UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escuela Técnica Superior de Ingeniería Informática

Universitat Politècnica de València

CONTROL DE UN COCHE ELECTRICO CON UN SISTEMA DE VISION A TRAVES DE ARDUINO

Proyecto Final de Carrera

Grado en Ingeniería Informática

Autor: Alberto Martinez Angulo

Director: Floreal Acebrón Linuesa

2016-2017

En primer lugar quería agradecer a mi tutor Floreal por su ayuda e implicación en el proyecto, a mi mujer por su paciencia y apoyo en todos esos momentos difíciles que las cosas no te salen y para terminar quiero mencionar también a toda esa cantidad de gente desconocida y totalmente altruista que deja sus aportaciones y su trabajo en los foros, blogs, webs y videos que en ocasiones nos sacan de un buen apuro.

Resumen

El trabajo realizado en el TFG ha consistido en equipar un coche eléctrico con sensores y actuadores que permiten un guiado remoto, sin la necesidad de visión directa sobre el coche a través de un dispositivo con sistema operativo Android, desde donde podemos ver un video en directo, mover la cámara, encender o apagar las luces, dependiendo de la necesidad del entorno, así como ver en tiempo real la distancia en centímetros hasta los objetos que apuntemos con la cámara.

Palabras clave: Coche, Arduino, Raspberry, Unity, Raspbian, Python, C#, Camara USB, WIFI, Circuito Integrado.

Abstract

The aim of this TFG is to fit an electric car with sensors that allow a distant guidance without needing a direct vision on the car through a device with Android operating system, where we can see a live video, move our camera, turn on or turn off the lights depending on the environment necessity as well as seeing in real time the distance on centimetres to the objects we aim with the camera.

Keywords: Car, Arduino, Raspberry, Unity, Raspbian, Python, C#, USB Camera, WIFI, Integrated Circuit.

Tabla de figuras

Figura 1: Placa Arduino Uno 9
Figura 2: Placa Raspberry Pi 3 10
Figura 3: Cámara USB 11
Figura 4: Protoboard 11
Figura 5: Conexionado Protoboard 11
Figura 6: Sensor de distancia HC-SR04 12
Figura 7: Circuito integrado L293D 12
Figura 8: Conexionado de un puente en H 12
Figura 9: Regulador de tensión L7805K 13
Figura 10: Transistor BC547 13
Figura 11: Logo Raspbian 14
Figura 12: Logo Arduino 15
Figura 13: Captura IDE de Arduino 16
Figura 14: Logo de Unity 16
Figura 15: Pantalla de tablas ip 20
Figura 16: Captura dhcpd.conf 21
Figura 17: Captura sysctl.conf 23
Figura 18: Captura contenido iptables 23
Figura 19: Captura hacer propietario a root 24
Figura 20: Captura cambio adaptador WiFi 1 25
Figura 21: Captura cambio adaptador WiFi 2 25
Figura 22: Captura estado servicio hostapd 26
Figura 23: Captura estado servicio isc-dhcp-server 26
Figura 24: Interfaz Gráfica 31
Figura 25: Diagrama de flujo 32
Figura 26: Esquema eléctrico del coche 33
Figura 27: Cableado interno del coche 33

Acrónimos

USB	Universal Serial Bus
SD	Secure digital
RAM	Random Access Memory
DDR	Double Data Rate
B	Byte
KB	Kilobyte
MB	Megabyte
GB	Gigabyte
MHz	Megahercio
GHz	Gigahercio
IDE	Integrated Development
SO	Sistema Operativo
SDK	Software Development Kit
QEMU	Quick Emulator
VNC	Virtual Network Computing
SSH	Secure Shell

Tabla de contenidos

1. Objetivo 8
2. Hardware 9
2.1 Arduino 9
2.2 Raspberry 10
2.3 Camara USB 11
2.4 Componentes Electronicos 11
3. Software 14
3.1 Raspbian 14
3.2 Software Arduino 15
3.3 Unity 16
3.4 SDK Android 17
3.5 VNC Server 17
3.6 Putty 18
4. Desarrollo 18
4.1 Lenguajes de programación 18
4.1.1. Python 18
4.1.2. C# 18
4.1.3. C 19
4.2 Red WIFI 19
4.3 Servidores 27
4.3.1. Servidor de Control 27
4.3.2. Servidor de Imagen 28
4.4 Aplicación 29
4.4.1. Cliente de Control 29
4.4.2. Cliente de Imagen 30
4.4.3. Interfaz Gráfica 31
4.5 Conexionado 32
5. Problemas encontrados 34
6. Conclusiones 35
7. Bibliografia 35

1. OBJETIVO

El objetivo del TFG es equipar un coche eléctrico con sensores que permitan un guiado remoto sin necesidad de visión directa sobre el coche a través de un dispositivo con sistema operativo Android, desde donde podremos ver un video en directo, mover nuestra cámara, encender o apagar las luces dependiendo de la necesidad del entorno así como ver en tiempo real la distancia en centímetros hasta los objetos que apuntemos con la cámara.

2. HARDWARE

El hardware hace referencia a las partes físicas, componentes eléctricos, electromecánicos y mecánicos, cables, periféricos de todo tipo y cualquier otro elemento físico involucrado.

2.1 Arduino

- **¿Qué es Arduino?**

  Arduino es una compañía de hardware libre que diseña y manufactura placas de desarrollo de hardware y software, compuesta por circuitos impresos que integran un microcontrolador y un entorno de desarrollo, donde se programa cada placa.

- **¿Para qué sirve?**

  Las placas Arduino se pueden utilizar para desarrollar todo tipo de proyectos, sistemas autónomos, (robots), control remoto (domótica), incluso puede ser conectado a software para adquisición de datos.

- **¿Modelo utilizado y por qué?**

  El modelo de placa utilizado en el proyecto es el Arduino uno, la razón por la que se eligió este modelo es la gran aceptación que tiene, la cantidad de información que circula sobre él, la compatibilidad y cantidad de componentes de que dispone y muy importante lo económico que es.

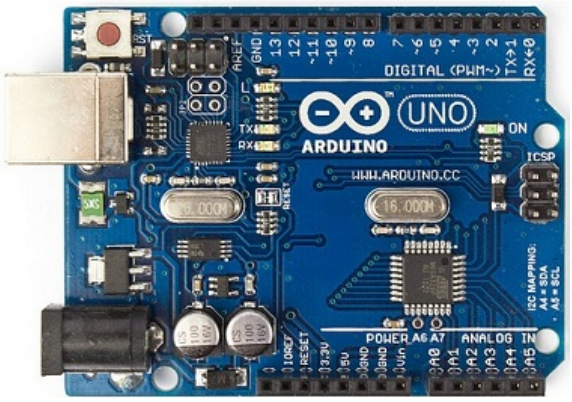

Figura 1: Placa Arduino Uno

2.2 Raspberry

- **¿Qué es Raspberry?**

  La Raspberry pi es un ordenador de placa única o de placa simple (SBC) desarrollado en Reino Unido por la Fundación Raspberry pi, con el objetivo de estimular la enseñanza de ciencias de la computación en las escuelas. Se trata de un producto de propiedad registrada pero de uso libre, la compañía aún no deja claro si es posible utilizarlo a nivel empresarial o si se puede obtener beneficios extra por su uso.

- **¿Para qué sirve?**

  La Raspberry pi ha sido utilizada en varios proyectos, como por ejemplo en impresoras y escáner 3D, miniordenadores desde la aparición del modelo Raspberry pi 2 que dispone de 4 núcleos y 1 Gb de RAM, mediacenter (reproducción de contenidos multimedia), estaciones meteorológicas… es decir en todo proyecto que necesites un ordenador que consuma y ocupe muy poco.

- **¿Modelo utilizado y por qué?**

  El modelo utilizado en el proyecto es la última versión a 20/10/2016 la Raspberry pi 3, la cual incorpora una pequeña mejora en el procesador respecto a la Raspberry pi 2, pero una gran mejora ya que incorpora el módulo WIFI y el módulo bluetooth algo que no hacen las versiones anteriores, lo cual nos ahorra espacio y problemas de compatibilidad con módulos externos

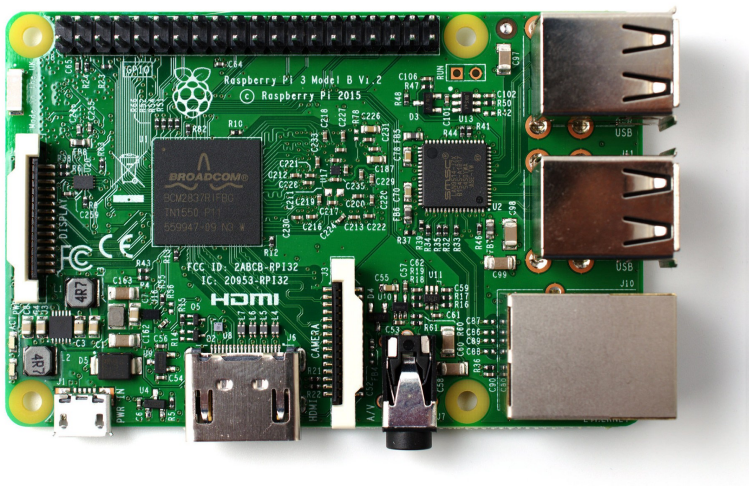

Figura 2: Placa Raspberry Pi 3

2.3 Cámara USB

La cámara utilizada es la Invido webcam de la marca trust, tiene conexión USB y una resolución máxima de 640 x 480 pixeles. Se ha elegido esta cámara aprovechando que ya disponía de ella y suponía un ahorro en el coste. Una mejora podría ser la cámara desarrollada para Raspberry pi que dispone de su propio conector directo a placa y una resolución de 1080 pixeles.

Figura 3: Camara USB

2.4 Componentes electrónicos

- **Protoboard**

  La protoboard es una de las placas de pruebas más usadas. Es un tablero con orificios conectados entre sí de manera interna y debido a su reducido tamaño, poco peso y la facilidad de uso al no ser necesario soldar los componentes la convierten en la mejor opción.

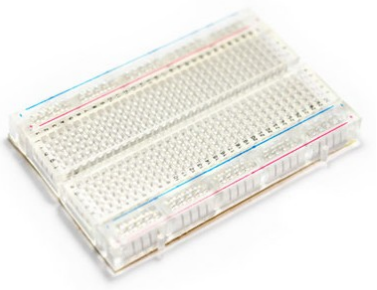

Figura 4: Placa Protoboard

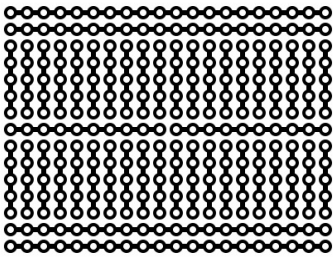

Figura 5: Imagen Conexiones Protoboard

- **Sensor de distancia con ultrasonidos HC-SR04**

  El HC-SR04 es un sensor de distancias por ultrasonidos capaz de detectar objetos y calcular la distancia a la que se encuentra en un rango de 2 a 500 cm. El sensor tiene todo lo necesario para hacer la medición. Tan solo con proporcionarle VCC = +5v , GND = -5v , enviar un pulso por la patilla trigger y recibir el retorno por la de echo, calculando antes el tiempo, podemos obtener la distancia.

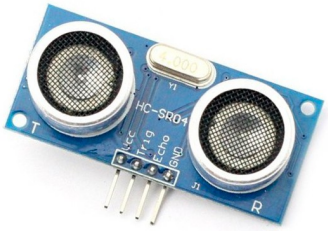

Figura 6: Sensor de Distancia HC-SR04

- **Circuito integrado L293D**

  El L293D es un integrado para controlar motores DC que usa el sistema puente en H. El puente en H es un sistema para controlar el sentido de giro de un motor DC usando cuatro transistores, que se comportan como interruptores y dependiendo que transistores conducen y cuales no, cambia el sentido de giro del motor. Con este integrado, además del sentido de giro, conseguimos controlar los motores de 12v con el Arduino uno y una batería externa de 12v.

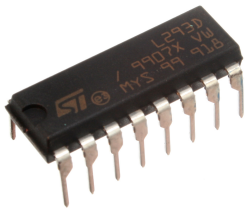

Figura 7: Integrado L293D

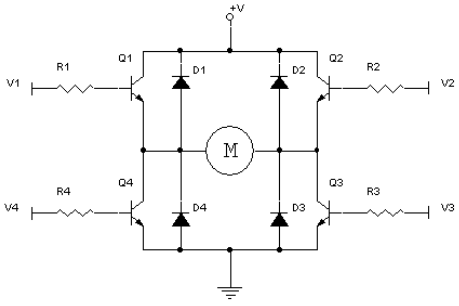

Figura 8: Imagen de conexión de un Puente en H.

- **Regulador de tensión L7805K**

  Este integrado nos ofrece la posibilidad de convertir la tensión de 12 V de la batería en 5 V con una corriente máxima de 1,5A. Para que el regulador funcione solo necesitamos introducir una tensión superior a 3V en la entrada Vi respecto de la salida Vo y el propio chasis como GND .

Figura 9: Regulador de tensión L7805K

- **Transistor BC547**

  El transistor BC547 es un transistor de baja potencia y baja frecuencia que se puede usar como un interruptor electrónico y como un amplificador de señal. En el proyecto se utiliza como interruptor.

Figura 10: Transistor BC547

3. SOFTWARE

El software representa toda la parte inmaterial o intangible, es lo conocido como programas. El software engloba toda la información digital que hace funcionar de una manera determinada al conjunto de elementos físicos y materiales que componen el computador. Ejemplos de software pueden ser los sistemas operativos y los programas que se instalan en estos para realizar tareas específicas.

3.1 Raspbian

- **¿Qué es Raspbian?**

Figura 11: Logo de Raspbian.

Es el sistema operativo que sa ha utilizado en la Raspberry pi en su versión Jessie, versión disponible desde septiembre de 2015. Raspbian es un sistema operativo libre basado en Debian optimizado para el hardware Raspberry pi.

Raspbian proporciona más que un sistema operativo puro. Incorpora más de 35.000 paquetes, software precompilado incluido y está preparado para una fácil instalación.

Raspbian fue creado por un pequeño y dedicado equipo de desarrolladores que son fans del hardware de Raspberry Pi.

- **¿Cómo instalarlo?**

Descargamos la imagen de la web oficial http://www.raspberrypi.org/downloads/
Una vez descargado el ZIP, hay que descomprimirlo para obtener el fichero .img que contiene dentro.
Necesitaremos una tarjeta micro SD donde instalaremos el sistema y que hace de disco duro, que debe ser mínimo de 4Gb, aunque lo recomendado es 8 o más ya que el sistema operativo ocupará ya casi los 4.

Necesitamos una ranura para tarjetas SD o un lector de tarjetas SD para USB.

Descargamos la utilidad Win32DiskImager desde la página del proyecto Sourceforge https://sourceforge.net/projects/win32diskimager/, extraemos el ejecutable del archivo zip y ejecutamos la utilidad Win32DiskImager.
Seleccionamos el archivo de imagen que se extrajo anteriormente, seleccionamos la letra de unidad de la tarjeta SD, cuidado de seleccionar la unidad correcta, hacemos clic en escribir y esperamos a que se complete la escritura.

Con el primer inicio de la Raspberry Pi se nos mostrará rapi-config, una herramienta de configuración de Raspbian para configurar las opciones principales. Después de esto se nos iniciará Raspbian ya configurado y preparado para empezar a trabajar.

3.2 Software de Arduino

- **¿Qué es el Software de Arduino?**

Figura 12: Logo de Arduino.

El IDE de código abierto de Arduino facilita la escritura de código y su posterior carga en la placa. Este IDE funciona en Windows, Mac OS X y Linux. El entorno está escrito en Java y otro software de código abierto. Este software se puede utilizar con cualquier tipo de placa Arduino.

- **¿Cómo instalarlo?**

Descargamos el archivo ZIP desde la web oficial, necesitaremos 155 MB de espacio libre en disco https://www.arduino.cc/download.php?f=/arduino-nightly-windows.zip.
Una vez descargado solo hay que extraer la carpeta en la ruta elegida y ejecutar el archivo .exe que contiene.

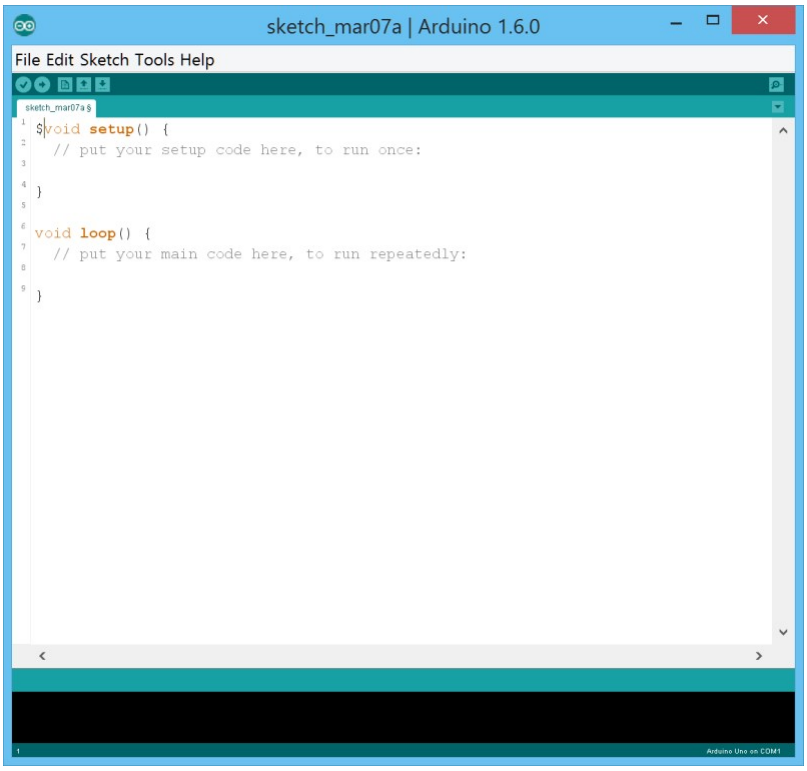

Figura 13: Captura IDE de Arduino.

3.3 Unity

- **¿Qué es Unity?**

Figura 14: Logo de Unity.

Unity es un motor de videojuego multiplataforma creado por Unity Technologies. Unity está disponible como plataforma de desarrollo para Windows, OS X y Linux. La plataforma de desarrollo tiene soporte de compilación con diferentes tipos de plataformas. A partir de su versión 5.4.0 ya no soporta el desarrollo de contenido para navegador a través de su plugin web, en su lugar se utiliza WebGL. Unity tiene varias versiones entre las cuales existe la versión Unity Personal "con licencia gratuita si tu empresa factura menos de 100 mil $".

- **¿Cómo instalarlo?**

Descargamos el software desde la web oficial https://store.unity.com/es.

Una vez descargado solo hay que instalarlo.

3.4 SDK Android

- **¿Qué es el Android SDK?**

  El SDK de Android incluye un conjunto de herramientas de desarrollo. Comprende un depurador de código, biblioteca, un simulador de teléfono basado en QEMU, documentación, ejemplos de código y tutoriales. Las plataformas de desarrollo soportadas son Linux, OS X, y Windows.

  Aunque el IDE soportado oficialmente es Android Studio, en este Proyecto no usaremos Android Studio sino Unity.

- **¿Cómo instalarlo?**

  Descargamos el software desde la web oficial, https://dl.google.com/android/installer_r24.4.1-windows.exe?hl=es-419.
  Una vez descargado lo instalamos y prestamos atención a la ruta donde se instala, ya que cuando intentemos compilar desde Unity la aplicación Android nos pedirá la ruta donde está instalado Android SDK.

3.5 VNC Server

VNC también denominado Software de escritorio remoto, es un programa de software libre basado en una estructura cliente-servidor que permite tomar el control del ordenador servidor remotamente a través de un ordenador cliente.

- **Instalación**

  El primer paso es instalar el servidor VNC en la Raspberry Pi, escribiendo en la terminal de la Raspberry Pi a través de SSH usando el software Putty en Windows o en local si tenemos la Raspberry conectada a un monitor.

  - Sudo apt-get install tightvncserver

  Una vez instalado iniciaremos el servicio con el siguiente comando:

  - vncserver :1 –geometry 1280x800 –depth 16 –pixelformat rgb565

  Al finalizar el paso anterior instalamos el Cliente VNC en nuestro pc y así podemos acceder a la Raspberry como un escritorio remoto.

3.6 Putty

Putty es un cliente SSH con licencia libre. Disponible originalmente solo para Windows, ahora también está disponible en varias plataformas Unix, y se está desarrollando la versión para Mac OS clásico y Mac OS X

- **Instalación**

  Este software no necesita instalación, se puede descargar el ejecutable directamente desde la web en el siguiente enlace:

  https://the.earth.li/~sgtatham/putty/latest/x86/putty.exe

4. DESARROLLO

4.1 Lenguajes de Programación

4.1.1 Python

Python es un lenguaje de programación interpretado. Se trata de un lenguaje de programación multiparadigma ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional, usa tipado dinámico y es multiplataforma.

He utilizado este lenguaje en el proyecto para la programación de los servidores alojados en la Raspberry, ya que Raspbian viene con Python 2, Python 3 y un IDL para Python instalados. Además dispone de la librería para controlar los pines GPIO

4.1.2 C#

C# es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET.

Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET similar al de Java, aunque incluye mejoras derivadas de otros lenguajes.

He utilizado este lenguaje en el proyecto para la programación de la aplicación cliente, implementada en Unity a base de scripts.

La otra posibilidad habría sido implementar los scripts en Javascript, pero debido a un mayor conocimiento sobre el lenguaje C# he usado este lenguaje.

4.1.3 C

C es un lenguaje orientado a la implementación de Sistemas Operativos, concretamente Unix. C es apreciado por la eficiencia del código que produce y es el lenguaje de programación más popular para crear software de sistemas, aunque también se utiliza para crear aplicaciones.

Se trata de un lenguaje de tipos de datos estáticos, débilmente tipificado, de medio nivel, ya que dispone de las estructuras típicas de los lenguajes de alto nivel pero, a su vez, dispone de construcciones del lenguaje que permiten un control a muy bajo nivel. Los compiladores suelen ofrecer extensiones al lenguaje que posibilitan mezclar código en ensamblador con código C o acceder directamente a memoria o dispositivos periféricos.

He utilizado este lenguaje para programar la placa de Arduino ayudándome de su software con el que posteriormente lo he cargado.

4.2 Red WIFI

La conexión con los servidores de la Raspberry se realiza a través una red WIFI creada en la Raspberry, y configurándola como punto de acceso. A continuación se describe el procedimiento.

Lo primero que tenemos que hacer una vez instalado el sistema operativo y configurado el inicio de la Raspberry, es conectarla a la red y asegurarnos que tenemos conexión a internet escribiendo el siguiente comando en la terminal:

```
ping 8.8.8.8 # Así comprobaremos que tenemos conexión a internet
```

Si tenemos respuesta ya podemos instalar hostapd, el software que nos permite crear un punto de acceso en nuestra Raspberry.

```
Sudo apt-get update # actualizamos
```

```
Sudo apt-get install hostapd isc-dhcp-server # instalamos hostapd
```

También debemos instalar un administrador de tablas IP con el comando:

```
Sudo apt-get install iptables-persistent
```

Aceptamos las 2 pantallas que salen.

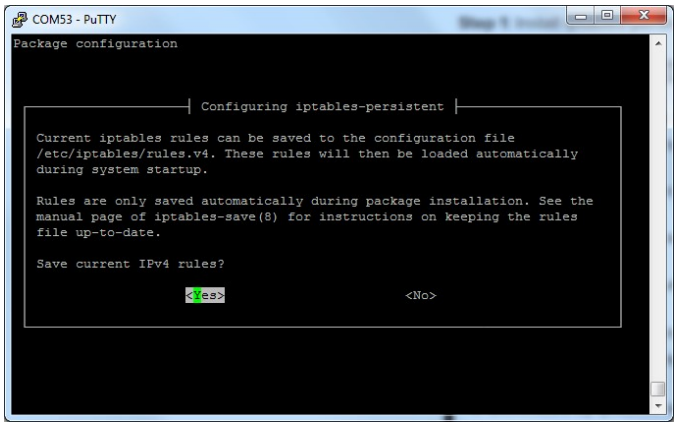

Figura 15: Pantalla tablas ip.

El siguiente paso es configurar el DHCP el cual posee una lista de direcciones IP dinámicas que ira asignando a los clientes conforme estas van quedando libres. Para eso tenemos que editar el siguiente fichero **/etc/dhcp/dhcpd.conf** con el siguiente comando:

```
sudo nano /etc/dhcp/dhcpd.conf
```

Busca las siguientes líneas:

option domain-name "example.org";option domain-name-servers ns1.example.org, ns2.example.org;

y añade # al comienzo.

#option domain-name "example.org";#option domain-name-servers ns1.example.org, ns2.example.org;

Busca la siguiente línea:

If this DHCP server is the official DHCP server for the local# network, the authoritative directive should be uncommented.#authoritative;

Y elimina # de .#authoritative

If this DHCP server is the official DHCP server for the local# network, the authoritative directive should be uncommented.authoritative;

Añade el siguiente código al final del archivo:

```
subnet 192.168.42.0 netmask 255.255.255.0 {
        range 192.168.42.10 192.168.42.50;
        option broadcast-address 192.168.42.255;
        option routers 192.168.42.1;
        default-lease-time 600; max-lease-time 7200;
        option domain-name "local";
        option domain-name-servers 8.8.8.8, 8.8.4.4;
```

}

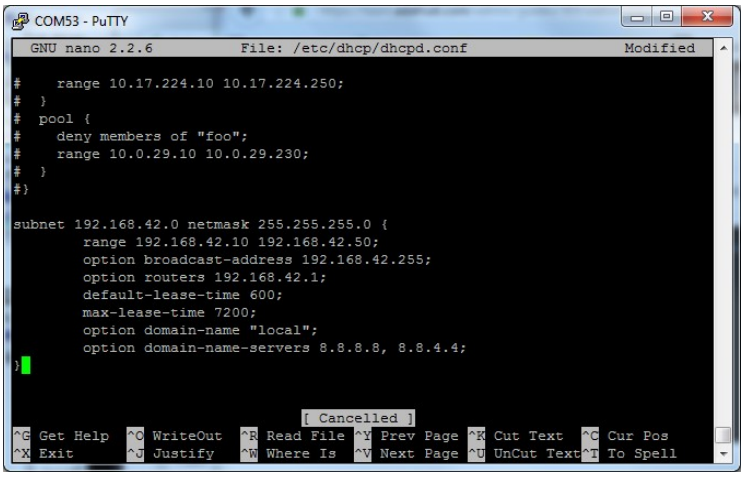

Figura 16: Captura dhcpd.conf

Salva el fichero con Control+x, luego pulsa Y + INTRO.

Ahora modificaremos el siguiente fichero:

sudo nano /etc/default/isc-dhcp-server

Haz scroll hasta el final del fichero y sustituye **INTERFACES=""** por:

INTERFACES="tu adaptador wifi" en mi caso **INTERFACES=”wlano”.**

Cierra y salva el fichero.

El siguiente paso es configurar nuestro adaptador WIFI con una IP estática, con el siguiente comando desactivaremos la interfaz antes de configurar la IP estática:

sudo ifdown wlano

Una vez desactivada editamos el fichero.

sudo nano /etc/network/interfaces

Buscamos **auto wlano** y añadimos un comentario # y luego añadimos la siguiente línea al fichero:

iface wlano inet static address 192.168.42.1 netmask 255.255.255.0

Cierra y salva el fichero.

Asignamos la dirección IP estática ejecutando

sudo ifconfig wlano 192.168.42.1

Ahora podemos configurar el punto de acceso.

Creamos un fichero con: sudo nano /etc/hostapd/hostapd.conf

Copiamos lo siguiente en el fichero donde ssid es el nombre de la red y wpa_passphrase es la contraseña de acceso:

```
interface=wlano
driver=rtl871xdrv
```

```
ssid=Pi_AP
country_code=US
hw_mode=g
channel=6
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=Raspberry
wpa_key_mgmt=WPA-PSK
wpa_pairwise=CCMP
wpa_group_rekey=86400
ieee80211n=1
wme_enabled=1
```

Como en mi caso estoy usando la Raspberry Pi 3 comentaré la línea de driver ya que no es el correcto para esta versión.
Y salvamos el fichero.
Editamos el fichero `sudo nano/etc/default/hostapd`
Sustituimos la línea:
#DAEMON_CONF="" por **DAEMON_CONF="/etc/hostapd/hostapd.conf"**
Y salvamos el fichero.
Editamos el fichero `sudo nano/etc/init.d/hostapd`
Sustituimos la línea:
DAEMON_CONF="" por **DAEMON_CONF="/etc/hostapd/hostapd.conf"**
Y salvamos el fichero.
El siguiente paso es configurar la NAT lo que nos permitirá la conexión de múltiples clientes a nuestra red Wifi y tener toda la información a través de una única IP.
Ejecutamos `sudo nano /etc/sysctl.conf`
y añadimos al final del fichero `net.ipv4.ip_forward=1` en una línea nueva y guardamos el fichero. Esto iniciará el reenvío IP al arrancar.

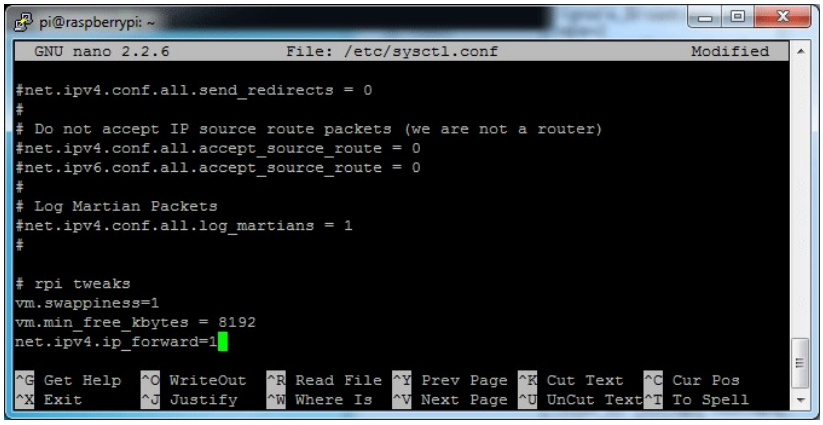

Figura 17: Captura sysctl.conf

Tambien ejecutaremos sudo sh -c "echo 1 > /proc/sys/net/ipv4/ip_forward" para activarlo inmediatamente.
Ejecute los siguientes comandos para crear la traducción de red entre el puerto Ethernet etho y el puerto wifi wlano.

sudo iptables -t nat -A POSTROUTING -o etho -j MASQUERADEsudo iptables -A FORWARD -i etho -o wlano -m state --state RELATED,ESTABLISHED -j ACCEPTsudo iptables -A FORWARD -i wlano -o etho -j ACCEPT

Podemos ver el contenido de las tablas con el siguiente comando:

sudo iptables -t nat -S
sudo iptables –S

Para que esto suceda en el reinicio y no tener que escribirlo cada vez, ejecutar:

sudo sh -c "iptables-save > /etc/iptables.ipv4.nat"

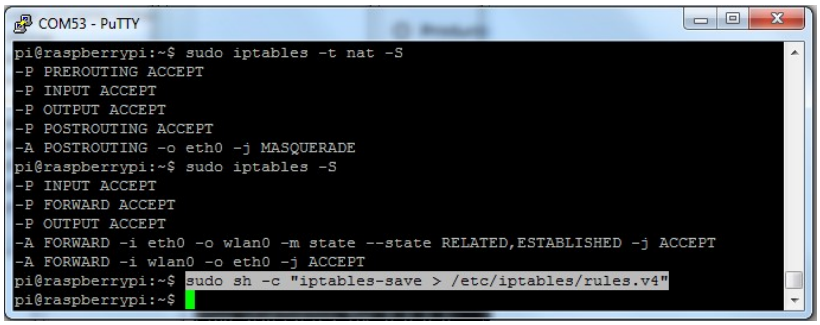

Figura 18: Captura contenido iptables

La herramienta iptables-persistent que instalamos al principio recargará automáticamente la configuración en el arranque.

Debemos actualizar hostapd con unos casos a tener en cuenta:

Si estamos usando una Raspberry pi kernel 4.4.13-v7+ o posterior, no necesitamos realizar el siguiente paso (el comando uname –a nos permite comprobar nuestra versión).

Si estamos usando el wifi incorporado de una Raspberry pi 3 o no estamos usando el adaptador WiFi de tipo RTL8192, debemos saltar el siguiente paso.

Antes de que podamos ejecutar el software para crear el punto de acceso, tenemos que actualizarlo a una versión que soporte nuestro adaptador WiFi.

Primero descargamos la nueva versión:

```
wget http://adafruit-download.s3.amazonaws.com/adafruit_hostapd_14128.zip
```

Descomprimimos el archivo zip.

```
unzip adafruit_hostapd_14128.zip
```

Creamos una copia del fichero original.

```
sudo mv /usr/sbin/hostapd /usr/sbin/hostapd.ORIG
```

Sustituimos el archivo original por la nueva versión.

```
sudo mv hostapd /usr/sbin
```

Para que pueda funcionar hacemos propietario a root.

```
sudo chown root:root /usr/sbin/hostapd
```

```
sudo chmod 755 /usr/sbin/hostapd
```

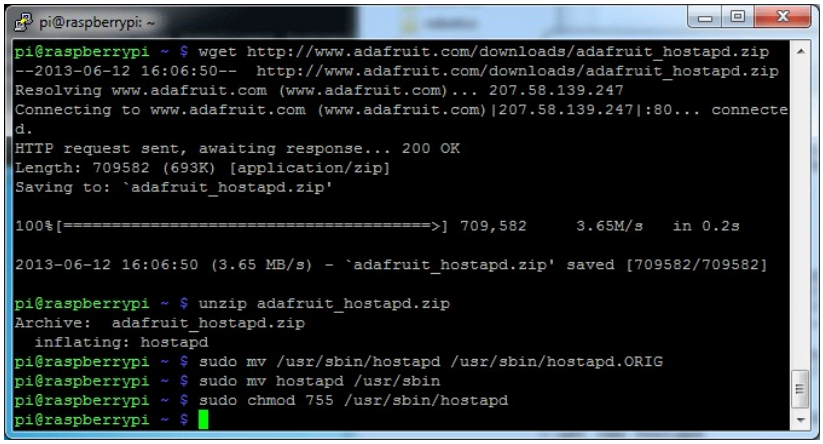

Figura 19: Captura hacer propietario a root

Y finalmente ya podemos probar el punto de acceso ejecutando:

```
sudo /usr/sbin/hostapd /etc/hostapd/hostapd.conf
```

Lanzaremos hostapd de manera manual y comprobaremos mediante el WiFi de otro dispositivo si nos aparece nuestro SSID, lo que nos indicará que tenemos el punto de acceso bien configurado.

Si obtenemos el siguiente mensaje:

Configuration file: /etc/hostapd/hostapd.confLine 2: invalid/unknown driver 'rtl871xdrv'1 errors found in configuration file '/etc/hostapd/hostapd.conf'Failed to set up interface with /etc/hostapd/hostapd.confFailed to initialize interface

es porque no estamos usando un adaptador WiFi RTL871Xdrv (por ejemplo, Pi 3 wifi interno) y debemos comentar la línea driver = rtl871xdrv en la configuración de hostapd o está utilizando ese chipset pero necesita descargar nuestro binario hostapd recompilado.
Si funciona, debemos obtener algo como esto:

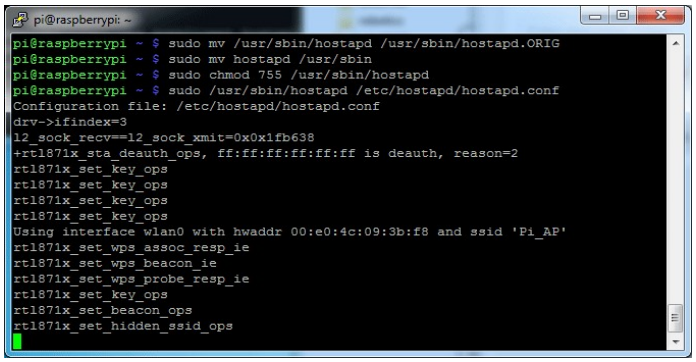

Figura 20: Captura cambio adaptador WiFi 1

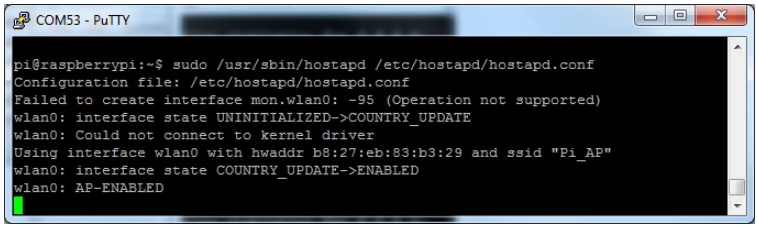

Figura 21: Captura cambio adaptador WiFi 2

Y veremos aparecer una red WiFi con nuestro SSID.

En algunos casos podría ser necesario eliminar el WPA-Supplicant. Lo haremos mediante el siguiente comando:

sudo mv /usr/share/dbus-1/system-services/fi.epitest.hostap.WPASupplicant.service ~/

Luego reiniciaremos (sudo reboot) y ejecutaremos hostapd nuevamente.

Ya solo nos queda configurar hostapd para que inicie solo al arrancar nuestra raspberry con los siguientes comandos:

sudo service hostapd start

sudo service isc-dhcp-server start

Podemos comprobar el estado del servicio de hostapd.

sudo service hostapd status

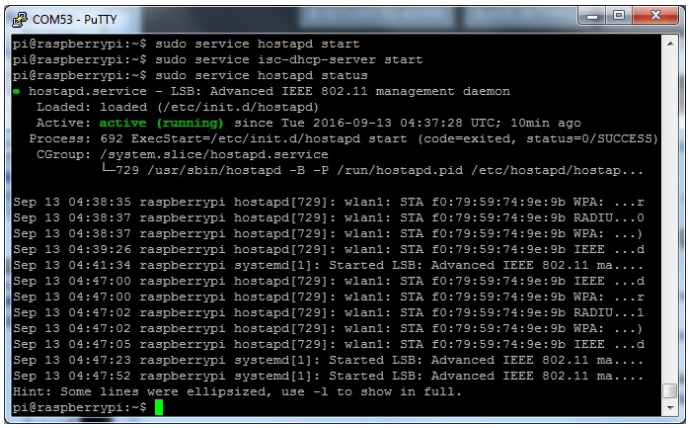

Figura 22: Captura estado servicio hostapd

También podemos comprobar el estado del servicio dhcp con:

sudo service isc-dhcp-server status

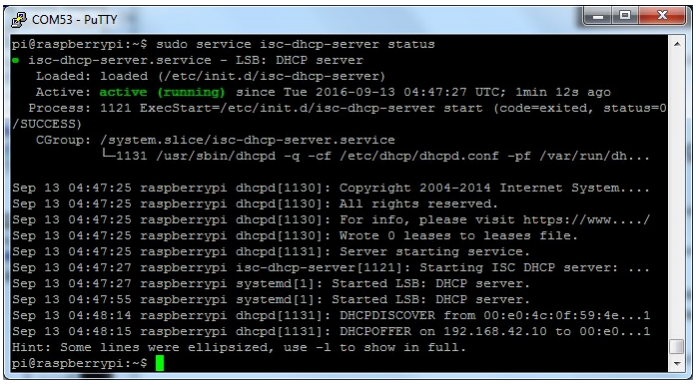

Figura 23: Captura estado servicio isc-dhcp-server

Antes de configurar el inicio de los servicios al arranque, verificar que inician y funcionan correctamente y después con los siguientes comandos los pondremos en el arranque.

```
sudo update-rc.d hostapd enable
sudo update-rc.d isc-dhcp-server enable
```

4.3 Servidores

Esta parte de la aplicación desarrollada en Python se encuentra en la Raspberry, compuesta por 2 servidores, uno que recibe las órdenes para controlar el Arduino y otro que se encarga de capturar las imágenes desde la cámara y enviarlas al cliente en la aplicación móvil.

4.3.1 Servidor de Control

El siguiente script en Python compone el servidor de control, el cual se ocupa del manejo del coche a través de las órdenes recibidas en la Raspberry.

```
import serial
import socket
import cv,cv2
import numpy as np
import time
import sys
import os
import cPickle

class ThreadedServer(object):
    def __init__(self, host, port):
        print("servidor iniciado IP "+str(host)+" PORT "+str(port))

    def listenToClient(self, host, port):
        self.host = host
        self.port = port
        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        sock.bind((self.host, self.port))
        self.runer(sock)
    def runer(self,sock):
        data, address = sock.recvfrom(1024)
        print("se han conectado con la IP "+str(address[0]))
        print("mensaje del cliente: "+str(data))
        sock.sendto("Estas conectado al servidor de arduino",address)
        dire=str(address[0])
        print('Starting!')
        arduino = serial.Serial('/dev/ttyACM0', 9600)
        while True:
            data, address = sock.recvfrom(1024)
            arduino.write(data)
            if data == 'H':
                print('LED ENCENDIDO')
            elif data == 'L':
                print('LED APAGADO')
            elif data == 'S':
                print('CAM MIRA ARRIBA')
            elif data == 'B':
                print('CAM MIRA ABAJO')
            elif data == 'I':
                print('CAM MIRA IZQUIERDA')
            elif data == 'D':
                print('CAM MIRA DERECHA')
            elif data == 'C':
                print('CAM MIRA DELANTE')
            elif data == 'Q':
                print('CLIENTE DESCONECTADO')
                break
        arduino.close() #finalizamos la comunicacion
        print("Cliente desconectado")
        sock.close()

if __name__ == "__main__":
    port_num = 5453
while True:             ThreadedServer('192.168.42.1',port_num).listenToClient('192.168.42.1',port_num)
```

El siguiente script en Python compone el servidor de Imagen, el cual se ocupa de capturar las imágenes con la cámara usb y servirlas desde la Raspberry hasta el cliente en el dispositivo móvil.

```
#!/usr/bin/python
import socket
import cv,cv2
import numpy as np
import time
import sys
import os
import cPickle
import threading
import thread
import RPi.GPIO as GPIO
import time

#configuracion de los pines

GPIO.setmode(GPIO.BOARD)

Trig = 10
Echo = 12

GPIO.setup(Trig,GPIO.OUT)
GPIO.setup(Echo,GPIO.IN)

class ThreadedServer(object):
    def __init__(self, host, port):
        print("servidor iniciado IP "+str(host)+" PORT "+str(port))

    def listen(self, host, port):
        print("passo")

    def captimg(self, dire):
        cap = cv2.VideoCapture(0)
        while True:
            try:
                ret, img = cap.read(1024)
                cv2.imwrite('captimage.jpeg', img)
                print(os.path.getsize('captimage.jpeg'))
            except:
                print("except")
    def listenToClient(self, host, port):
        self.host = host
        self.port = port
        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        sock.bind((self.host, self.port))
        sock2 = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        sock2.bind(("192.168.42.1", 5452))
        t=thread.start_new_thread(self.runer2, (sock2,))
        self.runer(sock)
    def runer(self,sock):
        data, address = sock.recvfrom(1024)
        print("se han conectado con la IP "+str(address[0]))
        print("mensaje del cliente: "+str(data))
        sock.sendto("Estas conectado al servidor de la camara",address)
        dire=str(address[0])
        cap = cv2.VideoCapture(0)
        cap.set(cv.CV_CAP_PROP_FRAME_WIDTH, 320)
        cap.set(cv.CV_CAP_PROP_FRAME_HEIGHT, 240)
        while True:
            ret, img = cap.read(1024)
            cv2.imwrite(dire+'.jpeg', img)
            with open(dire+'.jpeg', 'rb') as f:
                bytestoSend = f.read(1024)
                sock.sendto(bytestoSend,address)
                while bytestoSend != "":
                    bytestoSend = f.read(1024)
                    sock.sendto(bytestoSend,address)
            try:
                data, address = sock.recvfrom(1024)
            except:
                break
        print("Cliente desconectado")
        sock.close()
        for t in self.threads:
            t.stop()
        sock2.close()
    def runer2(self,sock2):
        data2, address2 = sock2.recvfrom(1024)
        print("se han conectado con la IP "+str(address2[0]))
        print("mensaje del cliente: "+str(data2))
        sock2.sendto("Estas conectado al servidor de ultrasonidos",address2)
        print("socket2 en servicio")
        print "Sensor Ultrasonico"

        try:
            while True:
                print("while 1")
                #establece el trigger en bajo
                GPIO.output(Trig,False)
                time.sleep(0.5)
                #manda un  pulso de 10us por el gatillo
                GPIO.output(Trig,True)
                time.sleep(0.00001)
                GPIO.output(Trig,False)
                inicio = time.time()
                #mientras no se recibe nada esperamos
                while GPIO.input(Echo)==0:
                    print("while 2")
                    inicio = time.time()
                while GPIO.input(Echo)==1:
                    print("while 3")
                    final = time.time()
                t_transcurrido = final - inicio
                distancia = t_transcurrido * 34000
                distancia = distancia/2
                sock2.sendto("Distancia=%.1fcm"%distancia,address2)
        except:
            GPIO.cleanup()


if __name__ == "__main__":
    signal = ''
    port_num = 5454
    while True:
        ThreadedServer('192.168.42.1',port_num).listenToClient('192.168.42.1',port_num)
```

4.4 Aplicación

La aplicación esta compilada para android con unity, aunque también se podría compilar para pc, Linux, ios, partiendo del mismo código.

Con la aplicación somos capaces de controlar el coche y ver lo que está mostrando por la cámara en tiempo real, además de controlar el encendido y apagado de las luces y medir la distancia hasta el objetivo.

4.4.1 Cliente de Control

Esta es la parte del programa cliente que se conecta al servidor de control. Está definida en la aplicación móvil y manda nuestras órdenes al servidor el cual las traslada desde la raspberry al arduino a través del puerto serie. A continuación se encuentra el script en lenguaje C#:

```
using UnityEngine;
using System.Collections;
using System.Net.Sockets;
using System.Net;
using System.IO;
using System.Text;
using System.Linq;
using System;
using System.Threading;
using UnityEngine.UI;

public class ClienteCam : MonoBehaviour {
    private Thread _t1;
    byte[] packetData;
    byte[] packetData2;
    byte[] bytes = new byte[1024];
    byte[] bytes2 = new byte[1024];
    string IP;
    string HOST;
    string HOST2;
    int port;
    int port2;
    string message;
    string message2;
    bool luces = true;
    public VirtualJoystick jos1;
    public VirtualJoystick jos2;
    IPEndPoint ep;
    IPEndPoint ep2;
    Socket client;
    Socket client2;
    GameObject texto;
    Text tes;
    void Start()
    {
        try
        {
            IP = "192.168.42.1";
            HOST = "android";
            port = 5453;
            port2 = 5452;
            message = "Soy " + HOST + " y quiero algo";
            message2 = "Soy " + HOST + " y quiero algo";
            packetData = System.Text.ASCIIEncoding.ASCII.GetBytes(message);
            ep = new IPEndPoint(IPAddress.Parse(IP), port);
            client = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);
            client.SendTo(packetData, ep);
            client.Receive(bytes);
            print((Encoding.UTF8.GetString(bytes)));
            packetData2 = System.Text.ASCIIEncoding.ASCII.GetBytes(message2);
            ep2 = new IPEndPoint(IPAddress.Parse(IP), port2);
            client2 = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);
            client2.SendTo(packetData2, ep2);
            client2.Receive(bytes2);
            bytes2 = new byte[1024];
            texto = GameObject.Find("Distancia");
            _t1 = new Thread(distancia);
            _t1.Start();

        }
        catch
        {
            print("error");
        }
    }
     void Update () {
        string mes1 = jos1.dir;
        string mes2 = jos2.dir;
        message = mes1 + mes2;
        if (message == "") {
            message = "T";
        }
        packetData = System.Text.ASCIIEncoding.ASCII.GetBytes(message);
        client.SendTo(packetData, ep);
        mes1 = "";
        mes2 = "";
        message = "";
        tes = texto.GetComponentInChildren<Text>();
        tes.text = (Encoding.UTF8.GetString(bytes2));

    }
    public void centerCam() {
        message = "C";
        packetData = System.Text.ASCIIEncoding.ASCII.GetBytes(message);
        client.SendTo(packetData, ep);
    }
    void OnApplicationQuit()
    {
        _t1.Abort();
    }

    public void luzOn(){
        if (luces) {
            message = "L";
            packetData = System.Text.ASCIIEncoding.ASCII.GetBytes(message);
            client.SendTo(packetData, ep);
            luces = false;
        }
        else {
            message = "H";
            packetData = System.Text.ASCIIEncoding.ASCII.GetBytes(message);
            client.SendTo(packetData, ep);
            luces = true;
        }
    }
    public void distancia() {
        while (true){
            client2.Receive(bytes2);
            print((Encoding.UTF8.GetString(bytes2))); }
    }
}
```

4.4.2 Cliente de Imagen

Esta es la parte del programa cliente que se conecta al servidor de Imagen al cual le va solicitando las imágenes capturadas por la cámara y mostrándolas en el visor. A continuación se encuentra el script en lenguaje C#:

```
using UnityEngine;
using System.Collections;
using System.Net.Sockets;
using System.Net;
using System.IO;
using System.Text;
using System.Linq;
using System;
using UnityEngine.UI;
using System.Threading;

public class ClientImg : MonoBehaviour {

    byte[] packetData;
    byte[] bytes = new byte[1024];
    byte[] fileData;
    byte[] z;
    string IP;
    string HOST;
    string filename;
    int port;
    string message;
    byte[] image;
    byte[] image2;
    bool cerrojo;
    IPEndPoint ep;
    Socket client;
    private Thread _t1;
    Texture2D tex;
    public RawImage raw;
    int cont;
    int aux;
    string path;
    System.IO.FileStream fileSave;
    System.IO.BinaryWriter binary;
    GameObject imagecam;
    // Use this for initialization
    void Start()
    {
        try
        {
            print("entras");
            IP = "192.168.42.1";
            HOST = "android";
            port = 5454;
            message = "Soy " + HOST + " y quiero algo";
            packetData = System.Text.ASCIIEncoding.ASCII.GetBytes(message);
            ep = new IPEndPoint(IPAddress.Parse(IP), port);
            client = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);
            client.SendTo(packetData, ep);
            client.Receive(bytes);
            print((Encoding.UTF8.GetString(bytes)));
            tex = new Texture2D(2, 2);
            image = new byte[0];
            cont = 0;
            aux = 0;
            filename = ".png";
            path = Application.dataPath + "/Save/";
            imagecam = GameObject.Find("ImagenCam");
            raw = imagecam.GetComponentInChildren<RawImage>();
            _t1 = new Thread(capture);
            _t1.Start();
            cerrojo = false;
        }
        catch
        {
            print("error");
        }
    }

    void Update()
    {

        try
        {
            if (cerrojo)
            {
                tex.LoadImage(image2);
                raw.texture = tex;
                cerrojo = false;
            }
        }
        catch {
            print("catch");
        }
    }

    void OnApplicationQuit(){
        _t1.Abort();
    }
    void capture() {
        while (true)
        {
            if (client.Receive(bytes) == 0)
            {
                while (!cerrojo)
                {
                    image2 = image;
                    cerrojo = true;
                }
                image = new byte[0];
                client.SendTo(System.Text.ASCIIEncoding.ASCII.GetBytes(""), ep);
                client.SendTo(System.Text.ASCIIEncoding.ASCII.GetBytes(""), ep);
                print("sale");
            }
            else
            {
                print("reciviendo imagen " +cont.ToString());
                z = new byte[image.Length + bytes.Length];
                image.CopyTo(z, 0);
                bytes.CopyTo(z, image.Length);
                image = z;
            }
        }
    }
    void SaveToFile(byte[] image, string filenamesave)
    {

        fileSave = new FileStream(path + filenamesave, FileMode.Create);
        binary = new BinaryWriter(fileSave);
        binary.Write(image);
        fileSave.Close();
    }
}
```

4.4.3 *Interfaz Gráfica*

La siguiente imagen muestra una captura de pantalla de la aplicación móvil y una pequeña explicación de los elementos que la componen:

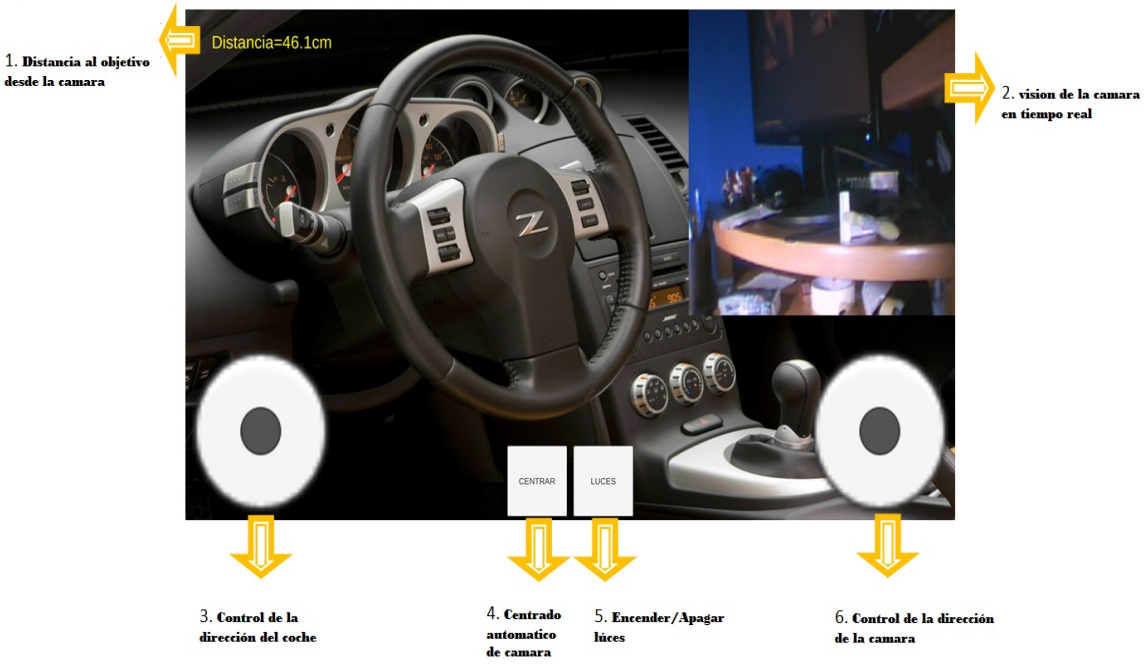

Figura 24: Interfaz Gráfica

1. Aquí la aplicación nos muestra la distancia desde la cámara hasta el objetivo en centímetros.
2. En esta ventana se puede ver en tiempo real lo que recoge la cámara.
3. Este mando nos permite controlar el movimiento del coche.
4. Este botón centra la cámara al frente automáticamente al pulsarlo, en caso de desorientarnos o necesidad.
5. Este botón nos permite encender y apagar las luces del coche.
6. Este mando nos permite controlar la orientación de la cámara.

4.5 Conexionado

La siguiente imagen muestra el diagrama de bloques de la aplicación y su funcionamiento:

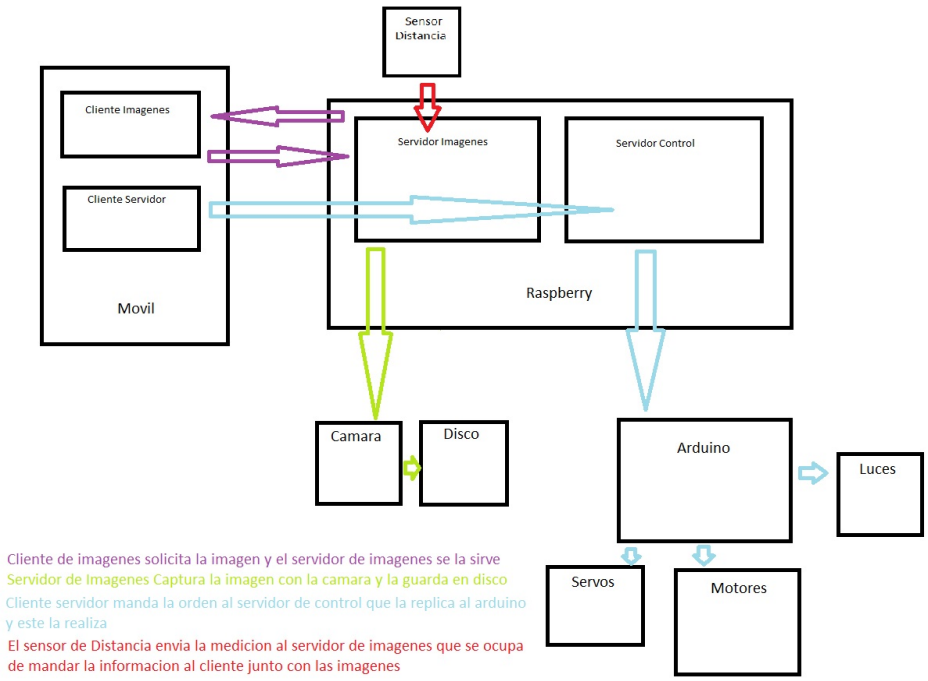

Figura 25: Diagrama de Flujo

La siguiente imagen nos muestra el esquema de conexiones del coche:

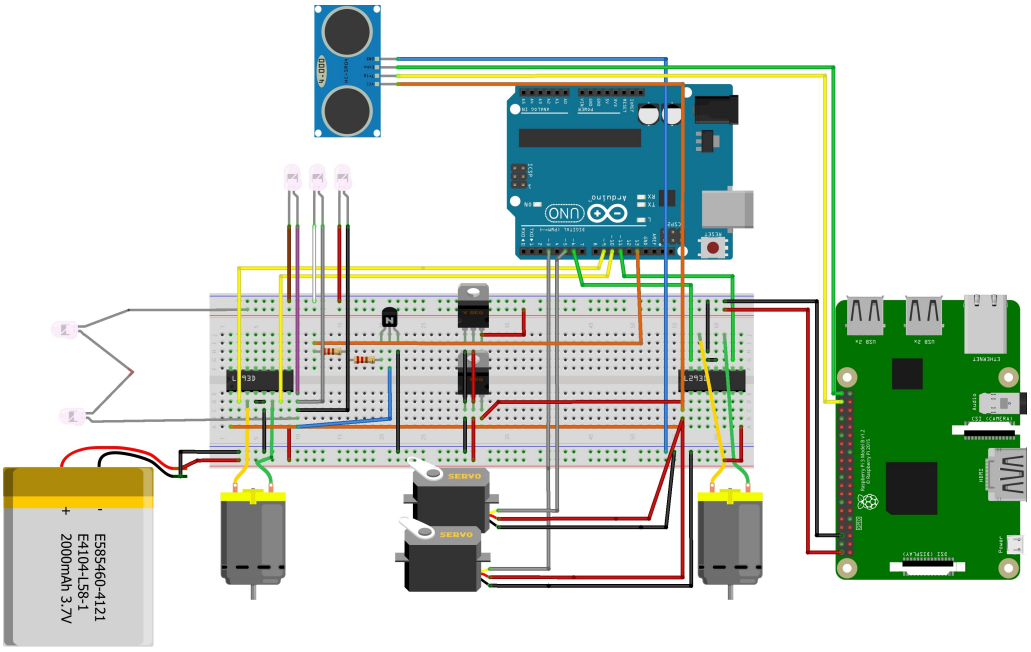

Figura 26: Esquema eléctrico del coche

Las siguientes imágenes nos muestran el cableado del coche instalado:

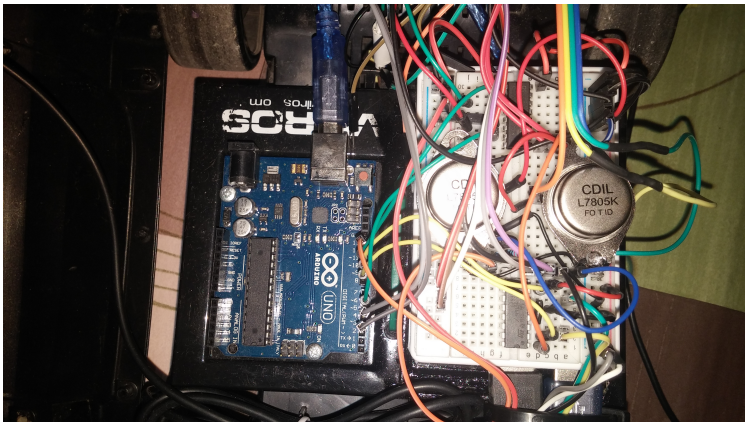

Figura 27: Cableado Interno del coche

5. PROBLEMAS ENCONTRADOS

En este proyecto me he encontrado con varios problemas de los cuales he aprendido mucho. El primer problema que tuve fue que arduino uno no era capaz de procesar la imagen (o al menos no como el proyecto necesitaba). El proyecto necesitaba grabar la imagen y reproducirla en el móvil a tiempo real y arduino uno no tiene un hardware capaz de hacerlo.

La solución que tomé para este problema fue insertar una raspberry pi 3, (ella si podría servir el video en tiempo real) y a la vez nos resolvería otro problema como era la comunicación por wifi, porque aunque arduino sí dispone de modulo wifi yo no disponía de él y hasta el momento la comunicación se realizaba por bluetooth. La Raspberry pi 3 dispone de wifi incorporado en la placa y como yo tenía una pude aprovecharla para solucionar estos problemas. La parte negativa fue que prácticamente tuve que empezar desde 0 el código otra vez para introducir todos estos cambios.

El siguiente problema con el que me encontré, el cual me resultó muy dificil solucionar, fue la comunicación por WiFi desde el móvil a la Raspberry, ya que para mi sorpresa el sistema operativo Android tiene capadas las redes Ad hoc sistema que yo había pensado utilizar. Después de probar e informarme ví que Android tiene su propia forma de hacer las cosas “ Wifi Direct ” que es el invento de android para este tipo de conexiones y el cual no fuí capaz de utilizar, lo que me llevó a crear una red WiFi configurando la Raspberry como punto de acceso.

Otro gran problema que me surgió fue el consumo, siendo más concreto la poca corriente que proporciona el Arduino 500 milis, al cual van conectados los servo motores, las luces leds, etc..., hasta que me dí cuenta de mi error. Las salidas de Arduino deben usarse para enviar señales y no para activar componentes, en mi caso las luces las cuales encendía con una salida, y lo solucioné con un transistor que ahora se activa y desactiva con esa salida y las luces se alimentan directamente desde la batería.

Esto mejoró el funcionamiento pero seguía teniendo problemas con la alimentación. Por una parte tenía varias baterías y tenía que alimentar todo desde la original del coche, y por otra había que duplicar algunos componentes para repartir el consumo (el problema con Arduino me ayudó a ver este problema). Así que mediante unos rectificadores para convertir los 12 v de la batería en 5 v necesarios poder alimentar la Raspberry y los demás componentes que lo necesitaran, conseguí tener solo una fuente de alimentación y duplicando los componentes que más consumo tenían los problemas desaparecieron.

A pesar de todos los problemas encontrados, en alguna ocasión como la del wifi y los fallos eléctricos que no entendía el por qué y que parecía que no encontraría la solución, he disfrutado mucho con este proyecto y he aprendido mucho ya que abarca

varios campos como la programación móvil, la comunicación y la electrónica y hace que me sienta muy orgulloso de haber podido desarrollarlo.

6. CONCLUSIONES

Los objetivos del proyecto se han conseguido, el coche se ha probado en condiciones de ausencia de luz y visión directa solo con la señal recibida en el móvil y medido la distancia hacia los obstáculos siendo todos los resultados satisfactorios.

7. BIBLIOGRAFIA

https://www.raspberrypi.org/downloads/raspbian/

https://www.arduino.cc/en/Main/Software

https://store.unity.com/es

www.es.wikipedia.org/wiki/Hardware

http://www.frambuesapi.co/2013/10/07/conexion-remota-al-raspberry-pi-usando-vnc/