_p.show()
    self.arriba_p.setScaledContents(True)
```

```
#Flecha abajo
pixmap = QtGui.QPixmap("./abajo.png")
self.abajo_p.setPixmap(pixmap)
self.abajo_p.show()
self.abajo_p.setScaledContents(True)

#Flecha izquierda
pixmap = QtGui.QPixmap("./izquierda.jpg")
self.izquierda_p.setPixmap(pixmap)
self.izquierda_p.show()
self.izquierda_p.setScaledContents(True)

#Flecha derecha
pixmap = QtGui.QPixmap("./derecha.png")
self.derecha_p.setPixmap(pixmap)
self.derecha_p.show()
self.derecha_p.setScaledContents(True)

#Numero
pixmap = QtGui.QPixmap("./1.png")
self.numero_p.setPixmap(pixmap)
self.numero_p.show()
self.numero_p.setScaledContents(True)

#Maquina
pixmap = QtGui.QPixmap("./machine.jpg")
self.maquina.setPixmap(pixmap)
self.maquina.show()
self.maquina.setScaledContents(True)
```

```
#Flecha arriba
pixmap = QtGui.QPixmap("./arriba.png")
self.arriba_m.setPixmap(pixmap)
self.arriba_m.show()
self.arriba_m.setScaledContents(True)

#Flecha abajo
pixmap = QtGui.QPixmap("./abajo.png")
self.abajo_m.setPixmap(pixmap)
self.abajo_m.show()
self.abajo_m.setScaledContents(True)

#Flecha izquierda
pixmap = QtGui.QPixmap("./izquierda.jpg")
self.izquierda_m.setPixmap(pixmap)
self.izquierda_m.show()
self.izquierda_m.setScaledContents(True)

#Flecha derecha
pixmap = QtGui.QPixmap("./derecha.png")
self.derecha_m.setPixmap(pixmap)
self.derecha_m.show()
self.derecha_m.setScaledContents(True)

#Numero
pixmap = QtGui.QPixmap("./2.png")
self.numero_m.setPixmap(pixmap)
self.numero_m.show()
self.numero_m.setScaledContents(True)

#time.sleep(5)
```

```
    #self.loop()
    print("main")

    self.timer = QTimer()
    self.timer.setInterval(100)
    self.timer.timeout.connect(self.update_front)
    self.timer.start()

    time.sleep(0.05)

    self.timer2 = QTimer()
    self.timer2.setInterval(100)
    self.timer2.timeout.connect(self.update_right)
    self.timer2.start()

def update_front(self):
    global n
    global first_f
    global image
    if (n>5):
        pixmap = QtGui.QPixmap("./machine.jpg")
        self.persona.setPixmap(pixmap)
        self.persona.show()
        self.persona.setScaledContents(True)
        self.persona.clear()
    if (n>10):
        self.persona.setPixmap(QtGui.QPixmap("./persona.png"))
    n = n + 1
    '''distance = ultrasonicRead(GPIO, TRIG_F, ECHO_F)
    if (distance>5 and distance<100):
        self.arriba_p.setPixmap(QtGui.QPixmap("./arriba.png"))
```

```
        if (first_f == False):
            n = n + 1
            first_f = True
        if (n==1):
            self.numero_p.setPixmap(QtGui.QPixmap("./1.png"))
        elif (n==2):
            self.numero_p.setPixmap(QtGui.QPixmap("./2.png"))
    else:
        if (first_f == True):
            n = n - 1
            first_f = False
        self.arriba_p.clear()
        if (n==0):
            self.numero_p.clear()
    #print(n) '''

def update_right(self):
    global n
    global first_r
    global image
    if (n>5):
        pixmap = QtGui.QPixmap("./machine.jpg")
        self.persona.setPixmap(pixmap)
        self.persona.show()
        self.persona.setScaledContents(True)
        self.persona.clear()
    if (n>10):
        self.persona.setPixmap(QtGui.QPixmap("./persona.png"))
    n = n + 1
    '''distance = ultrasonicRead(GPIO, TRIG_D, ECHO_D)
    if (distance>5 and distance<100):
```

```
        self.derecha_p.setPixmap(QtGui.QPixmap("./derecha.png"))
        if (first_r == False):
            n = n + 1
            first_r = True
        if (n==1):
            self.numero_p.setPixmap(QtGui.QPixmap("./1.png"))
        elif (n==2):
            self.numero_p.setPixmap(QtGui.QPixmap("./2.png"))
    else:
        if (first_r == True):
            n = n - 1
            first_r = False
        self.derecha_p.clear()
        if (n==0):
            self.numero_p.clear()
    print(distance)'''



if __name__ == "__main__":
    app = QtWidgets.QApplication([])
    window = MainWindow()
    window.showFullScreen()

    sys.exit(app.exec_())
```

Cuadro 3.1.- Script interfaz.py

3.1.3.2.- Interfaz led

Para la interfaz LED se han usado tiras LED mostradas en la figura 1.9. Se colocarán 4, en las 4 barras verticales de la máquina en cuestión. El aviso se hará bien por parpadeos, subiendo la frecuencia de parpadeos según el peligro esté más cerca, cambiando el color de las tiras entre amarillo, naranja y rojo o aumentando el brillo de las tiras. La elección de uno de estos 3 métodos se hará a través del estudio explicado a continuación.

3.2.- Estudio

En esta sección se hablará del estudio que se tiene planeado realizar. Se explicarán sus objetivos, su forma de realización y todo lo que se ha desarrollado y construido para su ejecución.

3.2.1.- Objetivo del estudio

El objetivo del estudio es determinar la mejor forma de señalizar al usuario el peligro próximo o inminente, mediante la utilización de tiras LED dispuestas en los traveseños verticales de una carretilla elevadora.

La forma de señalizarlo se podrá extrapolar a diferentes vehículos industriales, pero el estudio se realizará en las condiciones de una carretilla elevadora.

Se analizará el tiempo de respuesta, el cansancio del sujeto de pruebas y la satisfacción y opinión del sujeto.

3.2.2.- Partes del estudio

El estudio se dividirá en 2 partes: una parte de ejercicios en las que se medirá tiempo de respuesta a través de unos ejercicios utilizando el volante y los pedales y unos test

para determinar el estado inicial y final del sujeto, además de la opinión personal sobre la mejor de las 3 opciones expuestas y su nivel de cansancio.

El modo en que transcurrirá el estudio será de la siguiente forma:

- Primero, se realizará un test para determinar el estado inicial del conductor. Se determinará el nivel de cansancio del sujeto, la experiencia en general en videojuegos de conducción y su experiencia en conducción.

- A continuación, se pondrá al sujeto a jugar a un simulador de conducción sencillo en el que se quitará cualquier otro vehículo y se irán realizando 7 pruebas diferentes, cuyo orden irá variando entre diferentes sujetos con el objetivo de determinar si un ejercicio produce más cansancio que otro. Después de cada prueba se realizará un pequeño test llamado NASA para determinar diferentes aspectos de la prueba. Las pruebas serán las siguientes: cambio en el brillo de los LED entre 3 intensidades diferentes; cambio en el color de los LED entre 3 colores diferentes (amarillo, naranja y rojo); cambio en el parpadeo de las tiras, entre 3 frecuencias diferentes. Cuando llegue el último nivel de los ejercicios se determinará el tiempo de reacción entre que aparece y el sujeto pisa el freno. Se realizarán 10 iteraciones por ejercicio. Tanto para el parpadeo como para el aumento de brillo se usará el color rojo en los LED. Estas pruebas se realizarán tanto con los LED de delante como con los de atrás. Además, se realizarán dos últimas pruebas, solo con los LED de delante, en la que se cambiará el color de los LED a morado y se realizarán las pruebas de parpadeo y cambio de brillo. En este caso, tendrá que calcar a un botón del cuadro de mando situado al lado del volante para simularla subida y la bajada de material en la carretilla.

- Por último, se realizará un test final para determinar el cansancio de todos los ejercicios y la opinión personal de los usuarios.

Los tests se recogen en la Figura 3.5.

Pretest

1. Edad.
2. Sexo.
3. Con qué frecuencia se coge el coche a lo largo de la semana.
4. Es conductor profesional o no (Esta pregunta si se hace la primera iteración en la universidad no tiene mucho sentido, se podría preguntar si alguna vez los sujetos de pruebas han conducido una carretilla elevadora).
5. Mano dominante.
6. Daltónico.
7. Ha jugado alguna vez a juegos de conducción.
8. Cuántas horas lleva trabajando hoy.

Postest

1. Cómo de cansado está.
2. Qué método le ha parecido mejor.
3. Qué color hubiese elegido para la prueba del botón.
4. Cómo le ha parecido la prueba en conjunto: pesada, normal o entretenida.

Figura 3.5.- Prestest y postest

3.2.3.- Estructura

Para el diseño de la estructura exterior se han cogido medidas de una carretilla elevadora real a gasolina, modelo 5FGL18 de Toyota. Las medidas se han recogido en la Figura 3.6.

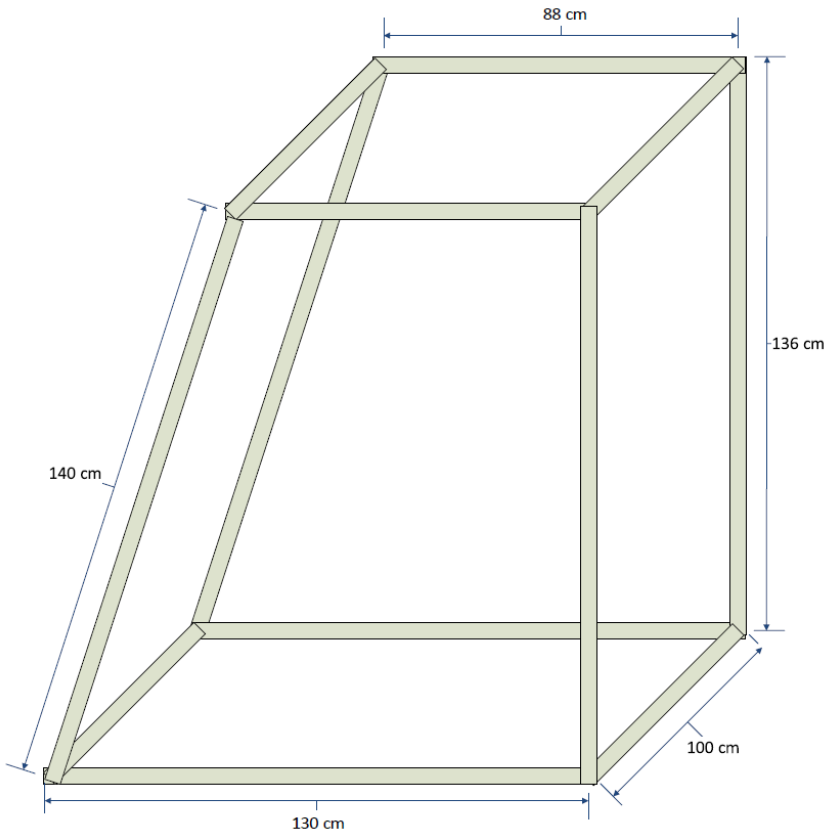

Figura 3.6.- Diseño de la estructura de la carretilla elevadora

Para su construcción, se han utilizado tubos de PVC de 40 mm de radio, y para las juntas se han diseñado e impreso en 3D codos. La unión entre codos y tubos se ha realizado con tubos termo retráctiles. Esto se muestra en la Figura 3.7.

Figura 3.7.- Estructura montada

Además, se ha incluido un volante y pedales con una estructura metálica para apoyarlos, y dos televisiones, una en la parte delantera y otra en la parte trasera, como se ve en la figura 3.8.

Figura 3.8.- Entorno de pruebas del estudio

Además, se ha diseñado dentro del grupo una placa de circuito impreso con dos objetivos: por un lado, aportar la alimentación a las 4 tiras LED de la estructura; y por otro, como la raspberry sólo tiene 2 canalas PWM y se quiere controlar 4 tiras LED, pero sólo en ejercicios con 2 a la vez, la placa servirá para cambiar el canal PWM entre las tiras delanteras y traseras. Esta PCB se muestra en la figura 3.9.

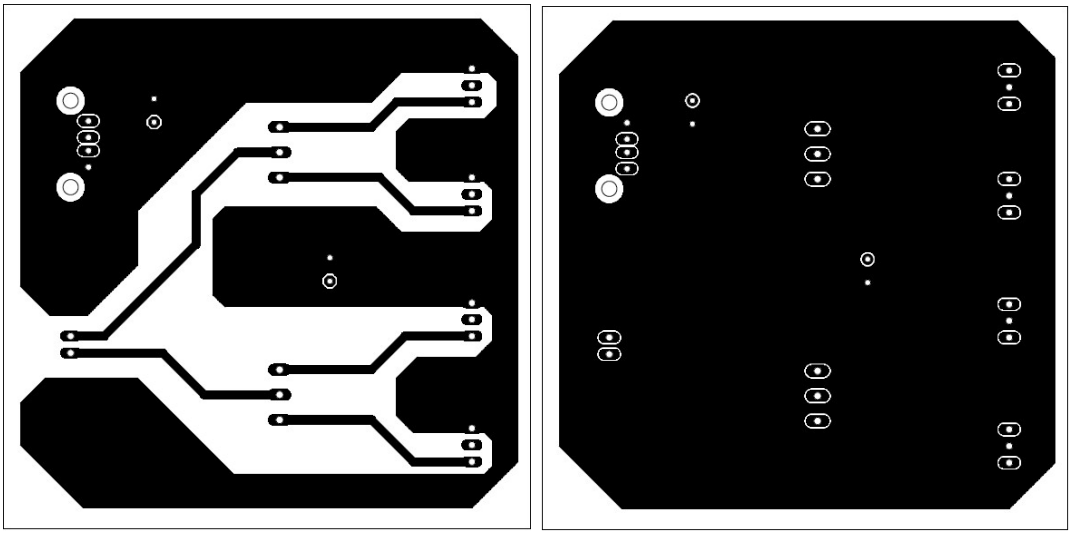

Figura 3.9.- Parte top y bottom de la PCB

3.2.4.- Scripts

Dentro de este apartado se distinguirán 2 scripts: el script ejecutado en la máquina Windows que se encargará de recibir los comandos de los pedales y enviarlos a la raspberry; y el script de la raspberry que cambiará los LEDs y guardará un registro de los tiempos de respuesta.

3.2.4.1.- Script volante.py

El script volante.py se recoge en el cuadro 3.2.

```
import pygame
import time
import os
import math

class DataControlLayer():
    sock = ""

    def connect(self):
```

```
        host = "169.254.93.88"
        port = 8000     # The same port as used by the server
        self.sock = socket.socket(socket.AF_INET, socket.
            SOCK_STREAM)
        self.sock.connect((host, port))

    def disconnect(self):

        self.sock.close()

    def sendmsg(self, text):

        self.connect()
        self.sock.sendall(text.encode('utf-8'))
        self.disconnect()
        #self.sock.sendall(text)


if __name__ == '__main__':
    conn = DataControlLayer()

    pygame.init()
    joystick = pygame.joystick.Joystick(0)
    joystick.init()
    print ("Detected joystick ")
    print("Name of joysticks: {}".format(joystick.get_name()))
    #print("Number of joysticks: {}".format(joystick.
        get_numballs()))
    joystick_count = pygame.joystick.get_count()
    i = 0
```

```
while(True):
    '''name = joystick.get_name()
    print("Name: "+name)

    axes = joystick.get_numaxes()
    for i in range(axes):
        axis = joystick.get_axis(i)
        print("Axis: "+str(axes)+"      i: "+str(i))

    buttons = joystick.get_numbuttons()

    print("Buttons: "+str(buttons))
    '''
    pygame.event.get()

    #axis(0) --> volante (derecha positivo, izquierda
        negativo) entre -1 y +1
    #axis(1) --> acelerador entre 0 (parado) y -1 (pisado)
        y freno entre 0 (parado) y +1 (pisado)

    x_r = joystick.get_axis(1)
    print(x_r)

    if(x_r > 0.5):
        conn.sendmsg("STOP")
```

Cuadro 3.2.- Script volante.py

Se usa la librería PyGame que incluye una clase dedicada a los joysticks. El constructor de la clase detecta automáticamente los joysticks (tanto mandos como volantes) e inicia su ejecución. Dentro del bucle se recogen los eventos del conjunto, es decir, tanto el volante como los pedales y los botones del controlador adicional y se

guardan en diferentes categorías como *axis* y *buttons*. En el caso de este script interesa recoger los datos del pedal de freno, que se encuentran dentro del axis 1. Este eje incluye el acelerados, cuyos valores varían entre 0 y -1 y el freno, cuyos valores se encuentran entre 0 y 1. De esta forma, cuando se detecte que hay algún valor por encima de 0,5 envía un mensaje *STOP* a la Raspberry a través de la función **sendmsg** incluida en la clase **DataControlLayer**. Esta clase inicia una conexión de tipo *socket* en el puerto 8000 con la Raspberry Pi de tipo TCP.

3.2.5.- Script led.py

El script led.py se recoge en el cuadro 3.3.

```
import time
from neopixel import *
import argparse
import csv
import random
import sys, termios, tty, os, time
# LED strip configuration:
LED_COUNT      = 30    # Number of LED pixels.
LED_PIN1       = 18    # GPIO pin connected to the pixels (18
    uses PWM!).
LED_PIN2       = 13
#LED_PIN       = 10    # GPIO pin connected to the pixels (10
    uses SPI /dev/spidev0.0).
LED_FREQ_HZ    = 800000  # LED signal frequency in hertz (
    usually 800khz)
LED_DMA        = 10    # DMA channel to use for generating
    signal (try 10)
LED_BRIGHTNESS = 122   # Set to 0 for darkest and 255 for
    brightest
LED_INVERT     = False   # True to invert the signal (when
    using NPN transistor level shift)
```

```
LED_CHANNEL1      = 0       # set to '1' for GPIOs 13, 19, 41, 45
    or 53
LED_CHANNEL2      = 1
amarillo = Color(254,255,51)
naranja = Color(125,224,0)
rojo = Color(0,255,0)
apagado = Color(0,0,0)

def parpadeo(strip1, strip2, start, finish, connection):
  estado = 0
  while (estado < 2):
    if (estado == 0):
      for i in range(start,finish):
        strip1.setPixelColor(i,rojo)
        strip2.setPixelColor(i,rojo)
        strip1.show()
        strip2.show()
      time.sleep(1)
      for i in range(start,finish):
        strip1.setPixelColor(i,apagado)
        strip2.setPixelColor(i,apagado)
        strip1.show()
        strip2.show()
      time.sleep(1)
    if (estado == 1):
      for i in range(start,finish):
        strip1.setPixelColor(i,rojo)
        strip2.setPixelColor(i,rojo)
        strip1.show()
        strip2.show()
      time.sleep(0.5)
```

```
        for i in range(start,finish):
          strip1.setPixelColor(i,apagado)
          strip2.setPixelColor(i,apagado)
          strip1.show()
          strip2.show()
        time.sleep(0.5)
      cambio = random.randrange(0,99)
      if (cambio > 65):
        estado += 1
    for i in range(start,finish):
      strip1.setPixelColor(i,rojo)
      strip2.setPixelColor(i,rojo)
      strip1.show()
      strip2.show()
      start_time = time.time()

    data = connection.recv(1000)
    end_time = time.time()
    total_time = end_time - start_time
    print("Tiempo total:",total_time)
    return start_time, end_time

def colores(strip1, strip2, start, finish, connection):
    estado = 0
    while (estado < 2):
      if (estado == 0):
        for i in range(start,finish):
          print("Amarillo")
          strip1.setPixelColor(i,amarillo)
          strip2.setPixelColor(i,amarillo)
          strip1.show()
```

```
            strip2.show()
        time.sleep(0.5)
      if (estado == 1):
        for i in range(start,finish):
          print("Naranja")
          strip1.setPixelColor(i,naranja)
          strip2.setPixelColor(i,naranja)
          strip1.show()
          strip2.show()
        time.sleep(0.5)
      cambio = random.randrange(0,99)
      if (cambio > 65):
        estado += 1
    for i in range(start,finish):
      strip1.setPixelColor(i,rojo)
      strip2.setPixelColor(i,rojo)
      strip1.show()
      strip2.show()
      start_time = time.time()

    data = connection.recv(1000)
    end_time = time.time()
    total_time = end_time - start_time
    print("Tiempo total:",total_time)
    return start_time, end_time

def brillo(strip1, strip2, start, finish, connection):
    estado = 0
    strip1.setBrightness(20)
    strip2.setBrightness(20)
    for i in range(start,finish):
```

```
    strip1.setPixelColor(i,rojo)
    strip2.setPixelColor(i,rojo)
    strip1.show()
    strip2.show()
  while (estado < 2):
    if (estado == 0):
      for i in range(start,finish):
        strip1.setBrightness(20)
        strip2.setBrightness(20)
        strip1.show()
        strip2.show()
      time.sleep(0.5)
    if (estado == 1):
      for i in range(start,finish):
        strip1.setBrightness(50)
        strip2.setBrightness(50)
        strip1.show()
        strip2.show()
      time.sleep(0.5)
    cambio = random.randrange(0,99)
    if (cambio > 65):
      estado += 1
  for i in range(start,finish):
    strip1.setBrightness(150)
    strip2.setBrightness(150)
    strip1.show()
    strip2.show()
    start_time = time.time()

  data = connection.recv(1000)
  end_time = time.time()
```

```
    total_time = end_time - start_time
    print("Tiempo total:",total_time)
    return start_time, end_time

# Main program logic follows:
if __name__ == '__main__':

    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    server_address = ('localhost', 8000)
    print('starting up on {} port {}'.format(*server_address))
    sock.bind(server_address)
    connection, client_address = sock.accept()
    # Create NeoPixel object with appropriate configuration.
    strip1 = Adafruit_NeoPixel(LED_COUNT, LED_PIN1, LED_FREQ_HZ,
        LED_DMA, LED_INVERT, LED_BRIGHTNESS, LED_CHANNEL1)
    strip2 = Adafruit_NeoPixel(LED_COUNT, LED_PIN2, LED_FREQ_HZ,
        LED_DMA, LED_INVERT, LED_BRIGHTNESS, LED_CHANNEL2)
    # Intialize the library (must be called once before other
       functions).
    strip1.begin()
    strip2.begin()
    #Aqui se cambia entre los diferentes modos
    opcion = 0
    with open('data.csv', 'w', newline='') as file:
      writer = csv.writer(file)
      writer.writerow(["Opcion", "Start", "Stop", "Iteration"])
    pruebas_colores = 0
    pruebas_brillo = 0
    pruebas_parpadeo = 0
    print ('Press Ctrl-C to quit.')
```

```
if not args.clear:
  print('Use "-c" argument to clear LEDs on exit')

try:
  while (pruebas_colores < 10 or pruebas_brillo < 10 or
     pruebas_parpadeo <10):
    #time.sleep(1)
    if (opcion == 0 and pruebas_colores < 10):
      print("Colores")
      start_time, end_time = colores(strip1,strip2,0,strip1.
         numPixels(),connection)
      pruebas_colores += 1
      with open('data.csv', 'w', newline='') as file:
        writer.writerow(["Colores", start_time, end_time,
           pruebas_colores])
    elif (opcion == 1 and pruebas_brillo<10):
      print("Brillo")
      start_time, end_time = brillo(strip1,strip2,0,strip1.
         numPixels(),connection)
      pruebas_brillo += 1
      with open('data.csv', 'w', newline='') as file:
        writer.writerow(["Colores", start_time, end_time,
           pruebas_brillo])
    elif (opcion == 2 and pruebas_parpadeo<10):
      print ("Parpadeo")
      start_time, end_time = parpadeo(strip1,strip2,0,strip1
         .numPixels(),connection)
      pruebas_parpadeo += 1
      with open('data.csv', 'w', newline='') as file:
        writer.writerow(["Colores", start_time, end_time,
           pruebas_parpadeo])
```

```
    for i in range (0, strip1.numPixels()):
      strip1.setPixelColor(i, apagado)
      strip2.setPixelColor(i, apagado)
      strip1.show()
      strip2.show()
    connection.close()
  except KeyboardInterrupt:
    for i in range (0, strip1.numPixels()):
      strip1.setPixelColor(i,Color(0,0,0))
      strip1.show()
      strip2.setPixelColor(i,Color(0,0,0))
      strip2.show()
      connection.close()
```

Cuadro 3.3.- Script led.py

Al principio del programa se recogen los datos para la inicialización de las tiras LED. Se diferencian 2 canales PWM, para controlar, si se quisiera, las tiras LED por separado.

A continuación se encuentran 3 funciones: parpadeo, colores y brillo. Las 3 funciones tienen la misma filosofía de programación: presentan 3 estados, que cambian de manera secuencial pasado un tiempo aleatorio; una vez se llegue al último estado, se guarda una marca de tiempo y se pone el socket a la escucha. Una vez se reciba un comando del script volante.py, se guarda la siguiente marca temporal. Las funciones devuelven ambos valores.

El programa principal se encarga de inicializar las tiras LED, abrir el socket de escucha y de llamar a una de las 3 funciones del estudio, cambiando el valor de la variable *opcion*. Los datos son guardados en un archivo csv llamado *data*. El formato del archivo es el siguiente.

Opción	Start Time	Stop Time	Iteración

Una vez se realicen las 10 iteraciones de la opcións señalada, se apagarán las tiras LED y se cerrará el socket.

4. Conclusiones

En este apartado se nombrarán los objetivos de partida de los proyectos y se mostrarán el grado de obtención de los mismos. Asimismo, se explicará, de forma resumida, las razones por las que algunos no han sido completados completamente.

4.1.- Objetivos iniciales del proyecto

En esta sección se explicarán los objetivos del proyecto comercial y del proyecto industrial.

4.1.1.- Proyecto comercial

Los objetivos iniciales del proyecto fueron:

- Desarrollo y construcción de una placa de recogida de datos de: ruido, geolocalización, aceleración, calidad del aire, nivel de luz, temperatura, humedad y presión.
- Obtención de los datos de la unidad ECU del vehículo.
- Integración con la inteligencia desarrollada dentro del grupo de investigación.

4.1.2.- Proyecto industrial

Los objetivos iniciales del proyecto fueron:

- Búsqueda e implementación de una alternativa para la detección de obstáculos.
- Diferenciación de obstáculos
- Búsqueda e implementación de una alternativa para la localización en interiores de los vehículos.

- Búsqueda de una alternativa de interfaz.

- Desarrollo del estudio de investigación de la interfaz.

4.2.- Grado de obtención de los objetivos

En esta sección se hablará del grado de obtención de los objetivos y se explicarán las razones por las que no se han conseguido al completo. De forma general mencionar que la mayor razón por la que el proyecto industrial no ha conseguido la gran mayoría de objetivos cumplidos es por las horas de dedicación a cada proyecto. El proyecto comercial apremiaba más, por lo que se le dedicaron más horas.

4.2.1.- Proyecto comercial

Dentro del proyecto comercial, se ha conseguido la construcción de una placa funcional, mejorando la que se había desarrollado antes. Se le han añadido sensores que proporcionan más datos en comparación con los anteriores y se han cambiado 3 sensores para mejorar la veracidad de los datos obtenidos.

Por otro lado, en lo referido a la obtención de datos de la ECU del vehículo, aún consiguiendo su funcionamiento, no se puede hablar de un grado de obtención del objetivo total ya que se ha tenido que arreglar el código de una forma imperfecta. Tras hacer numerosas pruebas dentro del vehículo, se ha visto que la conexión con la ECU del vehículo tras varios minutos de funcionamiento se cierra. De esta forma, se ha tenido que abrir y cerrar la conexión en cada iteración para conseguir datos de un largo periodo de tiempo, haciendo el programa más pesado y más lento.

Por último, la integración con la inteligencia no se ha conseguido por falta de tiempo. Sin embargo, se ha cambiado la filosofía de ejecución de como estaba al principio del proyecto (varios scripts ejecutándose en paralelo) a la actual (un script ejecutando diferentes hilos). De esta forma, la integración con la inteligencia es más sencilla, puesto que sólo se tendrá que comunicar con un programa y no con varios.

4.2.2.- Proyecto industrial

En el proyecto industrial se han buscado alternativas para la detección de obstáculos y la localización en interiores. Sin embargo, sólo se ha probado de forma muy pobre la detección de obstáculos, sin diferenciar qué tipo de obstáculo era. Todo esto ha sido debido, como se ha aclarado en la introducción, por falta de tiempo.

La interfaz se ha desarrollado y probado de forma completa. El estudio, aunque también está completamente pensado y desarrollado, se necesita pulir a través de simulaciones de ejecución con personal del grupo de investigación.

4.3.- Conclusiones

Durante el desarrollo de estas prácticas se han desarrollado 2 proyectos que, aunque compartiendo varias características, se han afrontado de una forma diferente. Por un lado, el proyecto comercial ha sido una continuación de un proyecto que ya llevaba varios años en desarrollo; por otro lado, el proyecto industrial se ha desarrollado desde cero, teniendo que pensar cada línea de trabajo.

En el proyecto comercial ya existía una placa de sensores funcional que, a pesar de tener sus limitaciones, conseguía recolectar los datos. Al saber qué datos se querían recoger, agilizó bastante el desarrollo de una nueva placa. El cambio introducido en esta placa ha sido en diferentes aspectos:

- Se ha optado por una alternativa GPS más barata que cumple, ahorrando aproximadamente 40 euros.

- Se ha cambiado el acelerómetro de uno que devolvía valores de -256 a +256, pudiendo funcionar en 3 formatos, +-2G, +-4G y +-8G, a otro que devuelve los valores de las aceleraciones en metros por segundo cuadrado, además de proporcionar valores del giroscopio, que pueden ser usados para saber si el vehículo está subiendo o bajando una cuesta, por ejemplo.

- Los sensores de nivel de luz y de calidad de aire se han cambiado de unas alternativas analógicas que necesitaban de una calibración previa, a unas alternativas digitales que no necesitan de esta calibración previa.
- El sensor de temperatura, humedad y presión se ha cambiado de un sensor que sólo daba temperatura y humedad, de un precio similar.

En el proyecto industrial, al partir de 0, no se tenía una referencia previa, con lo que sólo se conocían las peticiones de las empresas con las que se contactó. El desarrollo del prototipo se vio atrasado por el estudio que se pretendía hacer ante la posibilidad de sacar un artículo sobre ello. Cuando se tenía el estudio a punto de empezar a hacer pruebas, debido a la crisis sanitaria, se paralizó completamente este proyecto, avanzando de una forma limitada en todos los aspectos.

5. Líneas de trabajo futuras

En este capítulo se hablarán de las líneas de investigación futuras para ambos proyectos.

5.1.- Parte comercial

Como trabajo futuro de la parte comercial se incluirían los siguientes puntos:

- Cambio en el OBD: por comodidad, sería deseable el cambio del OBD link utilizado por una alternativa inalámbrica, tanto Wi-Fi como Bluetooth. Las alternativas probadas hasta ahora no han dado resultados satisfactorios en ningún caso, pero se podrían probar alternativas más caras.
- Integración con la inteligencia: dentro del grupo de está desarrollando una inteligencia que interprete los datos recogidos por el prototipo y de recomendaciones personalizadas a cada conductor con el fin de aumentar la eficiencia y dar más seguridad.
- Estudio de una alternativa digital del sensor de sonido: sería interesante el cambio del sensor de sonido por una alternativa digital para simplificar el sistema y la recolección de datos.

5.2.- Parte industrial

Como trabajo futuro de la parte industrial se debería:

- Realizar el estudio que se ha planteado: a causa de la emergencia sanitaria que está sucediendo no se ha podido realizar el estudio planteado.
- Prueba del sistema de detección de obstáculos: se deberían realizar pruebas y diseños más detallados del sistema de detección de obstáculos.

- Diseño y pruebas del sistema de localización en interiores: se tendrá que realizar un diseño detallado del sistema de localización en interiores y unas pruebas de campo para comprobar su eficiencia.

6. Glosario

En este capítulo se explicarán algunos conceptos de los que se hablan en el documento y no se explican al completo.

- Intervalo R-R: intervalo de tiempo, en ms, que pasa entre picos del complejo QRS de la señal cardíaca. En la Figura 6.1.

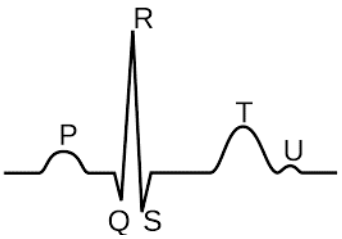

Figura 6.1.- Complejo QRS de la señal cardíaca

- SPI:es un estándar de comunicaciones, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos.
- TVOC: compuestos orgánicos volátiles totales.
- I2C: puerto y protocolo de comunicación serial, define tanto las tramas enviadas como la conexión física entre 2 dispositivos digitales.
- Script: programa simple.
- *Fuel Status*: mostrará si el sistema se encuentra en bucle abierto o cerrado, además de si tiene algún fallo en el bucle debido a un fallo en el sistema (OL-Fault, Fallo en Bucle Abierto) o si algún sensor de Oxígeno tiene un fallo y se encuentra en bucle cerrado (CL-Fault).
- *Engine Load*: se calcula mediante la siguiente ecuación:

$$LOADPCT = \frac{currentairflow}{(peakairflowatWOT@STPasafunctionofrpm)*(BARO/29{,}92)*SQRT(298/(AAT+273))} \tag{6.1}$$

Donde STP es la temperatura y la presión estándar (25ºC y 29.92 en Hg Baro); SQRT es raíz cuadrada; WOT es el acelerador completamente abierto para que el aire y combustible estén en su máximo; y AAT es la Temperatura Ambiente del Aire (en ºC).

- *Fuel Pressure*: presión del combustible en las manguito de alta presión.
- *Intake Pressure*: presión total tomada en la entrada del motor como medida de la densidad del aire.
- *RPM*: Revoluciones Por Minuto.
- *Timing Advance*: indica si el árbol de levas de admisión del banco 1 está más avanzado de lo que la centralita del coche ha ordenado que esté.
- *MAF*: indica si hay un problema con el circuito del sensor de flujo de masa de aire.
- *Throttle Position*: posición del pedal de aceleración, en porcentaje.
- *Fuel Rail Pressure Vac*: presión que tiene el combustible en los inyectores.
- *Absolute Load*: este parámetro es el valor normalizado de la masa de aire por carrera de admisión, en porcentaje.
- *Relative Throttle Pos B*: cantidad de apertura del acelerador.
- *Throttle Pos B, C*: en el acelerador del vehículo se encuentran 2 potenciómetros conectados para que la ECU verifique la integridad de los datos.
- *Accelerator Pos D*: mínima posición del acelerador, alrededor del 5
- *Accelerator Pos E*: máxima posición del acelerador, alrededor del 95
- *Throttle Actuator*: devuelve un código de error en caso de fallo del acelerador.
- *Fuel Inject Timing*: devuelce un código de error en caso de fallo en el tiempo de inyección del combustible.
- *Fuel Rate*: consumo de combustible en litros por hora.

- Sensor de ultrasonidos: sensor que dispone de 1 altavoz y 1 micrófono. El altavoz emite un sonido en frecuencia de ultrasonidos y, una vez rebote en alguna superficie, se refleja y es captado por el micrófono.
- LED: fuente de luz constituida por 1 semiconductor y 2 terminales.

Bibliografía

[1] Python 3.7 Documentation *Documentación detallada de Python 3.7* https://docs.python.org/3/

[2] SQL Documentation *Documentación detallada de sql* https://www.w3schools.com/sql/

[3] Página oficial de Matlab *Documentación detallada de Matlab* https://www.mathworks.com/products/matlab.html

[4] Página oficial de Raspberry *Descarga y documentación de contenido relacionado con Raspberry Pi* https://www.raspberrypi.org

[5] Página oficial del framework OBD de python *Tutoriales y ayuda para el desarrollo de aplicaciones en Python usando el framework OBD* https://python-obd.readthedocs.io/en/latest/

[6] Página oficial de Raspberry *Descarga y documentación de contenido relacionado con Raspberry Pi* https://www.raspberrypi.org

[7] Página oficial de Mouser España *Documentación e imágenes detalladas de componentes electrónicos* https://www.mouser.es/

[8] Tutorial de uso del GPS con raspberry Pi *GPS GY-NEO6MV2 en la Raspberry Pi* http://carlini.es/gps-gy-neo6mv2-en-la-raspberry-pi/

[9] Página oficial de Eagle https://www.autodesk.com/products/eagle/overview

[10] Framework de BME280 para Python *RPi.bme280 0.2.3* https://pypi.org/project/RPi.bme280/

[11] Framework de BH1750 para Python https://gist.github.com/oskar456/95c66d564c58361ecf9f

[12] Tutorial del CCS811 para Raspberry Pi *Adafruit ccs811 air quality sensor* https://learn.adafruit.com/adafruit-ccs811-air-quality-sensor/python-circuitpython

[13] Tutorial del LSM9DS1 para Python *Adafruit LSM9DS1 Accelerometer + Gyro + Magnetometer 9-DOF Breakout* https://learn.adafruit.com/adafruit-lsm9ds1-accelerometer-plus-gyro-plus-magnetometer-9-dof-breakout/python-circuitpython

[14] Tutorial del uso de tiras LED con Raspberry Pi *NeoPixels on Raspberry Pi* https://learn.adafruit.com/neopixels-on-raspberry-pi

[15] OBD-II Resource *Interpreting Generic OBD-II Scan Data* http://obdcon.sourceforge.net/2010/06/interpreting-generic-obd-ii-scan-data/

[16] Motor Vehicle Maintenance and Repair *How is engine load determined?* https://mechanics.stackexchange.com/questions/17537/how-is-engine-load-determined

[17] Código OBDII P0101 - Sensor MAF *Código OBDII P0101 - Sensor MAF* https://www.e-auto.com.mx/enew/index.php/85-boletines-tecnicos/6802-codigo-obdii-p0101-sensor-maf

[18] Team-BHP *OBD2 Parameter IDs (PIDs) and what they mean?* https://www.team-bhp.com/forum/technical-stuff/125767-obd2-parameter-ids-pids-what-they-mean.html

[19] OBD-Codes *P2119 Throttle Actuator Control Throttle Body Range* https://www.obd-codes.com/p2119

[20] Wikipedia *OBD-II PIDs* https://en.wikipedia.org/wiki/OBD-II_PIDs

[21] Hetpro *I2C – Puerto, Introducción, trama y protocolo* https://hetpro-store.com/TUTORIALES/i2c/

[22] Wikipedia *Led* https://es.wikipedia.org/wiki/Led

[23] Wikipedia *Serial Peripheral Interface* https://es.wikipedia.org/wiki/Serial_Peripheral_Interface