DEPARTAMENTO DE INGENIERÍA E INVESTIGACIONES TECNOLÓGICAS

Sistemas Operativos Avanzados

Internet of Things (IOT)

Sistema embebido Arduino y aplicación mobile Android

Segundo Cuatrimestre - Año 2017

Proyecto: Car Rent Tracking

Integrantes:

Tonlorenzi, Luciano	DNI: 39.244.171
Ratibel, Pablo	DNI: 37.792.037
Gambacorta, Mariano	DNI: 29.279.015
Pin Etchave, Eleazar	DNI: 35.532.456
Avalo, Pablo	DNI: 39.244.171
Dal Borgo, Gabriel	DNI: 35.944.975

CONTENIDO

Objetivo del TP

Aplicar conocimientos obtenidos sobre IOT, Android y Sistemas Embebidos en el desarrollo de un sistema que realice un seguimiento de las condiciones de manejo de un auto rentado midiendo aspectos como la velocidad y el nivel de humo dentro del vehículo.

Descripción del Entorno

Hardware

Sistema Embebido

- 1 Módulo LDR
- 1 Módulo Shield SD
- 1 Módulo Bluetooth HC-06
- 1 Sensor de gases MQ7
- 1 Modulo GPS GY-GPS6MV2
- 1 Potenciómetro de 10k
- 1 Cooler
- 1 Transistor TIP 122
- 1 Acelerómetro MPU6050
- Resistencias: 220ohms (x3), 120ohms (x2), 10ohms (x1), 1k ohms (x1)
- 1 Capacitor cerámico 104 (0,1 microfaradios)
- Capacitores electrolíticos: 10 microfaradios x 50v (x2), 220 microfaradios x 25v (x1)
- 1 Diodo 1N4004
- 1 Amplificador LM386D
- 4 LED‘s
- Pineras para conexiones
- 1 Parlante

Aplicación Android

- Acelerómetro
- Sensor de Proximidad

Software

Sistema Embebido

- IDE Arduino
- Libreria TinyGPS.h
- Libreria SD.h
- Libreria SPI.h
- Libreria Audio.h
- Libreria I2Cdev.h
- Libreria MPU6050.h
- Libreria Wire.h

Aplicación Android

- IDE Android Studio
  - Java 8
  - Android SDK 6.0.1
  - Apache Gradle

Alcance del Sistema

Sistema Embebido

1. El sistema embebido debe medir el nivel de monóxido de carbono dentro del vehículo mediante el sensor MQ7.
2. El sistema embebido debe dar aviso ante un exceso de monóxido de carbono a través del parlante, registrarlo en la SD e iniciar la ventilación del vehículo a través del Cooler.
3. El sistema embebido debe registrar la velocidad y ubicación del vehículo a través del módulo GPS.
4. El sistema embebido debe dar alerta a través del parlante ante un exceso de velocidad, registrando en la SD el máximo valor alcanzado.

5. El sistema embebido debe verificar el nivel de luminosidad en el rango horario preseteado a través del módulo LDR
6. El sistema embebido debe dar alerta de luces apagadas, cuando el sistema detecta baja luminosidad en la óptica del vehículo durante horario nocturno.
7. El sistema embebido debe dar alerta de modificación parámetros del sistema.
8. El sistema debe comunicarse con la aplicación Android a través del Bluetooth.
9. El sistema debe setear la velocidad máxima permitida con los parámetros recibidos desde la aplicación Android.
10. El sistema debe setear el nivel límite de monóxido de carbono permitida con los parámetros recibidos desde la aplicación Android.
11. El sistema embebido debe permitir eliminar los datos de navegación.
12. El sistema embebido debe auditar las acciones de configuración de parámetros.
13. El sistema debe permitir consultar los parámetros funcionales
14. El sistema debe permitir transferir los archivos de Tracking y Eventos a través del Bluetooth en respuesta a mensajes recibidos desde la aplicación Android.

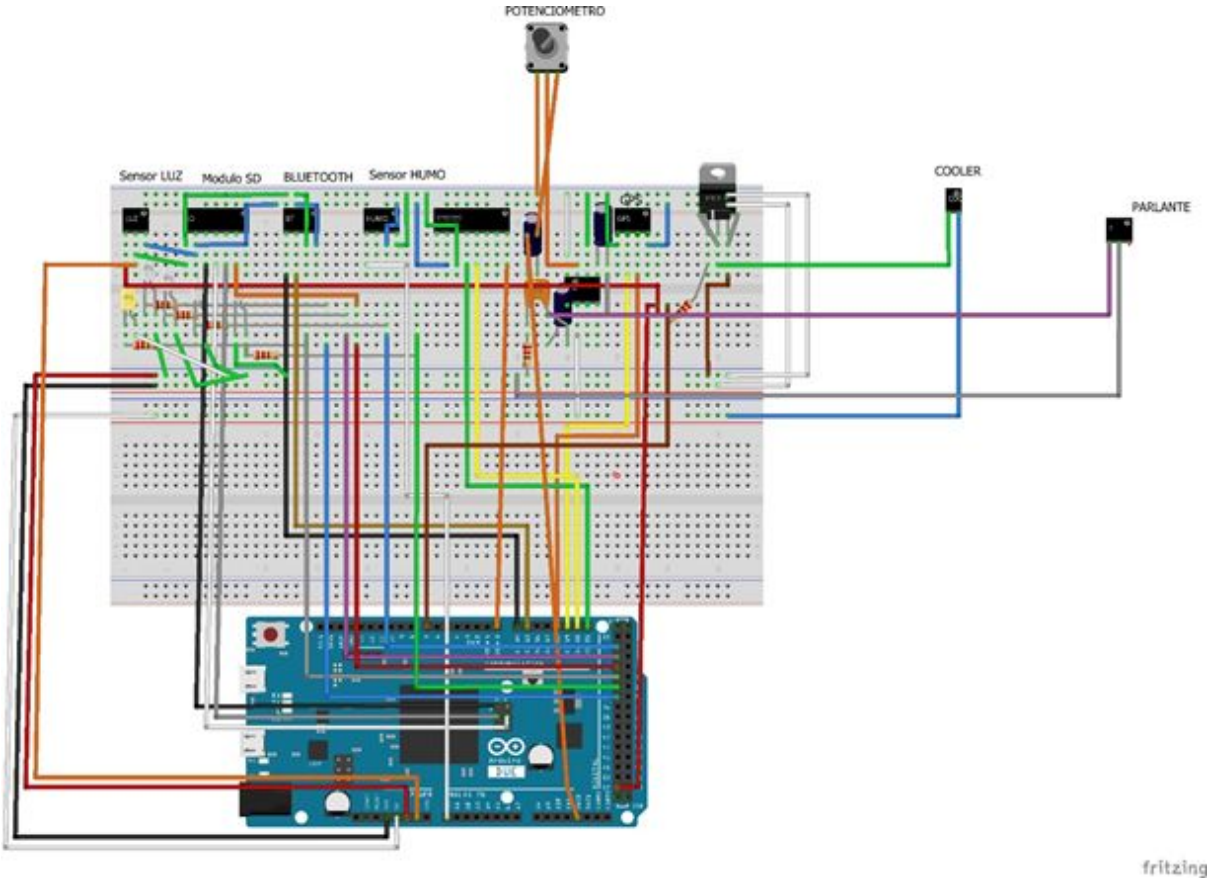
Aplicación Android

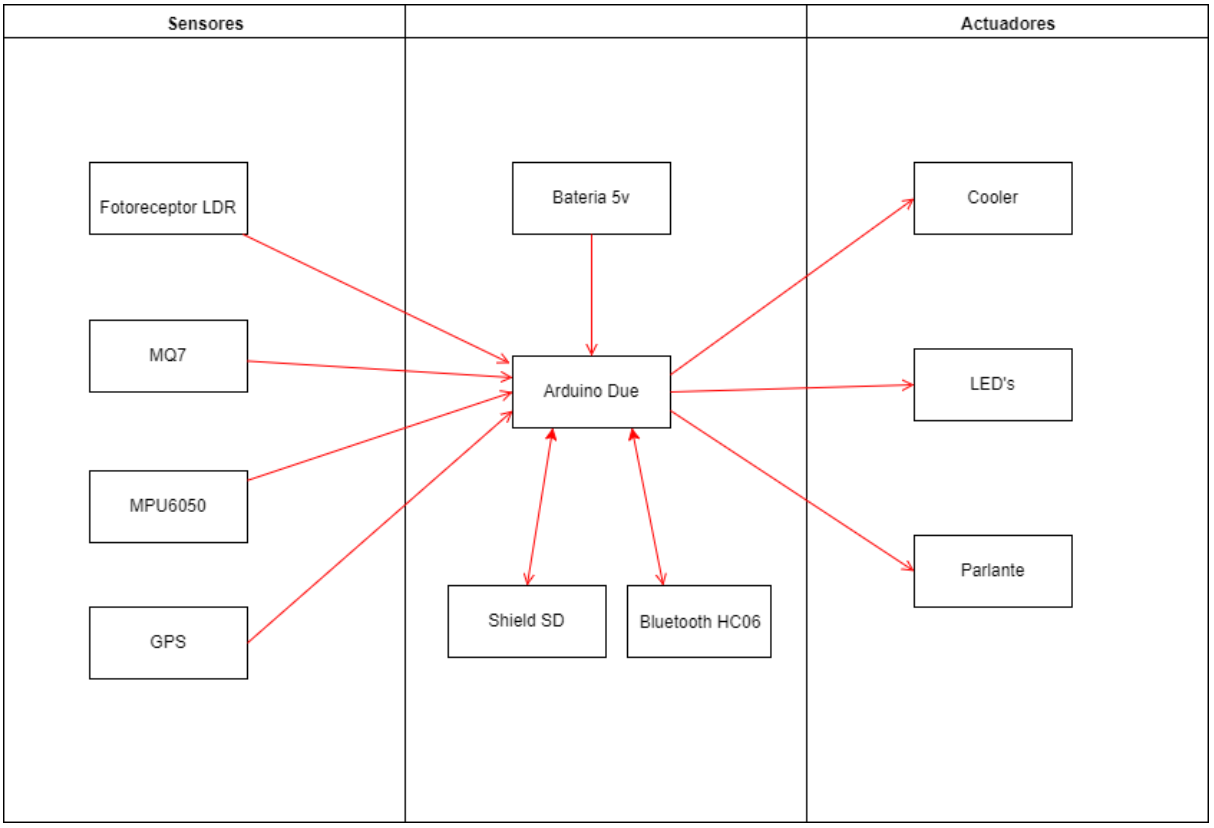
1. La aplicación Android debe comunicarse vía bluetooth con el sistema embebido para:
    a Configurar parámetros funcionales del sistema
    b Consultar parámetros funcionales del sistema
    c Obtener registros de geolocalización e historial de eventos.
2. La aplicación Android debe permitir definir el tipo de neumático del vehículo.
3. La aplicación Android debe enviar el tipo de neumático al Arduino luego de realizar un shake.
4. La aplicación Android debe permitir configurar el nivel límite de monóxido de carbono en habitáculo.
5. La aplicación Android debe informar el historial de navegación y los eventos capturados por el SE, durante el alquiler del vehículo.
6. La aplicación Android debe brindar la opción de reinicio de los registros de historial del sistema embebido.

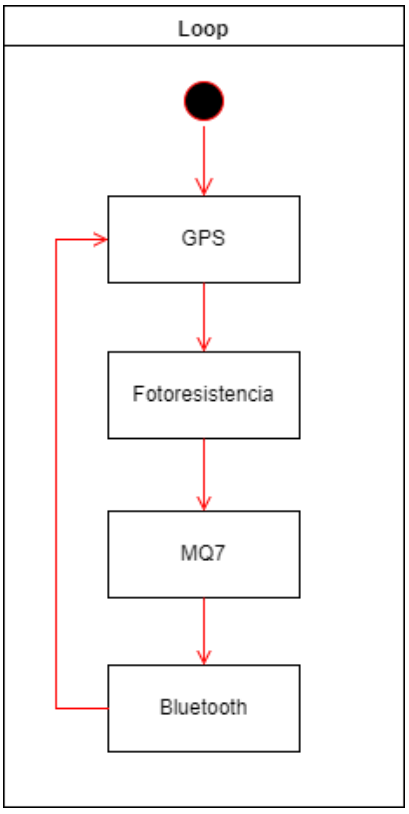
7. La aplicación Android debe implementar el sensor de Proximidad del celular para confirmar la operación de reinicio de los registros de historial del sistema embebido.

Diseño

Previo a meter mano en la placa, sensores, actuadores y demás decidimos utilizar algunos de los simuladores online recomendados por la cátedra.
A través de estos logramos toma conocimientos básicos de electrónica, los cuales nos empezaron a dar una idea de cómo realizar las conexiones necesarias para que nuestro Sistema Embebido vaya tomando forma.
Diseño del sistema implementado, utilizando la herramienta fritzing.




Sensores
Actuadores
Fotoreceptor LDR
MQ7
MPU6050
GPS
Bateria 5v
Arduino Due
Shield SD
Bluetooth HC06
Cooler
LED's
Parlante



Sensores

Módulo GPS

El módulo GPS es de vital importancia para nuestro Proyecto. A través del mismo
Obtenemos distintos parámetros como la hora (GMT), latitud, longitud y velocidad.
Así podemos trazar un seguimiento del vehículo, pudiendo detectar todas las ocasiones
En las cuales el auto superó el límite de velocidad establecido.
Para su conexión utilizamos el Serial1.

Caracteristicas:

- Protocolo: UART (asincrónica)
- Voltaje: 3.3v
- Pines: Tx, Rx, Gnd, Vcc
- Formato: bigendian

Módulo LDR

A través de este fotoresistor medimos la cantidad de luz que recibe el vehículo pudiendo
Detectar si se está conduciendo en un ambiente nocturno o diurno y así controlar las
Luces del auto.
El LDR varía su resistencia con la luz que recibe, por otro lado su valor resistivo también
Se ve afectado con la luz infrarroja y ultravioleta.

Este módulo posee una salida digital y utiliza un potenciómetro para regular la Sensibilidad de esta salida. Debe ser alimentado desde 3.3V.
Un fotorresistor está hecho de un semiconductor de alta resistencia como el sulfuro de Cadmio, CdS. Las células de sulfuro del cadmio se basan en la capacidad del cadmio de Variar su resistencia según la cantidad de luz que incide en la célula. Cuanta más luz Incide, más baja es la resistencia
La variación del valor de la resistencia tiene cierto retardo, diferente si se pasa de oscuro A iluminado o de iluminado a oscuro. Es un límite si se quiere medir cambios rápidos de Luminosidad.

Caracteristicas:

- Digital
- Voltaje: 3.3v
- Pines: DO, Gnd, Vcc

Sensor MQ7

El sensor MQ7 nos permite medir la cantidad de CO que hay dentro del vehículo. Gracias A esto podemos detectar si hay un excedente en el nivel de humo permitido en habitáculo Para luego activar el Cooler.
Los Sensores MQ son electroquímicos y varían su resistencia al exponerse a Determinados gases. Posee un calentador encargado de aumentar la temperatura interna Y con esto el sensor pueda reaccionar con los gases.
Debido al calentador es necesario esperar un tiempo de calentamiento para que la salida Sea estable.
En nuestro proyecto utilizamos el MQ7 ya que este sensor es de alta sensibilidad al Monóxido de carbono (CO), aunque también es sensible al H2.
Para su conexión necesitamos alimentarlo con 5V y conectar la salida analógica del Módulo a una entrada analógica del Arduino.

Caracteristicas:

- Protocolo: “
- Voltaje: 3.3v
- Pines: A0, Gnd, Vcc

Acelerómetro MPU 6050

Utilizamos el acelerómetro para confirmar el desplazamiento del dispositivo y asi descartar datos espurios provocada por interferencias en el sensor GPS.

Caracteristicas:

- Protocolo: I2C (sincrónica)
- Voltaje: 3.3v
- Pines: SCL, SDA, Vcc, Gnd, Int
- Formato: bigendian

Actuadores

Cooler

Utilizamos un cooler para ventilar el vehículo en caso de que se detecte un nivel de humo superior al permitido.
Salida: Analogica

Parlante

La principal función es la de dar alerta ante distintos eventos (Velocidad, humo).
Salida: Analogica.

LED

Se utiliza un LED en el caso de que el LDR recibe poca luz. Además se usan otros LED para cuando se disparan las alertas.
Salida: Digital.

Módulo Micro SD

Se utiliza un Shield SD para poder registrar el seguimiento y las distintas alertas generadas. La comunicación de la memoria es por SPI. Se usa el pin 28 para el CS.

Caracteristicas:

- Protocolo: SPI (sincrónico)
- Voltaje: 3.3v
- Pines: Mosi, Miso, Sck
- Formato: bigendian

Implementación

Sistema Embebido

Implementar GPS

Se inicia el serial del GPS (Serial.Begin).
Se instancia un objeto TinyGps de la biblioteca homónima. Mediante el mismo podemos manipular los datos recibidos por el GPS de una manera más sencilla.

Implementar LDR

Se setea al pin del LDR como un INPUT y al pin del LED como output. Se realiza un DigitalRead al pin de LDR y si el valor es 1 (no se detectó luz) se setea el valor del LED en HIGH.

Implementar MQ7

Se setea el pin analógico del MQ7 como un INPUT. Se lee la salida analógica del sensor y se la convierte a voltaje. Si el voltaje leído supera el límite establecido se debe registrar el evento en la SD, reproducir la alerta en el parlante e iniciar el funcionamiento del cooler.

Implementar MPU6050

Haciendo uso de las librerías Wire y MPU6050, que implementan el protocolo de comunicación I2C, se accede al valor de la aceleración instantánea que posteriormente se utilizará para el cálculo de la aceleración lineal.

Implementar Cooler

Se setea el pin del cooler como output (es una salida analógica por PWM). Al momento que se detecta un exceso en monóxido de carbono se realiza una escritura analogica sobre el pin del cooler enviandole un valor que representará a la velocidad del mismo.

Implementar Parlante

Se utiliza la librería Audio de Arduino para poder reproducir los archivos .wav que están guardados en la SD.
Se utilizan
Audio.Begin() : inicializa la biblioteca.
Audio.Prepare() : prepara las muestras de audio desde un archivo a un buffer y setea volumen.
Audio.Write() : escribe la señal de audio desde un buffer a la salida analogica.

Implementar Módulo Micro SD

Se utiliza la biblioteca SD que permite leer y escribir en la memoria SD.
Esta biblioteca soporta FAT 16 y FAT 32 File System.

La clase SD permite acceder y manipular los archivos de la tarjeta.
SD.Begin() : inicializa la biblioteca.
SD.Open() : permite abrir un archivo.

La clase File permite leer y escribir los archivos.

Aplicación Android

Creación de Activities

Activity Main()

Es la activity que se abre al iniciar la aplicación. Esta actividad inicia el Servicio de Bluetooth BTService y es donde se buscan los dispositivos para conectarse.
Esta activity tiene un BroadcastReciever definido, para capturar los eventos de Bluetooth. (mReciever)
La búsqueda se inicia mediante el boton (Buscar Dispositivos) y cuando finaliza captura el evento: BluetoothAdapter.ACTION_DISCOVERY_FINISHED
Si se encontraron dispositivos, se inicia la activity DeviceListActivity()

DeviceListActivity()
Por medio de un Bundle del Intent, se obtiene la lista de dispositivos encontrados y se muestra en un ListView.
Se define un BroadcastReciever donde se obtiene el estado de cada dispositivo (Emparejado, No emparejado)
El boton listenerBotonEmparejar tiene un listener, que realiza la conexion bluetooth del dispositivo seleccionado y inicia la activity NavigationDrawer()

NavigationDrawer()
Al iniciar esta activity, se crea por defecto el fragment BluetoothFragment, con los datos del dispositivo conectado.
Al abrir el panel lateral, se ejecuta onNavigationItemSelected donde se encuentran todos los fragment:

BluetoothFragment()
Fragment inicial que se muestra al conectarse con el dispositivo bluetooth cuya finalidad es totalmente informativa. En ella se puede visualizar el nombre y la dirección del dispositivo con el cual se esta conectado.

LimiteHumoFragment()
Utiliza a los métodos setMQ7 o getMQ7 de la clase BTService para enviar y recibir datos de arduino y así obtener el límite de humo registrado o setear un nuevo límite.

TrackingFragment()
Solicita los registros de tracking al arduino a través del método getFileTracking y muestra todas las mediciones.

EventosFragment()
Solicita los registros de tracking al arduino a través del método getFileEventos y muestra todos los eventos capturados..

NeumaticosFragment()
Utiliza los métodos getVel y setVel de la clase BTService para enviar y recibir datos de arduino y así obtener la velocidad máxima registrada o setear una nueva velocidad máxima.

ResetFragment()
Utiliza a los métodos clearEventos y clearTracking de la clase BTService para limpiar los registros de arduino.

Implementar Sensor de Proximidad
Se utiliza en en fragment ResetFragment para limpiar los registros cuando se detecta proximidad.

Implementar Shake
Se utiliza en en fragment LimiteHumoFragment enviar los límites nuevos de velocidad a arduino cuando se detecta un shake.

Implementar Servicio Bluetooth
BTService()
Servicio que realiza la conexión Bluetooth y gestiona la comunicación con arduino. Tiene todos los métodos para enviar y recibir datos.

BTActivity()
Se utiliza para hacer un Bind y poder llamar a los métodos de la clase BTService con el objeto desde los fragment.

Comunicación

Bluetooth HC-06

Realizamos la comunicación entre el dispositivo android y la placa a través de Bluetooth. Para ello incorporamos un HC-06 (Esclavo) al Arduino.

Desde el arduino al android transferimos los archivos de tracking y de eventos que se encuentran almacenados en la SD.

Por otro lado desde el dispositivo Android se envían al arduino datos de configuración y mensajes para la activación del parlante, de las luces o para limpiar la SD.

La conexión al arduino la realizamos a través del Serial3 . Luego configuramos el nombre , la velocidad de transmisión y la clave de pairing mediante comandos AT.

Dentro de la aplicación Android los objetos necesarios para manejar el bluetooth son los siguientes.

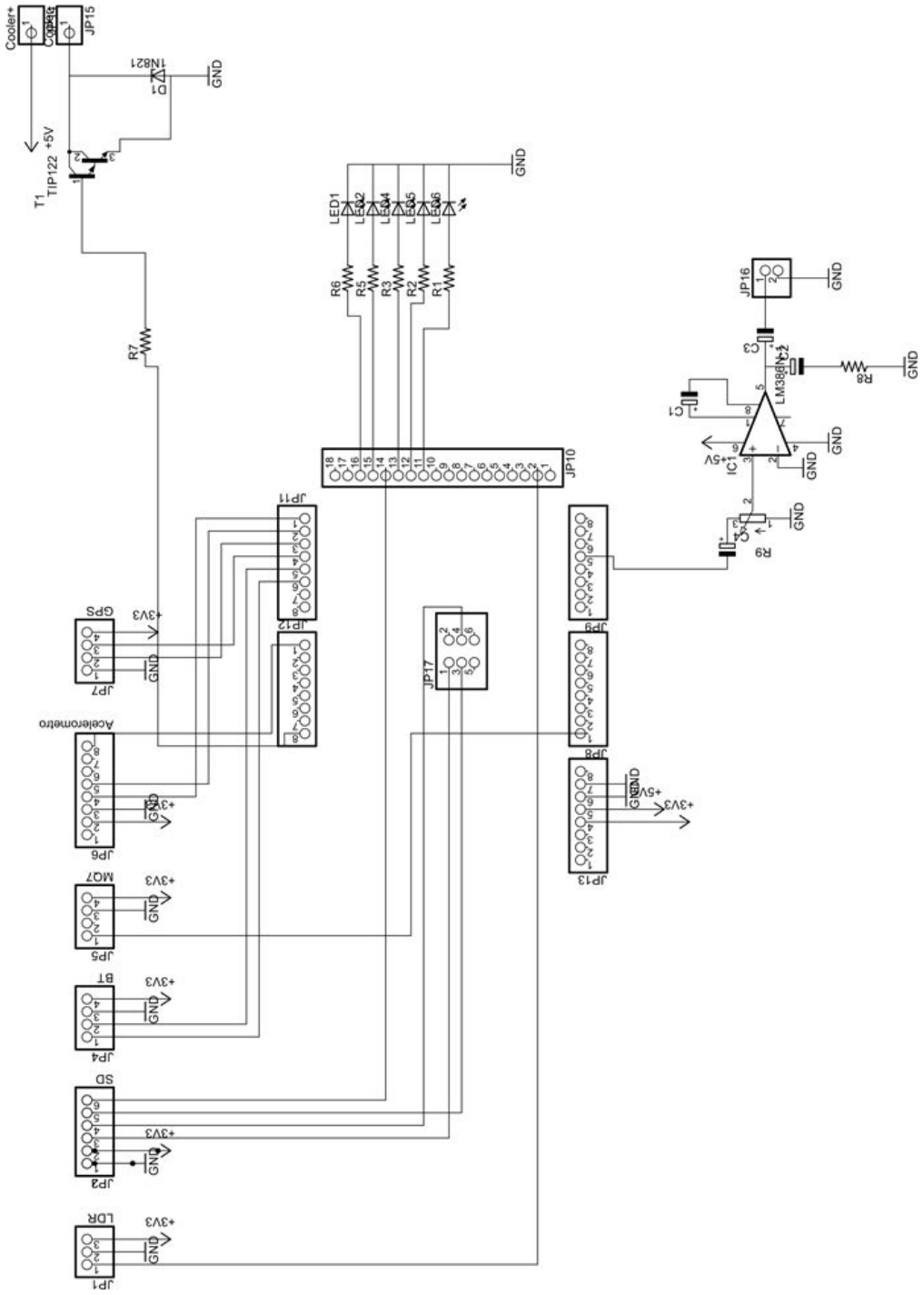
BluetoothAdapter : permite realizar todo tipo de operación sobre el bluetooth del android.
BluetoothDevice : Representa al Bluetooth del Arduino con sus atributos . Se lo usa para el emparejamiento.
BluetoothSocket : Permite la transferencia de datos entre dispositivos.

Protocolo: UART (Asincronico)

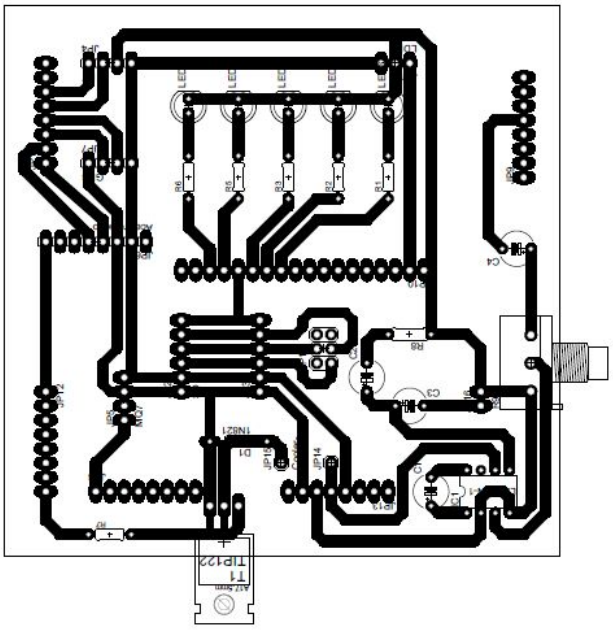
Producto Terminado

Sistema Embebido

Esquemático



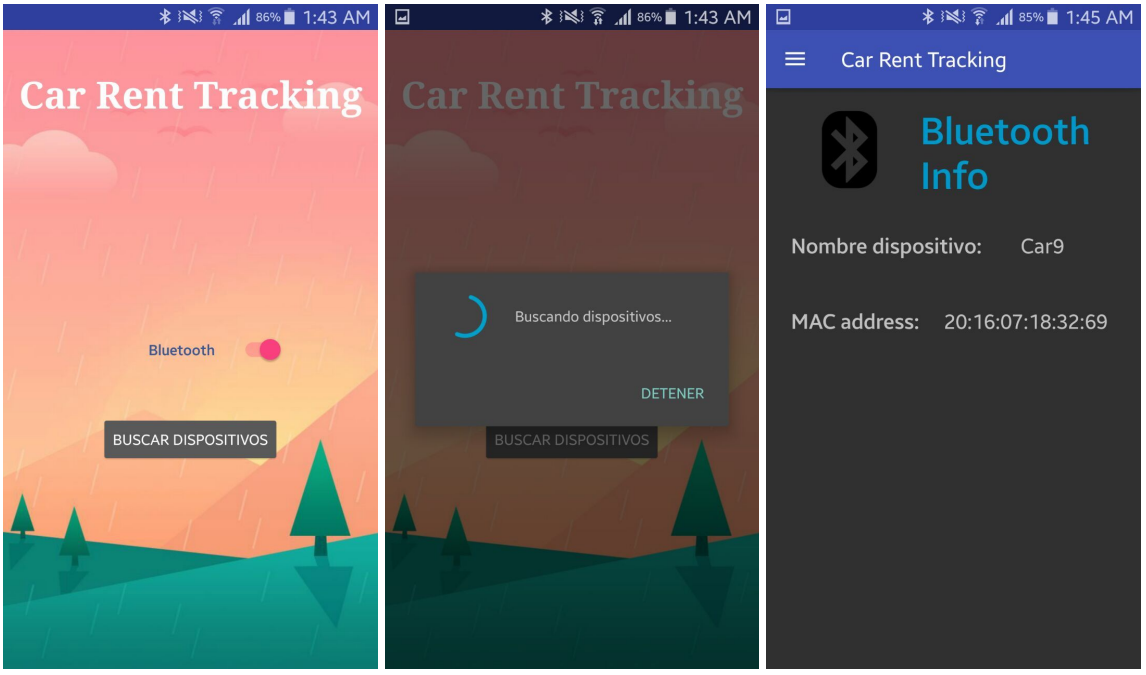
Placa con componentes

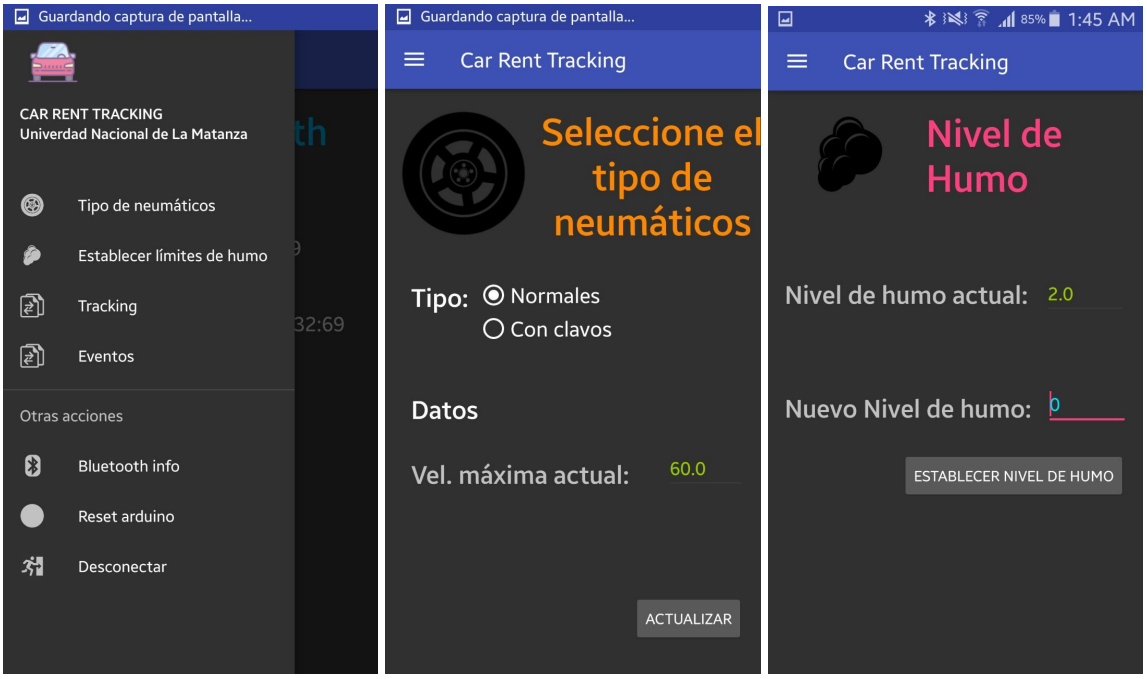


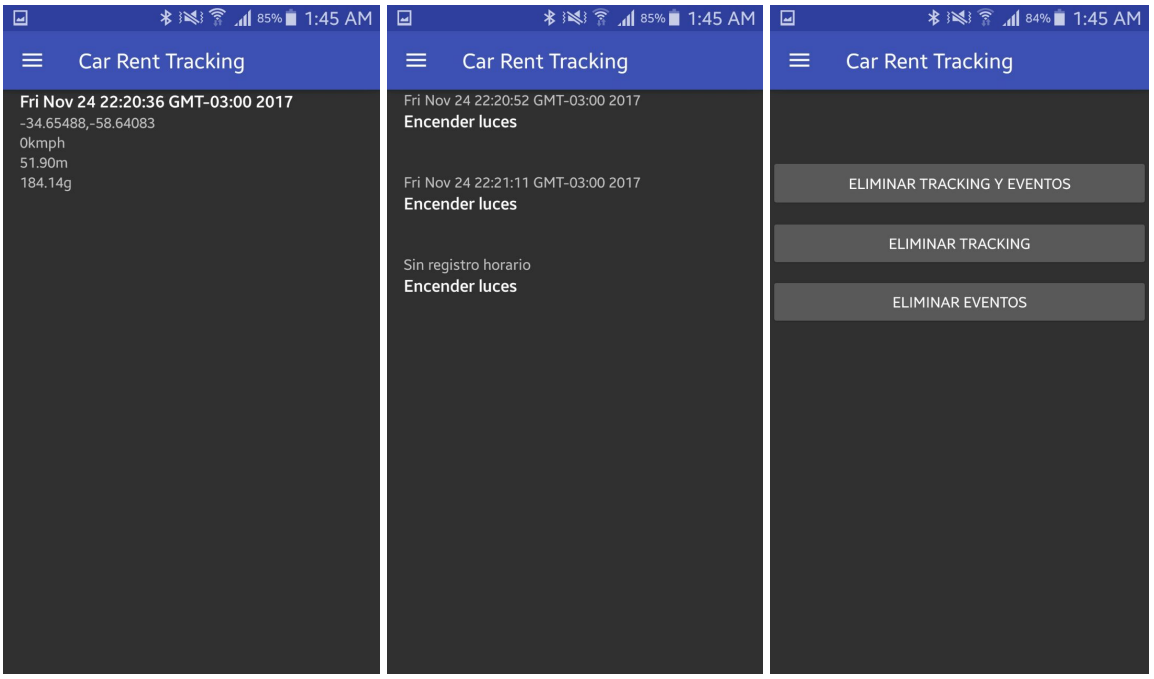
Producto empaquetado

Aplicación Android

Pantallas







Problemáticas

Durante el desarrollo del proyecto nos encontramos con diversas dificultades relacionadas con la falta de experiencia en sistemas embebidos por parte de todos los integrantes del grupo.
Por otro lado, las mediciones efectuadas sobre el gps, el acelerómetro, etc nos brindaban datos espurios por lo cual tuvimos que aprender a salvar/filtrar estos inconvenientes para intentar consolidar la información de la mejor forma posible.
También, hemos tenido serios problemas con el montaje del prototipo sobre una placa física y posterior ensamblado en su packaging final.
Por otro lado, en la aplicación android, tuvimos un retraso mayor a lo esperado debido a problemáticas con el establecimiento de la conexión bluetooth. Lo cual derivó en cambiar en varias ocasiones el módulo HC06

Links de Interés

Código arduino: https://github.com/EleazarPin/CarRentTracking-Arduino
Código android: https://github.com/EleazarPin/CarRentTracking-Android
Informe: https://github.com/EleazarPin/CarRentTracking-Informe