Universidad Zaragoza
1542

Trabajo Fin de Grado

Diseño e implementación de un sistema de control remoto basado en Raspberry para una maqueta de un vehículo oruga.

Autor

Roberto Civera Jambrina

Director

Alfonso Blesa Gascón

Escuela Universitaria Politécnica de Teruel

2015

Diseño e implementación de un sistema de control remoto basado en Raspberry para una maqueta de un vehículo oruga.

Resumen.

El presente trabajo de fin de grado consiste en el diseño e implementación de un sistema de control remoto basado en RaspberryPi para una maqueta de un vehículo oruga utilizando las herramientas aprendidas a lo largo de la carrera.

Se ha utilizado la plataforma RaspberryPI como sistema de control, donde se controla tanto el movimiento del vehículo como la retrasmisión de imágenes. Para el control de los motores se ha utilizado el microcontrolador MC9S08QG8. Para el control de la potencia transmitida a los motores se ha utilizado el integradoL293D.

La estructura mecánica del vehículo se ha construido en marquetería, lo que permite no elevar el peso del vehículo.

Gracias a la comunicación wifi el control remoto puede realizarse desde un Smartphone o un pc.

Contenido

1 Introducción.

1.1 Presentación del problema.

Este proyecto presenta una aplicación para el control de un pequeño robot rodante dotado de una cámara para la retrasmisión de imágenes. El robot y la cámara estarán controlados a través de la plataforma Raspberry Pi. El control de los motores se realizara mediante un microcontrolador MC9S08QG8 mediante señal de tipo PWM. El interfaz de usuario se realizara desde un ordenador a través de un dongle wifi instalado previamente en la plataforma Raspberry Pi.

Este proyecto refleja una gran cantidad de áreas aprendidas durante la carrera. Algunos de los ejemplos son: programación en lenguaje C, comunicaciones, diseño de circuitos, programación de microcontroladores, manejo avanzado de Linux, control de motores mediante señal PWM.

1.2 Objetivos del proyecto.

El objetivo principal del proyecto es el desarrollo de una aplicación para el movimiento de un vehículo controlado mediante wifi desde un ordenador utilizando la plataforma Raspberry Pi.

En este proyecto existen los siguientes objetivos específicos a realizar:

- Construcción de una maqueta para la implementación de la aplicación.
- Diseñar el control del vehículo en la plataforma Raspberry Pi.
- Implementación de las características de comunicación de la Raspberry PI.
- Programación de un microcontrolador para el control de los motores mediante la señal de tipo PWM.
- Programación del protocolo I2C para la comunicación de la Raspberry Pi y el microcontrolador.
- Retransmisión de imágenes vía streaming tomadas por una cámara que portara el prototipo.

2. Análisis teórico.

2.1 Definición de robótica.

La robótica es una ciencia o rama de la tecnología, que estudia el diseño y construcción de máquinas capaces de desempeñar tareas realizadas por el ser humano o que requieren del

uso de inteligencia. Las ciencias y tecnologías de las que deriva podrían ser: el álgebra, los autómatas programables, las máquinas de estados, la mecánica o la informática.

2.2 Plataforma Raspberry PI

2.2.1 Introducción.

Raspberry Pi es un ordenador de placa reducida o placa única de bajo coste, desarrollado en Reino Unido por la Fundación Raspberry Pi, con el objetivo de estimular la enseñanza de la ciencia computacional en la escuela.

El diseño incluye un System-on-chip Broadcom BCM2835, que contiene un procesador central (CPU) ARM1176JZF-S a 700MHz, un procesador gráfico (GPU) Video Core IV, y 512 MiB de memoria RAM(aunque originalmente al ser lanzado eran 256 MiB). El diseño no incluye disco duro, ya que usa tarjeta SD para el almacenamiento. Tampoco incluye fuente de alimentación.

La fundación de soporte para la descargas de las distribuciones para arquitectura ARM, Raspbian(derivada de Debian),RISC OS 5, Arch Linux ARM(derivado de ARCH Linux) y Pidora, y promueve principalmente el lenguaje de programación Linux.

2.2.2 Especificaciones técnicas.

	Raspberry Pi Model B
SoC	Broadcom BCM2835
CPU	ARM 1176JZFS a 700 MHz
GPU	Videocore 4
RAM	512 MB
Video	HDMI y RCA
Resolución	1080p
Audio	HDMI y 3.5 mm
USB	2 x USB 2.0
Redes	Ethernet 10/100
Dimensiones	85.6mm x 53.98 mm
Electricidad	5V via micro USB oGPIO
Consumo energético	700mA(3.5W)
Periféricos de bajo nivel	8x GPIO, SPI, I2C, UART

2.2.3 Software.

Raspberry Pi usa mayoritariamente sistemas operativos basados en el núcleo Linux. Rasbian, una distribución de Debian que esta optimizada para el hardware de Raspberry Pi, se lanzo en julio de 2012 y es la distribución recomendada por la fundación para iniciarse, además es la distribución utilizada en este proyecto.

2.2.4 Programación de la Raspberry Pi.

La programación de la aplicación en la Raspberry se realiza en lenguaje C a través de clientes SSH, SSH es el nombre del protocolo que sirve para acceder a maquinas remotas. En Windows se ha realizado mediante el programa PuTTy de licencia libre y cuando se trabaja desde Linux se realiza a través del terminal. Si se quiere trabajar desde un móvil solo hay que descargar una aplicación de cliente SSH y se puede conectar con la Raspberry.

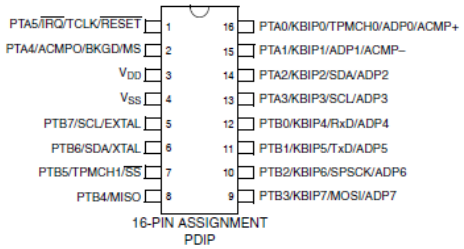
La programación se realiza en el sencillo editor de textos Nano para el terminal que ya viene instalado en la distribución Debian. Para que el archivo texto se convierta en un archivo C se guarda con la extensión .c. Una vez guardado en extensión .c, se compila con el compilador gcc que también viene ya instalado en la distribución y que compilara el programa y creara el ejecutable de la aplicación.

2.3 Microcontrolador MC9S08QG8.

2.3.1 Introducción.

Los microcontroladores son circuitos integrados que incorporan todos los bloques funcionales de un Sistema Microprocesador en un único encapsulado. Requieren una tensión continua estable (5V, 3.3V, 2.5V, 1.5V...) y un oscilador Secuencialmente interpretan unas instrucciones y generan señales digitales internas y/o externas que permita controlar un sistema electrónico. El microcontrolador MC9S08QG8 no requiere de oscilador extorno porque incluye uno interno, este microcontrolador requiere una tensión de 3.3 v para su alimentación.

El MC9S08QG8 es un microcontrolador de bajo coste, miembro la familia de alto rendimiento de microcontroladores de 8 bits HCS08. Todos los MCUs de la familia usan el núcleo HCS08 mejorada y están disponibles con una variedad de módulos de memoria, tamaños, tipos de memorias y tipos de encapsulado. El utilizado en este proyecto es el encapsulado de 16 pines PDIP.

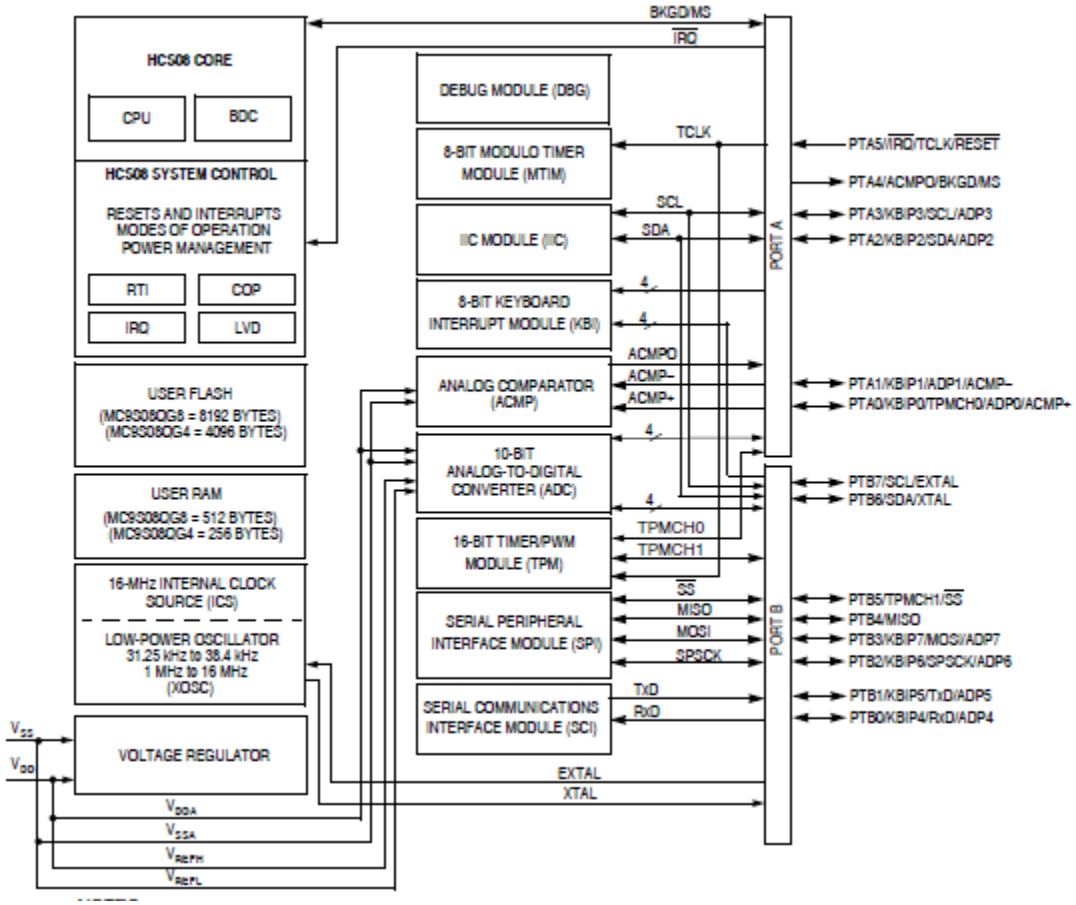


16-PIN ASSIGNMENT
PDIP

En la siguiente tabla se muestra el resumen de las características disponibles en la MC9S08QG8 /4 series de MCUs.

Feature	Device					
	MC9S08QG8			MC9S08QG4		
Package	24-Pin	16-Pin	8-Pin	24-Pin	16-Pin	8-Pin
FLASH	8K			4K		
RAM	512			256		
XOSC	yes	yes	no	yes	yes	no
ICS	yes			yes		
ACMP	yes			yes		
ADC	8-ch	8-ch	4-ch	8-ch	8-ch	4-ch
DBG	yes			yes	yes	yes
IIC	yes			yes		
IRQ	yes			yes		
KBI	8-pin	8-pin	4-pin	8-pin	8-pin	4-pin
MTIM	yes			yes		
SCI	yes	yes	no	yes	yes	no
SPI	yes	yes	no	yes	yes	no
TPM	2-ch	2-ch	1-ch	2-ch	2-ch	1-ch
I/O pins	12 I/O 1 Output only 1 Input only	12 I/O 1 Output only 1 Input only	4 I/O 1 Output only 1 Input only	12 I/O 1 Output only 1 Input only	12 I/O 1 Output only 1 Input only	4 I/O 1 Output only 1 Input only
Package Types	24 QFN	16 PDIP 16 QFN 16 TSSOP	8 DFN 8 SOIC	24 QFN	16 QFN 16 TSSOP	8 DFN 8 PDIP 8 SOIC

2.3.2 Diagrama de bloques de MC9S08QG8.



En este proyecto utiliza los pines PTA0 como TPMCH0, PTAB5 como TPMCH1, PTA5 como RESET, PTA3 como SCL, PTA2 como SDA y los pines de Vss y Vdd como GND y alimentación.

2.3.3 Entorno de programación.

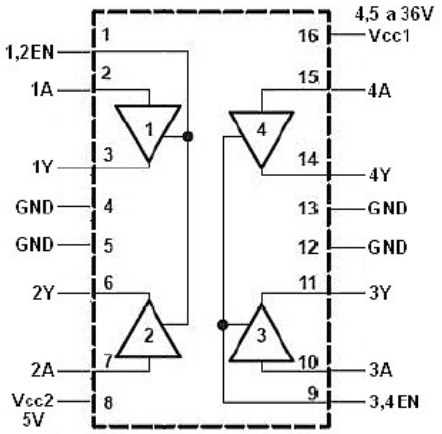
El entorno de programación del MC9S08QG8 de Freescale es el CodeWarrior IDE. Este IDE permite al usuario a través de todas las etapas de desarrollo desde el manejo hasta programar la memoria del microcontrolador.

2.4 Integrado L293D.

El integrado L293D incluye cuatro circuitos para manejar cargas de potencia media, en especial pequeños motores, con la capacidad de controlar corriente hasta 600 mA en cada circuido y una tensión entre 4.5V a 36V.

Los circuitos individuales se pueden usar de manera independiente para controlar cargas de todo tipo y además, cualquiera de los cuatro circuitos sirve para configurar la mitad de un puente H. El integrado permite formar dos puentes H completos, con lo que se puede manejar dos motores, con un manejo bidireccional.

En la siguiente figura se observa la distribución de pines del integrado y la tabla de funcionamiento de cada uno de los circuitos.



ENTRADAS†		SALIDA
A	**EN**	**Y**
H	H	H
L	H	L
X	L	Z

H = nivel alto **L** = nivel bajo **X** = indiferente
Z = alta impedancia **EN** = habilitación
† en modo de parada o corte por protección térmica, las salidas se ponen en estado de alta impedancia, sin que afecte al estado de las entradas

Este proyecto utiliza los circuitos 3 y 4 para el movimiento de los motores, por ello se habilitan estos circuitos mediante el pin 3,4EN y se alimenta al integrado por el pin 16 a 9v.

2.5 Comunicación wifi.

WLAN (Wireless Local Area Network, en ingles) es un sistema de comunicación de datos inalámbrico, muy utilizado como alternativa a las redes LAN cableadas. Utiliza tecnología de radiofrecuencia que permite mayor movilidad a los usuarios al minimizar las conexiones cableadas.

Características por las que se decide utilizar wifi en este proyecto.

- Movilidad: permite transmitir información desde el usuario a la Raspberry Pi siempre que los dos estén conectados a la misma red.
- Facilidad en la instalación: la instalación del dongle wifi se realiza mediante la consola o en modo grafico.
- Flexibilidad: puede llegar donde los cables no pueden, superando mayor numero de obstáculos. Permitiendo así el control del robot a mayor distancia que si este estuviera conectado mediante cable.

Para la conexión de la Raspberry Pi a la red wifi se ha utilizado un dongle wifi EDIMAX EW-7811Un con las siguientes características.

- Cumple con las normas inalámbricas 802.11b/g/n.
- Soporta una velocidad de datos de hasta 150Mbps.

- Conexión USB

- Soporta 64/128-bit WEP, WPA, WPA2 y compatible con WPS.
- Su diseño compacto hace ideal para dispositivos móviles.
- El adaptador inteligentemente puede ajustar la salida de la transmisión por la distancia y descarga de CPU para ayudar a reducir el consumo de energía cuando inalámbricamente está inactivo. Con la tecnología WLAN verde, el consumo de energía se puede reducir hasta un 20% ~ 50%.

Ilustración 1: Dongle wifi

2.6 Comunicaciones I²C.

2.6.1 Introducción.

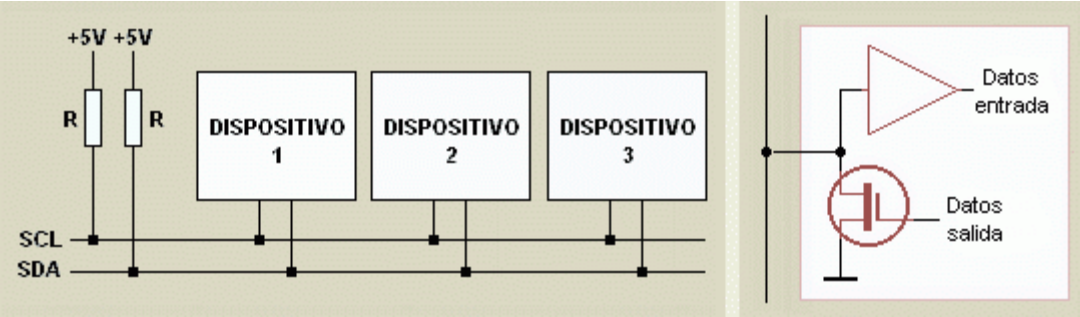
El bus I2C es un estándar que facilita la comunicación entre microcontroladores, memorias y otros dispositivos con cierto nivel de "inteligencia", sólo requiere de dos líneas de señal y un común o masa. Fue diseñado a este efecto por Philips y permite el intercambio de información entre muchos dispositivos a una velocidad aceptable, de unos 100 Kbits por segundo, aunque hay casos especiales en los que el reloj llega hasta los 3,4 MHz.

La metodología de comunicación de datos del bus I2C es en serie y sincrónica. Una de las señales del bus marca el tiempo (pulsos de reloj) y la otra se utiliza para intercambiar datos.

2.6.2 Descripción de las señales.

- SCL (System Clock) es la línea de los pulsos de reloj que sincronizan el sistema.
- SDA (System Data) es la línea por la que se mueven los datos entre los dispositivos.
- GND (Masa) común de la interconexión entre todos los dispositivos "enganchados" al bus.

Las líneas SDA y SCL son del tipo drenaje abierto, es decir, un estado similar al de colector abierto, pero asociadas a un transistor de efecto de campo (o FET). Se deben polarizar en estado alto (conectando a la alimentación por medio de resistores "pull-up") lo que define una estructura de bus que permite conectar en paralelo múltiples entradas y salidas.

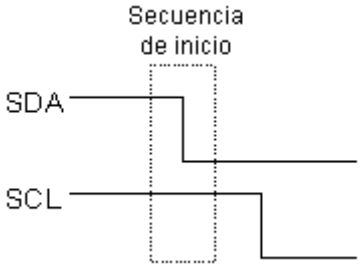


Las dos líneas del bus están en un nivel lógico alto cuando están inactivas. En principio, el número de dispositivos que se puede conectar al bus no tiene límites, aunque en este proyecto solo se conectaran al bus la Raspberry Pi como maestro y el microcontrolador como esclavo. El valor de las resistencias de polarización del bus en este proyecto es de 2K2 ohm. Se ha elegido este valor porque el consumo del integrado no es muy elevado y la sensibilidad al ruido es aceptable, además de ser adecuada para el tiempo de los flancos de subida y bajada de las señales.

2.6.3 Protocolo de comunicación IIC.

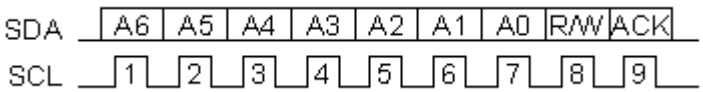
Estando los dos dispositivos conectados sobre el bus, es lógico que para establecer una comunicación a través de él se deba respetar un protocolo. En primer lugar existe un maestro como es la Raspberry y un esclavo como es el microcontrolador, sólo la Raspberry Pi como maestro pueden iniciar una comunicación.

La condición inicial, de bus libre, es cuando ambas señales están en estado lógico alto. En este estado la Raspberry Pi puede ocuparlo, estableciendo la condición de inicio (start). Esta condición se presenta cuando un dispositivo maestro pone en estado bajo la línea de datos (SDA), pero dejando en alto la línea de reloj (SCL).



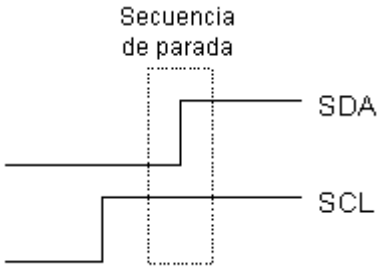
El primer byte que se transmite luego de la condición de inicio contiene siete bits que componen la dirección del microcontrolador, y un octavo bit que corresponde a la operación que se quiere realizar con él (lectura o escritura).

Si el dispositivo cuya dirección corresponde a la que se indica en los siete bits (A0-A6) está presente en el bus, éste contesta con un bit en bajo, ubicado inmediatamente luego del octavo bit que ha enviado el dispositivo maestro. Este bit de reconocimiento (ACK) en bajo le indica a la Raspberry Pi que el microcontrolador reconoce la solicitud y está en condiciones de comunicarse. Aquí la comunicación se establece en firme y comienza el intercambio de información entre los dispositivos.



Si el bit de lectura/escritura (R/W) fue puesto en esta comunicación a nivel lógico bajo (escritura), la Raspberry PI envía datos de la dirección en la que se tiene que mover el vehículo

al microcontrolador. Esto se mantiene mientras continúe recibiendo señales de reconocimiento, y el contacto concluye cuando se hayan transmitido todos los datos.



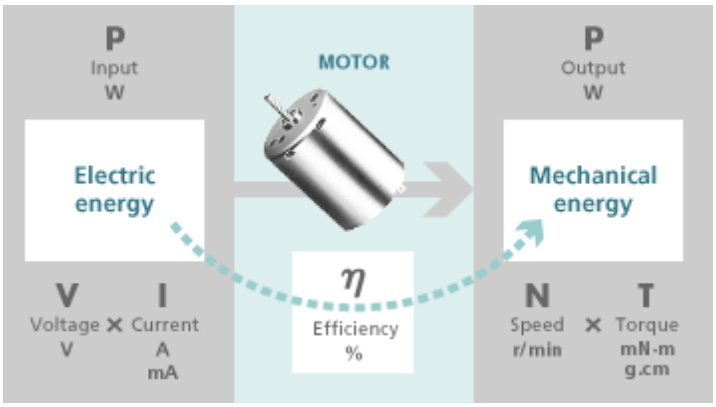
En el caso contrario, cuando el bit de lectura/escritura estaba a nivel lógico alto (lectura), la Raspberry Pi genera pulsos de reloj para que el dispositivo esclavo pueda enviar los datos de la dirección en la que realmente se está moviendo el vehículo para que el usuario vea por pantalla la dirección en la que se está moviendo el vehículo. Luego de cada byte recibido la Raspberry Pi (quien está recibiendo los datos) genera un pulso de reconocimiento.

La Raspberry Pi puede dejar libre el bus generando una condición de parada (stop).

Si se desea seguir transmitiendo, la Raspberry PI puede generar otra condición de inicio en lugar de una condición de parada. Esta nueva condición de inicio se denomina "inicio reiterado" y se puede emplear para direccionar un dispositivo esclavo diferente o para alterar el estado del bit de lectura/escritura.

2.7 Motores y PWM.

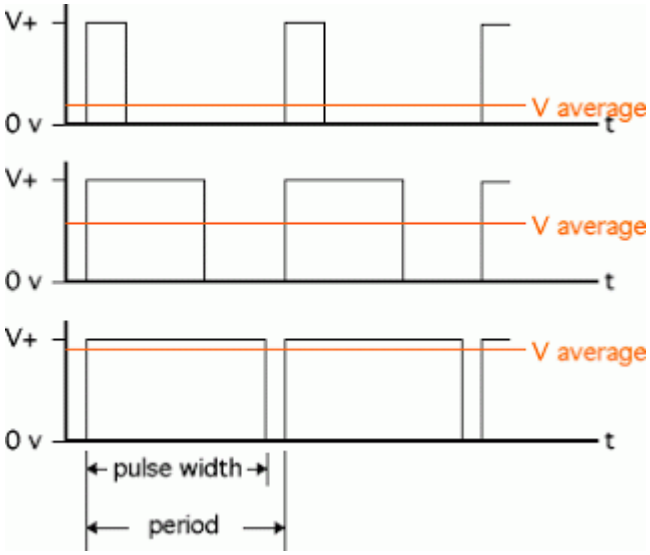
Un motor es una máquina que transforma la energía eléctrica en movimiento. Hay muchos tipos diferentes, pero el empleado en este proyecto es el motor de corriente continua con escobillas. Es un motor de bajo coste y es suficiente para mover el prototipo.



Un motor de corriente continua (DC) con escobillas está formado por un imán permanente en el exterior (estator) y unas bobinas electromagnéticas montadas en el eje del motor (armadura o rotor). Las escobillas se deslizan por unas piezas metálicas colocadas en el eje del motor, al girar estas conmutan la potencia de una bobina a otra, de forma que los campos magnéticos generados por las bobinas estén continuamente atraídos por el imán y el motor gire siempre en la misma dirección.

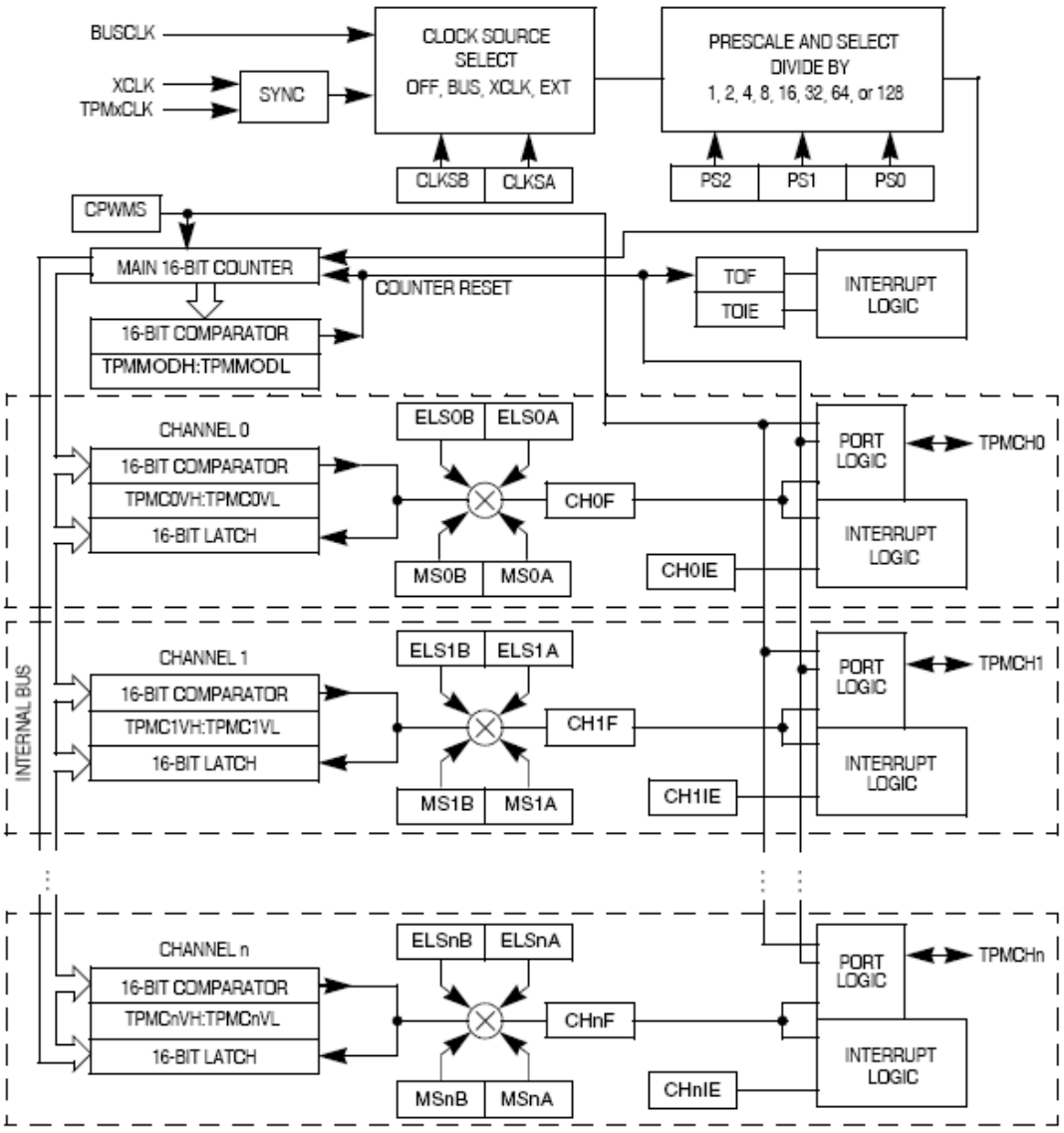
La modulación por anchura de pulso o PWM es una técnica en la que se modifica el ciclo de trabajo de una señal periódica, ya sea para transmitir información a través de un canal de comunicaciones o para controlar la cantidad de energía que se envía a una carga. En este caso se va a utilizar para controlar la cantidad de energía que se transmite a los motores.

Para una duración del pulso de un 25% o lo que es lo mismo un ciclo de trabajo del 25% del nivel máximo dará una tensión promedio de un 25% de la tensión máxima.



El microcontrolador MC9S08QG8 tiene un “bus clock” de 16MHz, el “timer” dispone de 2 canales, además dispone de un contador de 16 bits, un preescalado programable del reloj de entrada y puede generar interrupciones en el canal1, canal2 o desbordamiento del contador.

A continuación se muestra el diagrama de bloques del modulo timer del microcontrolador:



2.8 Tracción

Existen dos tipos de movimiento en el prototipo gracias a las orugas:

1. Si las dos ruedas tractoras se mueven a la vez, el movimiento será en línea recta.
2. Si solo gira una de las dos ruedas tractoras, el vehículo girara en su propio eje, proporcionando la capacidad de giro al vehículo.

Estos dos tipos de movimientos proporcionan al vehículo sencillez a la hora de programar el cálculo de la dirección en la que se mueve el vehículo, este fue uno de los motivos por el que se realizo este tipo de robot, descartando la construcción de una maqueta con cuatro ruedas y un servomotor para dotar al vehículo de capacidad de giro.

Gracias a estos dos movimientos también permitía la construcción de un vehículo con dos ruedas tractoras y una tercera rueda loca, este modelo de vehículo se descarto porque elevaba el coste de prototipo y su construcción era más complicada.

2.9 Alimentación.

2.9.1 Alimentación de motores.

Para la alimentación de los motores en el proyecto se utiliza un zócalo de 6 baterías de 1.5 voltios tipo AA LR6. El voltaje de una pila AA es de 1,5 Voltios y tienen una capacidad de 2000mAh.

Ilustración 2: Zócalo para baterías.

2.9.2 Alimentación Raspberry Pi y MC9S08QG8.

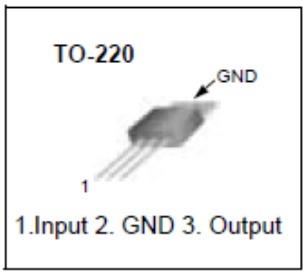
La alimentación de la Raspberry Pi se realiza a través de un regulador de tensión de 5 v, el cual es alimentado por zócalo de 9v igual que el utilizado para manejar los motores.

Para la alimentación del micro se vuelve a regular la tensión, mediante un divisor de tensión, para que de 5v que da el regulador pase a tres que necesita el micro para su correcto funcionamiento.

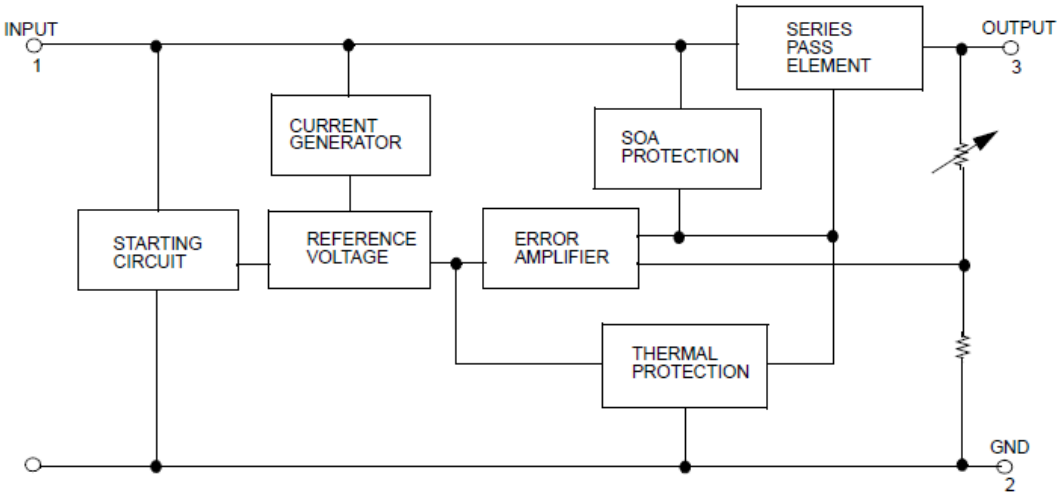
2.10 Regulador de tensión MC78T05CT.

Un regulador de tensión o regulador de voltaje es un dispositivo electrónico diseñado para mantener un nivel de tensión constante.

El utilizado en este proyecto es el MC78T05CT, que regula a 5v y que suministra una corriente de 3A.



Internal Block Diagram



2.11 Cámara Web

2.11.1 Introducción.

Una cámara web o cámara de red (en inglés: webcam) es una pequeña cámara digital conectada a una computadora la cual puede capturar imágenes y transmitirlas a través de Internet, ya sea a una página web o a otra u otras computadoras de forma privada.

2.11.2 Programa MJPG-Streamer.

MJPG Streamer es un simple y sencillo video streamer, pero más rápido que la mayoría de los programas de stream, esto lo hace ideal para proyectos de control remoto en el cual el tiempo real del video puede ser crucial para la navegación y orientación.

La aplicación captura JPG de webcams compatibles con Linux-UVC, sistema de archivos u otros plugins de entrada y los distribuye como M-JPG a través de HTTP para navegadores web.

Configuración del programa:

MJPG Streamer	
-i	input "<inputplugin.so> [parameters]"
-o	output "<outputplugin.so> [parameters]"
-h	help
-v	version
-b	background]...: fork to the background, daemon mode

Nota: Si se usa mjpg-streamer en blackground, se puede apagar de la siguiente forma:

kill `pidof mjpg_streamer`

Parámetros de entrada:

-d	video device to open (your camera)
-r	the resolution of the video device, can be one of the following strings: QSIF QCIF CGA QVGA CIF VGA SVGA XGA SXGA or a custom value like: 640×480
-f	frames per second
-y	enable YUYV format and disable MJPEG mode
-q	JPEG compression quality in percent (activates YUYV format, disables MJPEG)
-m	drop frames smaller then this limit, useful if the webcam produces small-sized garbage frames may happen under low light conditions
-n	do not initalize dynctrls of Linux-UVC driver
-l	switch the LED “on”, “off”, let it “blink” or leave it up to the driver using the value “auto”

Parámetros de salida

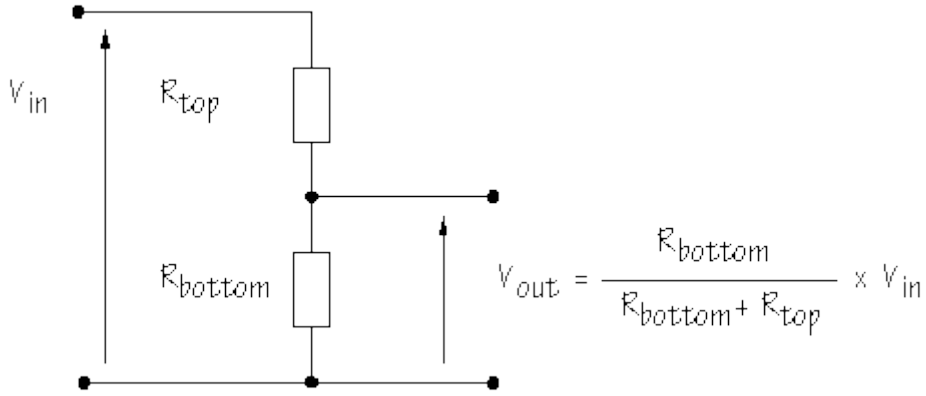
-w	folder that contains webpages in flat hierarchy (no subfolders)
-p	TCP port for this HTTP server
-c	ask for “username:password” on connect
-n	disable execution of commands

3. Diseño.

3.1 Esquema hardware.

A continuación se presenta el esquema hardware del proyecto,

Las resistencias R1 y R2, se calculan mediante un divisor de tensión para obtener los tres voltios necesarios para que el micro pueda funcionar.



Haciendo los cálculos R1 = 100 ohm y R2 = 180 ohm.

La alimentación de microcontrolador debe ser de 3v, por motivos ajenos al proyecto no se pudo conseguir un regulador de tensión, por lo que se decidió buscar la solución alternativa de construir un divisor de tensión.

Esta solución solo sería posible si la carga equivalente que genera el microcontrolador fuera lo suficientemente pequeña comparada con R2 (Rbottom en la figura), ya que si la carga equivalente fuera elevada la tensión Vout no serian los 3v deseados sino que la tensión seria más pequeña.

Los valores medidos de Vout no son significativamente distintos de 3v cuando los dos motores están funcionando por lo que el microcontrolador puede funcionar correctamente, por lo tanto la solución adoptada es válida.

Diagrama de bloques, donde se muestra todos los elementos hardware del proyecto:

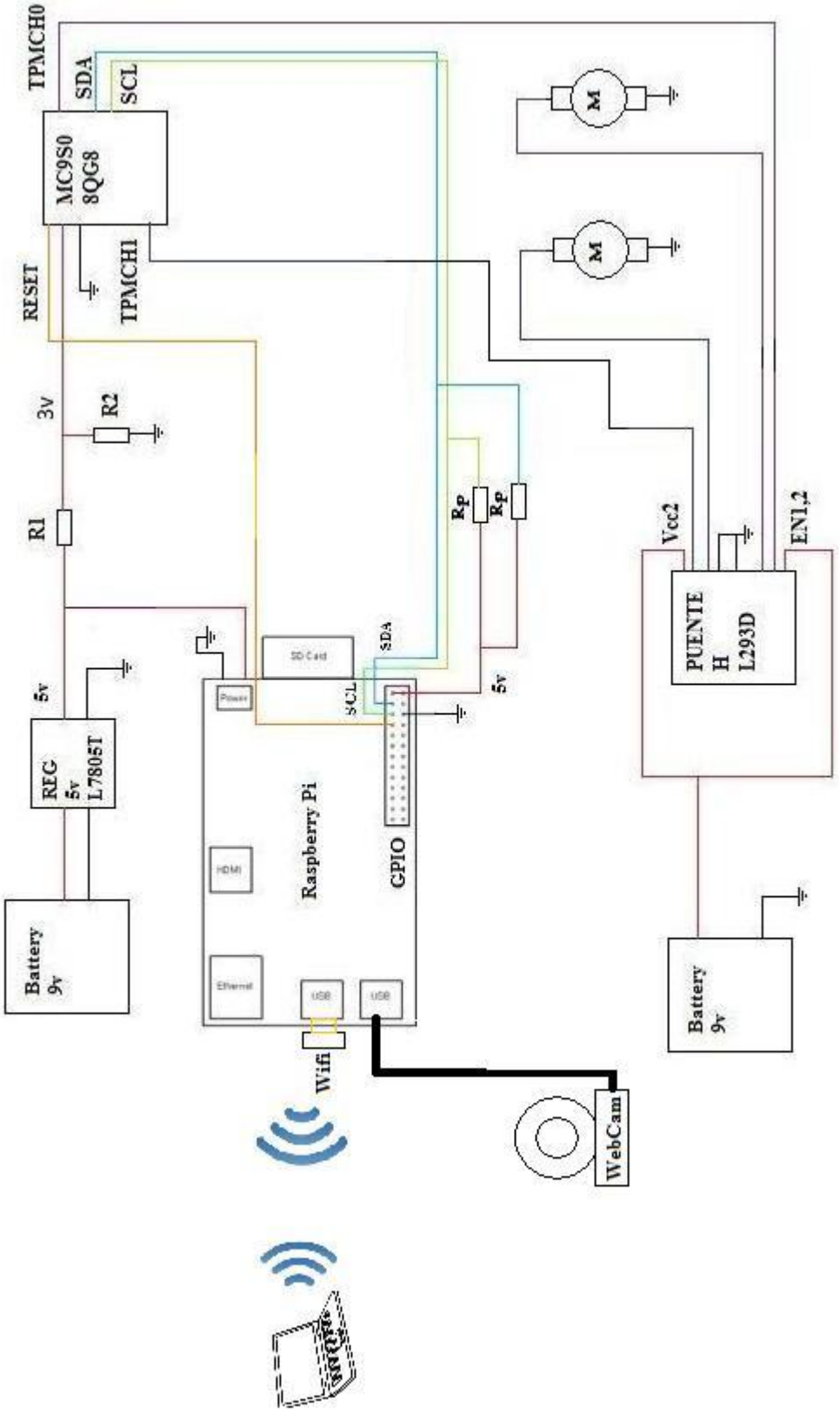


Ilustración 3: Diagrama de bloques.

A continuación se muestra el esquema eléctrico donde se muestra el cableado del proyecto:

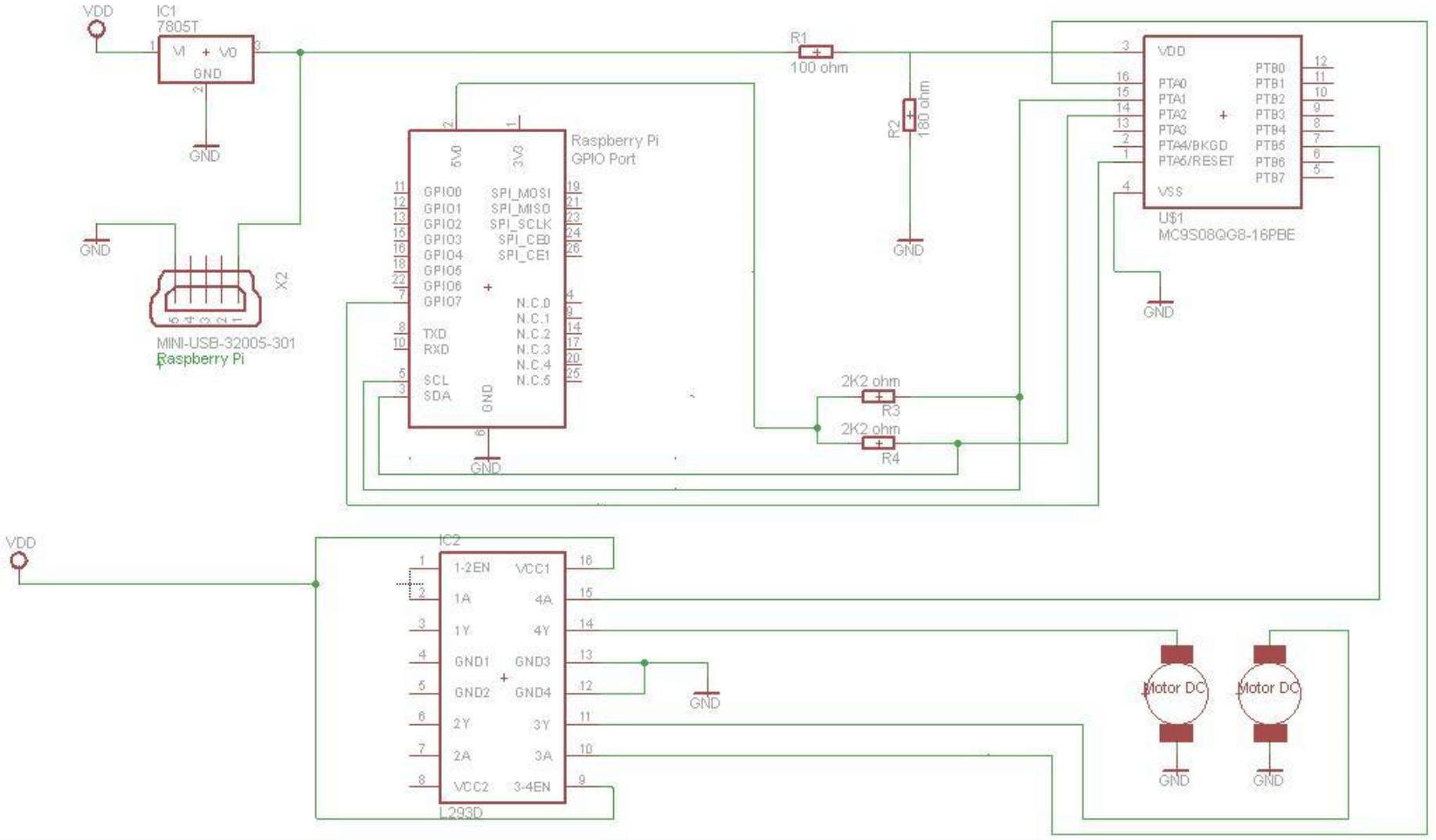
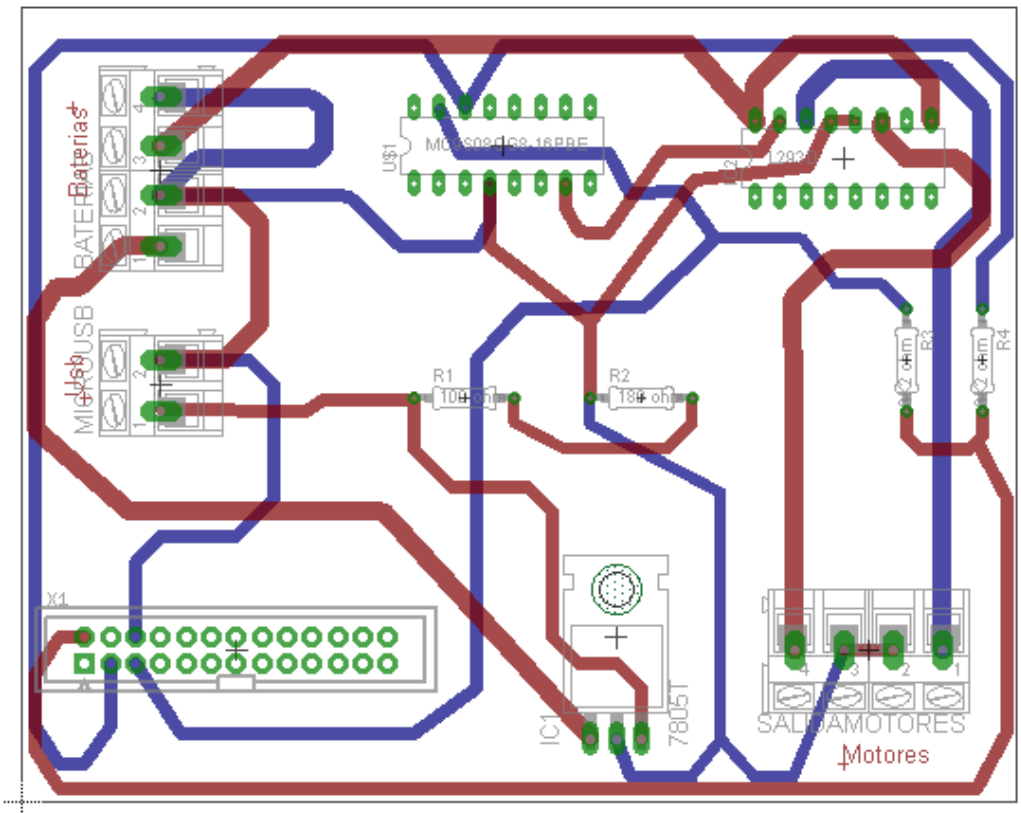

VDD
IC1
7805T
VI
VO
GND
R1
100 ohm
R2
180 ohm
GND
Raspberry Pi
GPIO Port
5V0
3V3
GPIO0
GPIO1
GPIO2
GPIO3
GPIO4
GPIO5
GPIO6
GPIO7
TXD
RXD
SCL
SDA
SPI_MOSI
SPI_MISO
SPI_SCLK
SPI_CE0
SPI_CE1
N.C.0
N.C.1
N.C.2
N.C.3
N.C.4
N.C.5
2K2 ohm
R3
2K2 ohm
R4
VDD
PTA0
PTA1
PTA2
PTA3
PTA4/BKGD
PTA5/RESET
VSS
PTB0
PTB1
PTB2
PTB3
PTB4
PTB5
PTB6
PTB7
U$1
MC9S08QG8-16PBE
X2
MINI-USB-32005-301
Raspberry Pi
IC2
1-2EN
1A
1Y
GND1
GND2
2Y
2A
VCC2
VCC1
4A
4Y
GND3
GND4
3Y
3A
3-4EN
L293D
Motor DC
Motor DC

Ilustración 4: Esquema eléctrico.

Para el diseño de la placa mediante Eagle se diseña la placa para la colocación de los compoenetes.



3.2 Diseño software.

Para el diseño software del proyecto es necesario la comunicación entre tres dispositivos, el primero es el dispositivo que controla y ejecuta la aplicación en la Raspberry, este dispositivo puede ser cualquiera que disponga de una aplicación de cliente SSH, a través de su pantalla y de su teclado se controla la aplicación.

La Raspberry pi es el segundo dispositivo y es la encargada de atender los hilos de la comunicación wifi, la comunicación con la webcam, la ejecución y cierre del programa MJPG Streamer, la comunicación IIC y del cálculo de la dirección del vehículo.

El microcontrolador es el encargado de atender al bus IIC cuando la Raspberry lo requiera y la generación del PWM según el valor recibido vía bus IIC.

Son necesarias dos aplicaciones para el funcionamiento del proyecto, una en la Raspberry y otra en el microcontrolador.

El programa principal de la Raspberry comienza con la inicialización de las librerías, a continuación se explican las más importantes:

- `#include <sys/ioctl.h>`
- `#include <sys/types.h>`
- `#include <sys/stat.h>`
- `#include <linux/i2c-dev.h>`

} Librerías destinadas al funcionamiento del bus IIC.

} Utilizada para lectura de una tecla sin tener que esperar a pulsar “intro”. Así como la impresión en pantalla de texto en color

- `#include <ncurses.h>`

- `#include <wiringPi.h>` } Librería destinada al control del GPIO

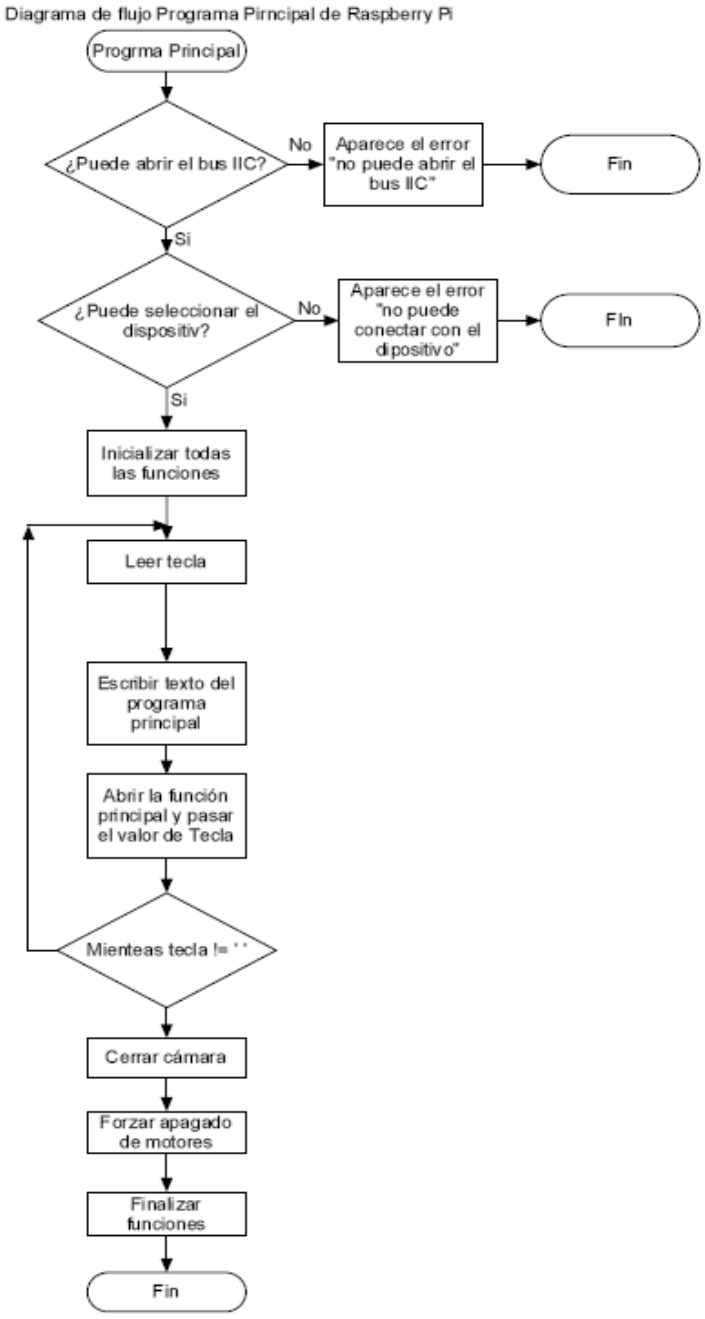
Las constantes definidas en el programa principal son:

- `#define ACABAR ' '` → El programa lo utiliza para salir de las funciones
- `#define micro_I2C_ADDR 0x38` → Se define la posición que ocupa el MS9S08QG8 en el bus IIC.

Las variables globales utilizadas en el programa principal son:

- `fd`: variable de tipo entero utilizada para el funcionamiento del bus IIC.
- `buf[]`: variable que representa al registro de datos del bus IIC en la Raspberry pi.
- `dirección`: variable de tipo entero utilizada para el cálculo de la dirección en la que se mueve el vehículo y que es transmitida al bus IIC.
- `error`: variable de tipo entero utilizada para la detección de errores en la comunicación IIC.
- `tecla`: variable de tipo char utiliza para tomar el valor de la tecla leída por teclado.

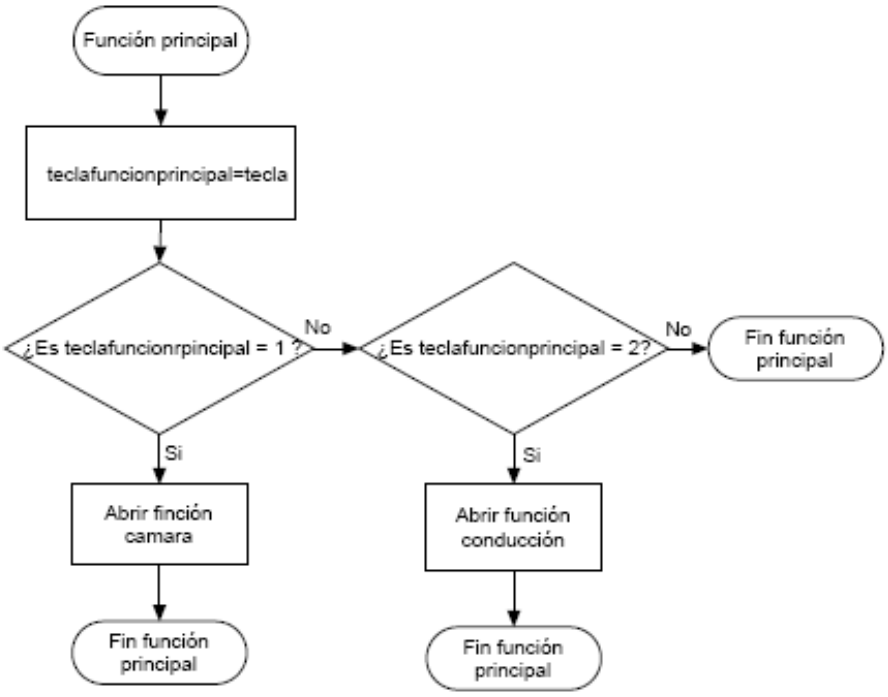
A continuación se inicializan el bus IIC y se selecciona el microcontrolador en el bus IIC para que funcione como esclavo.



La función principal del programa de la Raspberry es donde se elige si se abre la función de la cámara o la función de la conducción del vehículo. Para esta función se crea una variable local.

- `teclafuncionprincipal`: variable de tipo char que toma el valor leído por el teclado en la función principal.

Diagrama de flujo del menu principal



La función cámara es utilizada para manejar y mostrar la explicación de cómo el usuario debe manejar la aplicación para poder ver la imagen emitida por la webcam. En esta función se crea una variable local para la lectura de una tecla que permitirá la ejecución o cierre del programa MJPG Streamer.

- `teclacamara`: variable de tipo char utilizada para la lectura de teclado para la elección de ejecución o cierre del programa MJPG Streamer.

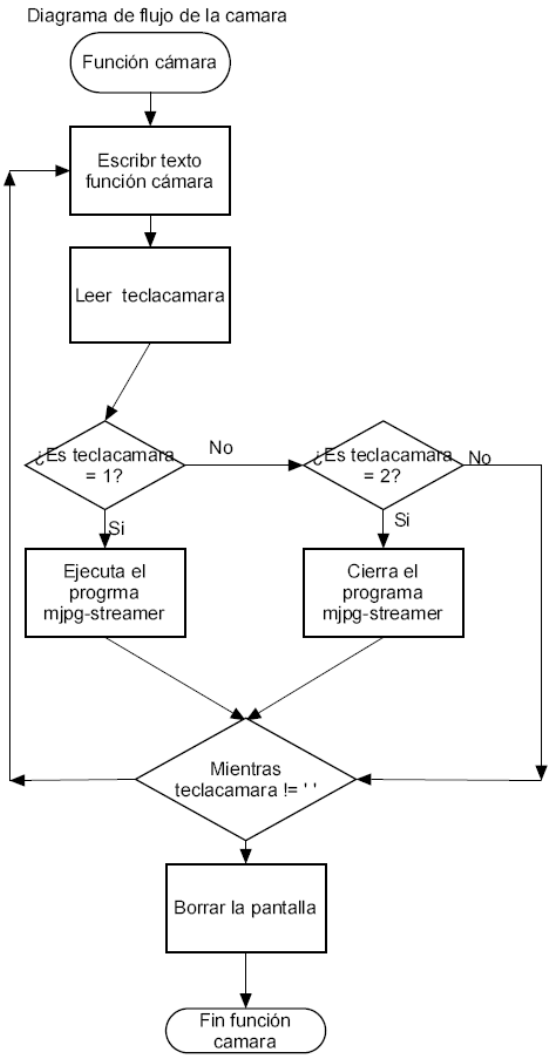
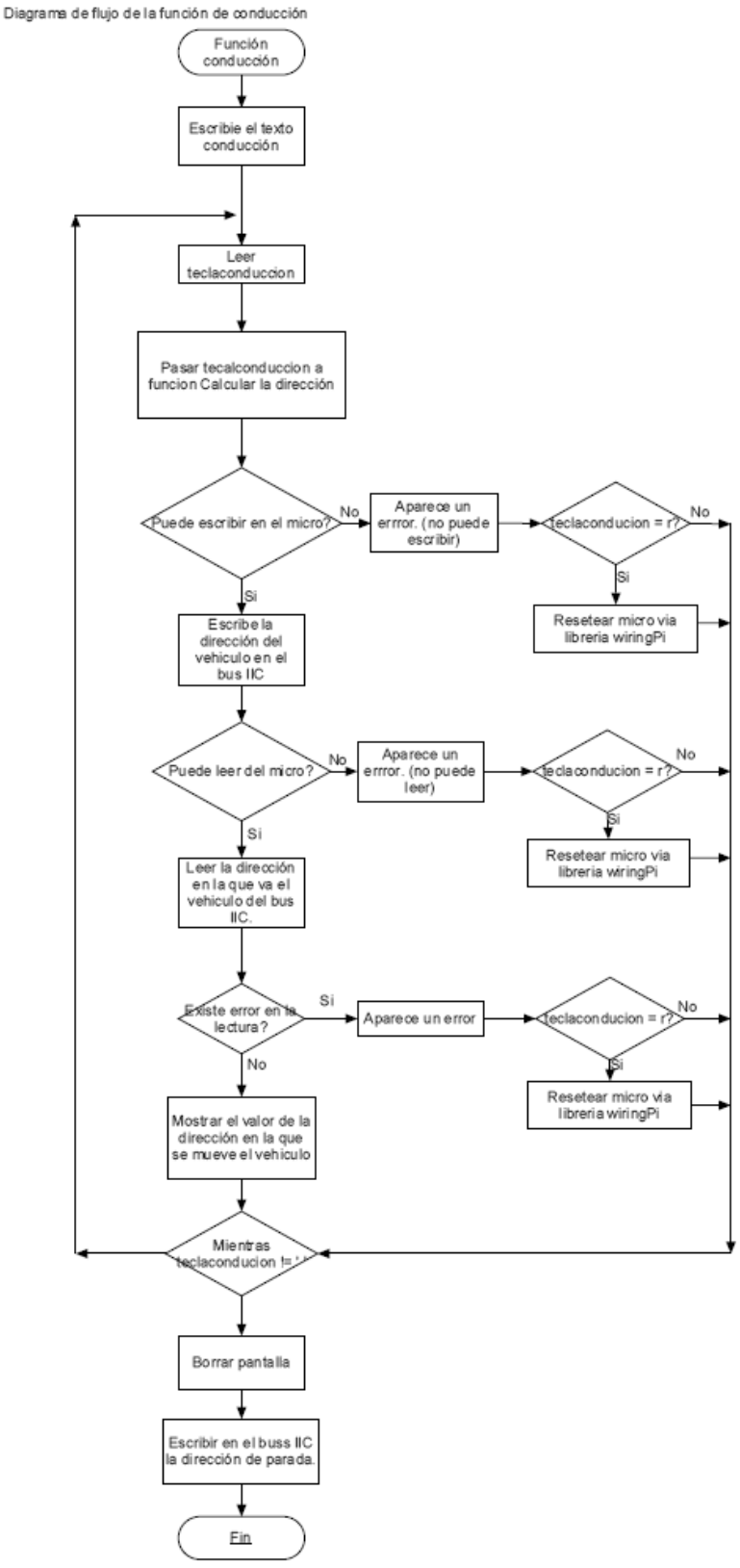


Diagrama de flujo de la camara
Función cámara
Escribr texto función cámara
Leer teclacamara
¿Es teclacamara = 1?
No
¿Es teclacamara = 2?
No
Si
Si
Ejecuta el progrma mjpg-streamer
Cierra el programa mjpg-streamer
Mientras teclacamara != ' '
Borrar la pantalla
Fin función camara

La función de conducción es la función más importante



La función de conducción calcula un valor mediante la sentencia `switch` de programación C, según la tecla leída y vuelca este valor en la variable `buf[]` para cuando se produzca una operación de escritura en el bus IIC.

```
switch(teclaconduccion){
        case 'w':
         direccion=1;
         buf[0]=direccion;
         break;
        case 's':
         direccion=0;
         buf[0]=direccion;
         break;
        case 'a':
         direccion=3;
         buf[0]=direccion;
         break;
        case 'd':
         direccion=2;
         buf[0]=direccion;
         break;
        }
```

La librería `wiringPi.h` es una librería de acceso a GPIO de la Raspberry que permite programar y configurar los pines GPIO. Para resetear el microcontrolador via la librería `wiringPi.h` se manda un pulso mediante un pin de GPIO si la tecla leída por teclado es "r", y para ello se utilizan las siguientes funciones de dicha librería:

En este proyecto se utilizan las siguientes funciones de esta librería:

- `wiringPiSetupGpio`: Inicializa GPIO con la librería wiringPi.
- `pinMode(pin, OUTPUT)`: Función en la que se selecciona el pin deseado como salida.
- `digitalWrite(pin,1)`: Función con la que se pone el pin deseado en un estado lógico alto.
- `digitalWrite(pin,0)`:Función con la que se pone el pin deseado en un estado lógico bajo.

El programa principal del microcontrolador comienza con la inicialización de las siguientes librerías:

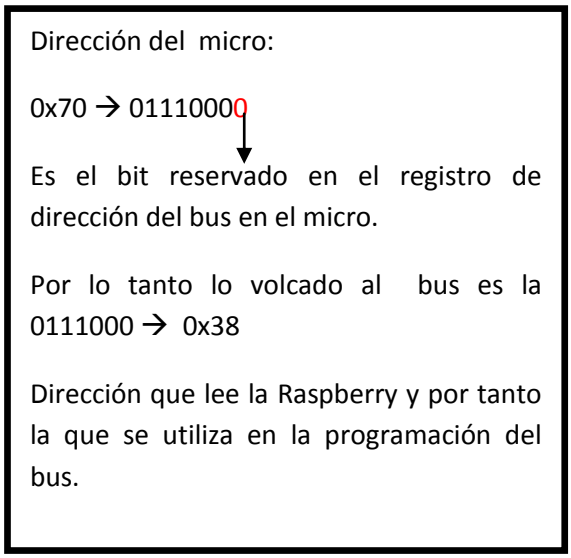
- `#include <hidef.h>` utilizada para poder habilitar interrupciones.
- `#include <MC9S08QG8.h>` utilizada para la declaración puertos del micro.

Acontinuacion se definen las constantes:

- `#define SLV_ADDRESS 0x70`

Se observa que la dirección no coincide con la leída por la Raspberry. La dirección en el bus IIC del MC9S08QG8 en este proyecto no coincide con la que lee la Raspberry Pi, ya que la dirección en el microcontrolador es de 8 bits y la que se vuelca en el bus es de 7 bits, esto se debe porque el bit 0 del registro de dirección del bus en el micro es un bit reservado que no se

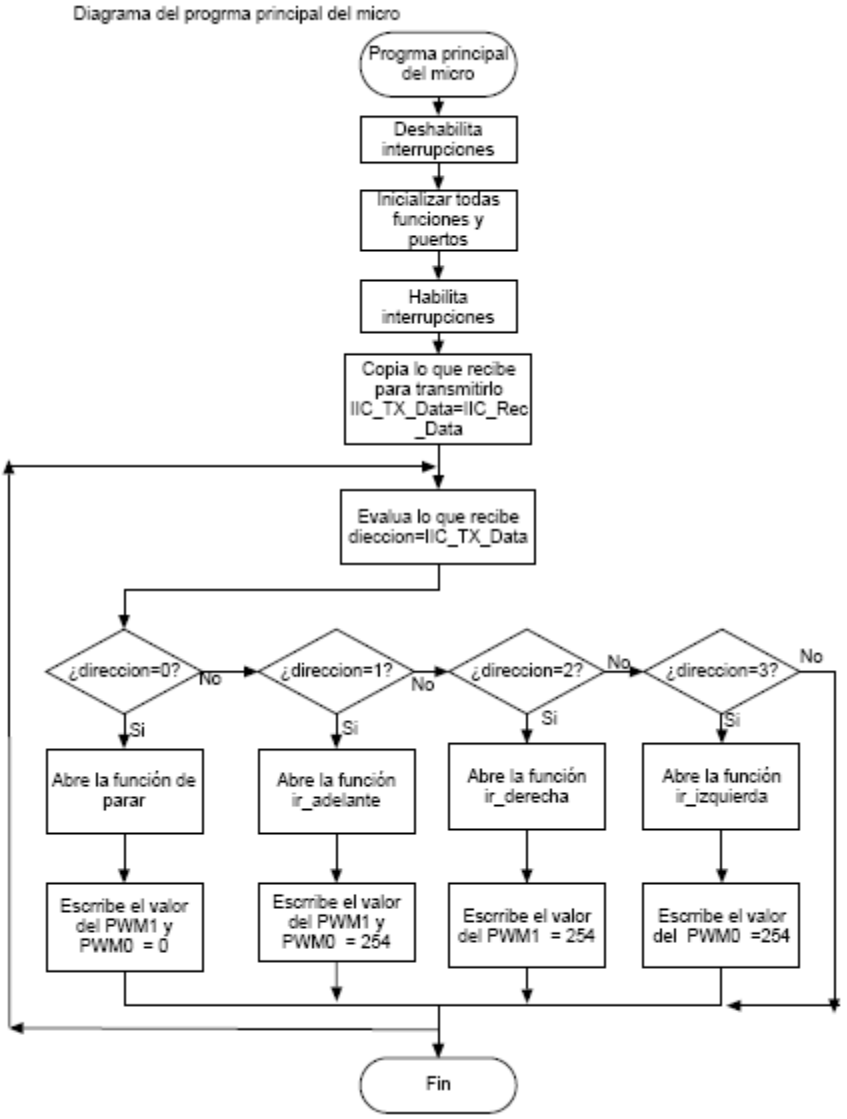
vuelca al bus. Esto se tiene en cuenta a la hora de programar la dirección del micro en la Raspberry Pi.



Definición de variables globales en la programación del microcontrolador:

- `count`: variable de tipo char utilizada para contar los datos transmitidos al bus IIC.
- `rec_count`: variable de tipo char utilizada para contar los datos recibidos del bus IIC.
- `num_to_rec`: variable de tipo char utilizada para saber el máximo de datos recibidos por el bus IIC.
- `IIC_Rec_Data[1]`: variable de tipo cadena utilizada para guardar el dato recibido del bus IIC.
- `IIC_TX_Data[1]`: variable de tipo cadena utilizada para guardar el dato transmitido al bus IIC.
- `Val_1[1]`: variable de tipo cadena utilizada para dar valor al PWM1.
- `Val_0[1]`: variable de tipo cadena utilizada para dar valor al PWM0.
- `direccion[1]`:variable de tipo cadena utilizada para evaluar la dirección en la que se tiene que mover el vehículo.

La función principal se explica con el siguiente diagrama de flujo:



La inicialización de los puertos y algunas funciones se realiza mediante la función `startup()`, donde las líneas más importantes son:

- `SOPT1_COPE=0;` Inhabilita el COP.
- `SOPT2_IICPS=0;` Configura el micro para que SDA del bus IIC este en PTA2 y SCL en PTA3.
- `PTAPE_PTAPE4=0`; Resistencia de pull up interna.
- `PTAPE_PTAPE5=0`; Resistencia de pull up interna.
- `IICA=SLV_ADDRESS`; Dota de dirección al micro para el bus IIC.
- `IICC=0xc0`; Habilita la interrupción del IIC.
- `IICC_MST=0`; Selecciona el micro como esclavo.
- `Init_PWM_0();` Función que inicializa el PWM.

La configuración de los registros de Timer para el funcionamiento de los dos canales es la siguiente:

- El registro de control del timer:

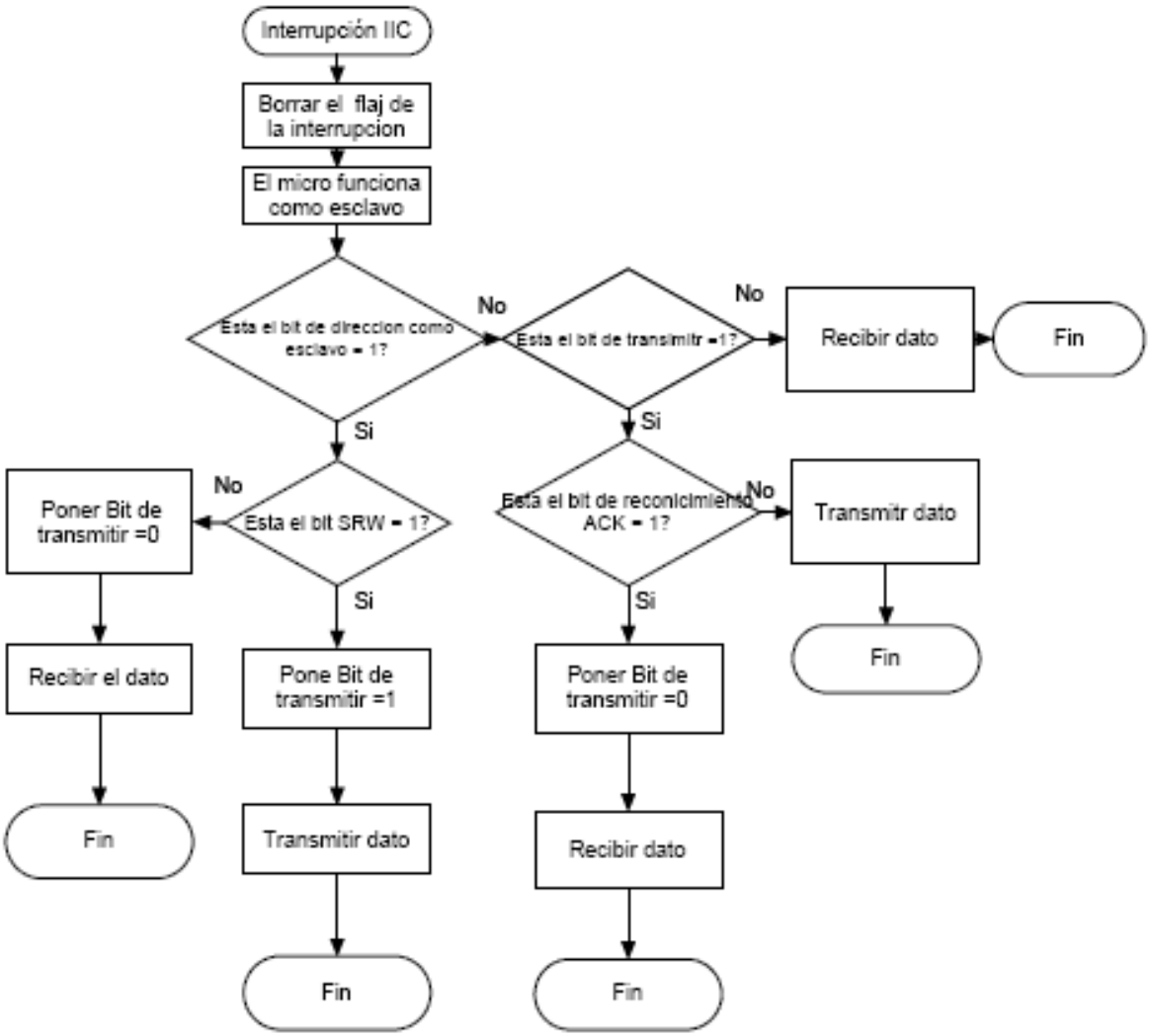
TOF	TOIE	CPWMS	CLKSB	CLKSA	PS0	PS1	PS2
0	1	0	0	1	1	1	1
Inicialmente el contador no ha alcanzado el valor o no hay sobrepaso	Habilita interrupciones	El TPM funciona como output-compare input – compare según se configure MSnA:MSnB	El timer funciona con el bus clock		Selecciona un preescalado de dividir 128 entre la frecuencia del reloj. La resolución es de 128/16=8ms. El desbordamiento es de 524 ms.		

- El registro de control del canal n:

CHnF	CHnIE	MSnB	MSnA	ELSnB	ELSnA		
0	1	1	0	1	0	0	0
Inicializa que no ha aparecido un evento	Habilita interrupciones en el canal	Configura el timer en modo Edge-aligned PWM y lo configura para funcionar como High-true Pulses(clear output on compare)					

El microcontrolador gestiona por interrupciones tanto el IIC como los dos canales del PWM. La interrupción del IIC en la librería del micro es la número 17 y es la siguiente.

Diagrama de flujo de la interrupcion del I2C del micro



El microcontrolador gestiona tres interrupciones más a parte de la del IIC. La primera es la interrupción de sobrepaso del Timer y es la interrupción número 7, donde se borra el flag del registro de control del Timer para que se pueda activar de nuevo la interrupción.

```
interrupt 7 void overflow_timer(void){

        TPMSC_TOF=0;
}
```

La segunda es la interrupción del canal 1 del Timer, donde se borra el flag del registro de control del canal1 para que se pueda activar de nuevo la interrupción. Esta interrupción es la número 6.

```
interrupt 6 void PWM_canal1(void){

        TPMC1SC_CH1F=0;
}
```

Y por último, la interrupción del canal 0 del Timer, donde se borra el flag del registro de control del canal0 para que se pueda activar de nuevo la interrupción. Esta interrupción es la número 5.

```
interrupt 5 void PWM_canal0(void){

        TPMC0SC_CH0F=0;
}
```

En el diseño software cabe destacar la importancia de la comunicación entre el MC9S08QG8 y la Raspberry Pi via bus IIC, por ello a continuación se explica el funcionamiento del bus en el proyecto.

- Lo primero, mediante programación se abre el puerto IIC 1 de la Raspberry Pi, para ello se utiliza la función `open("dev/i2c-1", 0_RDWR)`.
- Después se selecciona el dispositivo a unir al bus mediante su dirección y se selecciona como esclavo mediante la función `ioctl(fd, I2C_SLAVE, addr)`.
- Cuando la aplicación lo requiere la Raspberry inicia una operación de escritura en el bus donde mediante la variable `buf` vuelca el dato a transmitir, esto se realiza mediante la función: `write(fd,buf,1)`, es en este punto cuando la Raspberry como maestro pone el bit R/W del bus con un nivel lógico bajo.
- Así se dispara el flag de la interrupción en el micro, que será puesto a 1 en la interrupción.
- En la interrupción se vuelca el contenido de la variable `buf` de la Raspberry en el registro IICD del microcontrolador. Y este a su vez se vuelca en la variable `direccion[]`.El bus IIC quedara libre cuando se termine de transmitir el dato.
- La Raspberry comienza una operación de lectura del bus y el valor leído lo guardara en la variable `buf`, para ello utiliza la función: `read(fd,buf,1)`. La Raspberry con esta función pone el bit R/W del bus con un nivel lógico alto.
- Así se dispara el flag de la interrupción en el micro, que será puesto a 1 en la interrupción.
- En la interrupción se vuelca el contenido de la variable `direccion[]` del micro en el registro IICD del microcontrolador.
- Comienza la transmisión de datos por el bus IIC volcando los datos en el variable `buf`.
- El bus IIC quedara libre cuando se termine de transmitir el dato, hasta que la Raspberry no comience otra operación de escritura o de lectura.
- Las funciones utilizadas en la Raspberry para el funcionamiento del bus IIC se encuentran en las librerías `<linux/i2c-dev.h>`, `<sys/ioctl.h>` y `<sys/typs.h>`

3.3 Configuración MJPG-Streamer

La instalación del MJPG-Streamer solo requiere de unos pocos pasos ejecutados en la Raspberry Pi, son los siguientes:

```
apt-get install libjpeg8-dev imagemagick subversion
cd /usr/src/
svn checkout svn://svn.code.sf.net/p/mjpg-streamer/code/ mjpg-
streamer-code
cd mjpg-streamer/mjpg-streamer
make
```

- La instalación consiste en la instalación de las librerías `libjpeg8-dev` e `imagemagick` y la aplicación `subversion`.
- Abrir el directorio `src`.
- Descargar mediante la aplicación subversión el código de ejecución del MJPG Streamer.
- Compilar este código en la carpeta que se crea `mjpg-streamer`.

Para la ejecución del programa MJPG-Streamer la aplicación mediante la función `system()` ejecuta un script de Linux que es el siguiente:

```
#!/bin/sh

PLUGINPATH=/usr/src/mjpg-streamer-code/mjpg-streamer
STREAMER=$PLUGINPATH/mjpg_streamer
DEVICE=/dev/video0
RESOLUTION=320x240
FRAMERATE=25
HTTP_PORT=8001

# check for existing webcam device
if [ ! -e "/dev/video0" ]; then
echo "stream.sh: Error - NO /dev/video0 device" 2>&1 | logger
exit 2
fi


$STREAMER -i "$PLUGINPATH/input_uvc.so -y -n -d $DEVICE -r $RESOLUTION
-f $FRAMERATE" -o "$PLUGINPATH/output_http.so -n -p $HTTP_PORT" -b
```

En este script se da valores a las variables de configuración del programa tanto de parámetros de entrada como de salida que permitirán la ejecución del MJPG Streamer a través del explorador de Internet.

- PLUGINPATH es la ruta donde se encuentra instalado el programa.
- STREAMER es el archivo de ejecución del programa.
- DEVICE es la ruta donde se encuentra la webcam.
- RESOLUTION es la resolución de la ventana donde se verá la imagen en el explorador, se elige esta resolución porque es adecuada para la visión de las imágenes y no sobrecarga el funcionamiento de la Raspberry.

- FRAMERATE son el número de imágenes que se imiten por segundo, se elige este número de imágenes porque es adecuada para la visión de un video y no sobrecarga el funcionamiento de la Raspberry.
- HTTP_PORT es el puerto por el que se emiten las imágenes.

Lo siguiente es la comprobación de que la webcam esa conectada y por último se ejecuta el programa con los parámetros elegidos anteriormente.

Para cerrar el programa MJPG-Streamer , la aplicación ejecuta otro script mediante la función *system()* que es el siguente:

```
#!/bin/sh
sudo kill `pidof mjpg_streamer`
```

Se utiliza la función de Linux "kill" para cerrar el programa.

3.4 Diseño mecánico del vehículo.

Para la construcción de la estructura mecánica se han realizado seis partes de las que se compone el chasis.

La primera es la parte inferior del chasis, es una pieza rectangular de 150mm x 80mm donde van anclados el conjunto motor-reductor, los zócalos de las baterías, los soportes para las ruedas delanteras y los soportes de la parte inferior.

Los soportes para las ruedas delanteras son dos piezas de 30mm x 30mm, donde su función es unir las ruedas delanteras y la parte inferior del chasis. Las ruedas van ancladas mediante tornillos y para la unión de estos soportes al chasis inferior se han construidos dos pequeños ángulos que se atornillan a los soportes y al chasis inferior.

Los soportes de la parte inferior son dos piezas de 100mm x 30mm, que permiten unir las ruedas inferiores al chasis inferior. Las ruedas van ancladas mediante tornillos y para la unión de estos soportes al chasis inferior se han construidos dos pequeños ángulos que se atornillan a los soportes y al chasis inferior.

Por último, la parte superior del chasis es una pieza de 170mm x 12mm, esta pieza sirve para anclar la Raspberry Pi, la webcam y la placa protoboar. Para la unión de la parte inferior del chasis con la parte superior se utiliza cuatro barrillas roscadas de métrica 4, que mediante tuercas permite ajustar la altura y que las dos partes queden bien unidas.

A continuación se muestra los cuatro tipos de piezas que se necesitan para la construcción de la estructura mecánica.

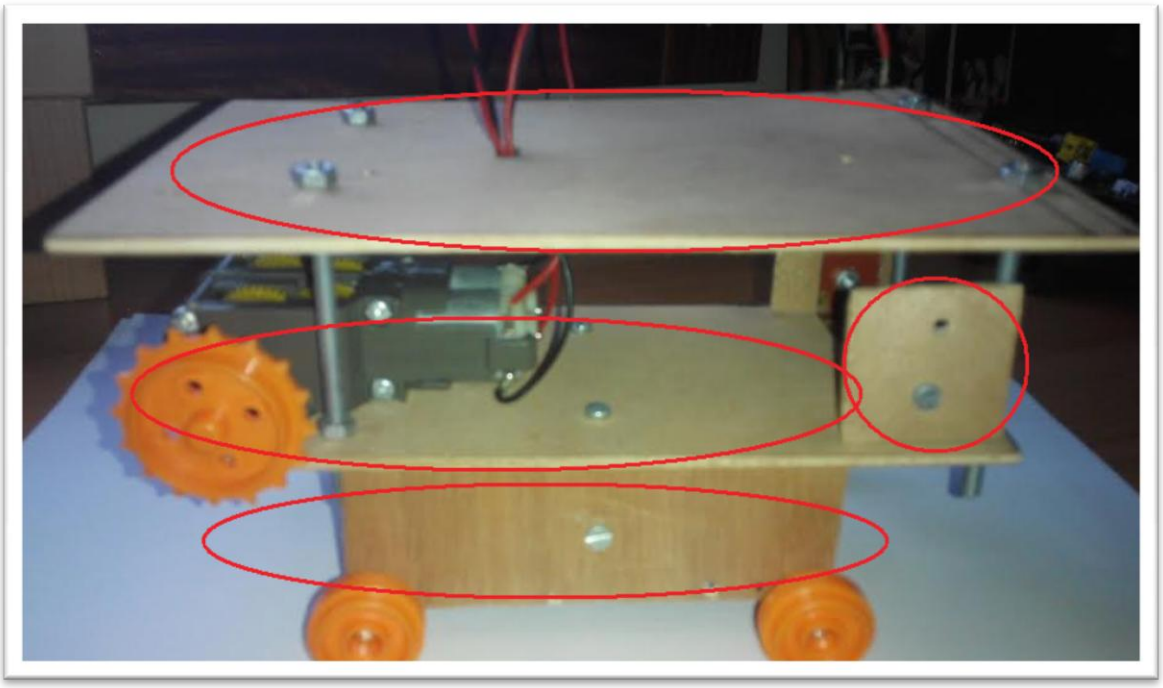


Ilustración 5: Piezas para la construcción del chasis

Se ha diseñado 3 planos donde se puede ver las medidas y las distancias de los agujeros de los 4 tipos de piezas que conforman la estructura mecánica del prototipo.

3.5 Construcción del prototipo.

Componentes necesarios para la construcción de prototipo.

Material	Cantidad
Placa protoboard	1
Raspberry Pi	1
Dongle wifi	1
Micro MC9S08QG8	1
Integrado L292D	1
Regulador 78T05CT	1
Resistencias	4
Tornillos Aluminio chasis	4
Track and Wheel set	1
Conjunto motor reductor	1
Zócalo baterías	2
Baterías AA LR6	12
Webcam	1
Marquetería	1
Tornillos con tuerca m3	12
Cableado	

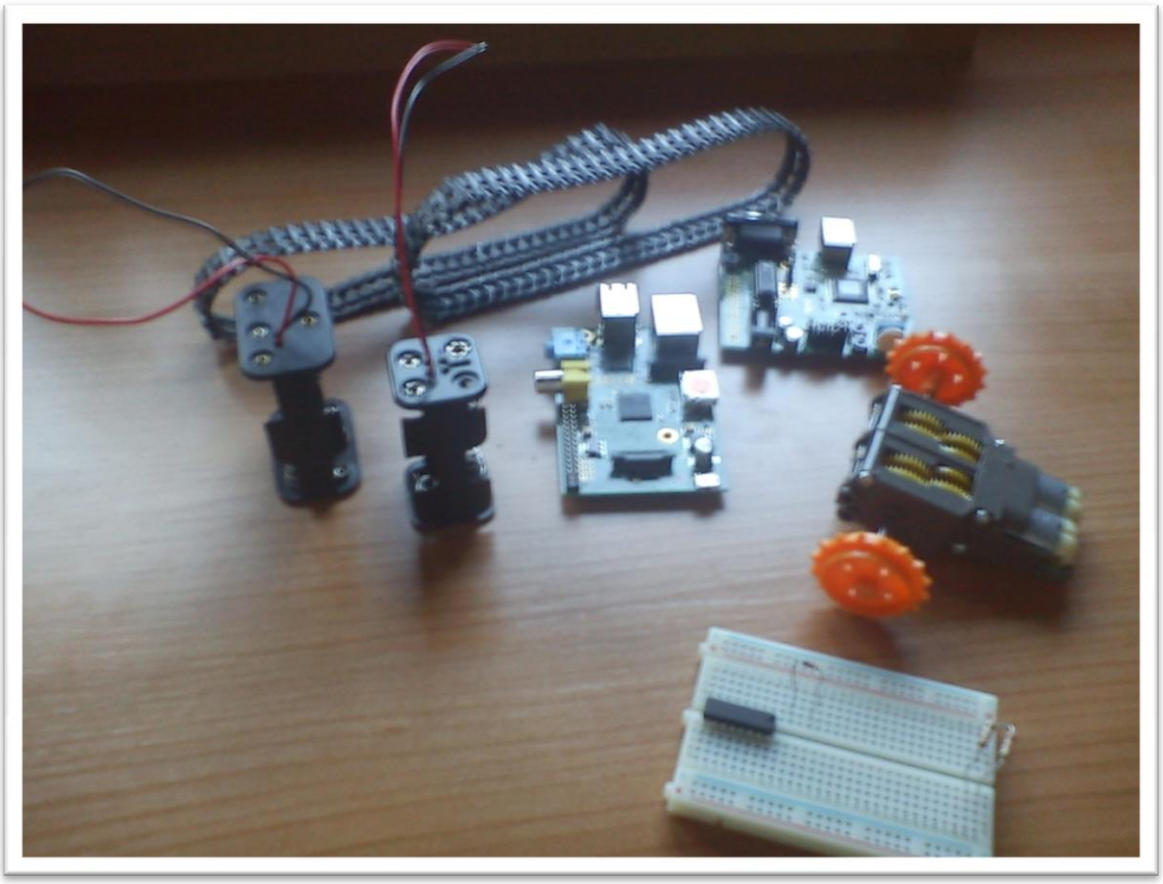


Ilustración 6: Algunas de las piezas utilizadas

El primer paso para la construcción de la maqueta es el montaje de las orugas, para se separan del plástico tanto las ruedas como las parte de las orugas que se unirán de forma que formen la oruga más larga posible. El siguiente paso, es el montaje del conjunto motor reductor donde lo único que se necesita es un pequeño destornillador de estrella y una llave allen que viene en la caja de este que se pueda montar. Se pueden montar de dos formas para tener una relación de transmisión de 58:1 o 203:1. En este proyecto se ha montado con una relación 203:1 debido a que el peso del prototipo no permite montar la otra relación y que el vehículo se mueva de forma adecuada.

Ilustración 7: Conjunto motor-reductor.

El siguiente paso en la construcción de la maqueta fue la construcción de chasis con marquetería, para ello se dibujo las piezas en una plancha de marquetería 300mm x 400mm. Después se recortaron las piezas y se taladraron los agujeros para la colocación de los torillos y las varillas.

A continuación se montaron todas las piezas para la construcción definitiva de la estructura mecánica del prototipo.

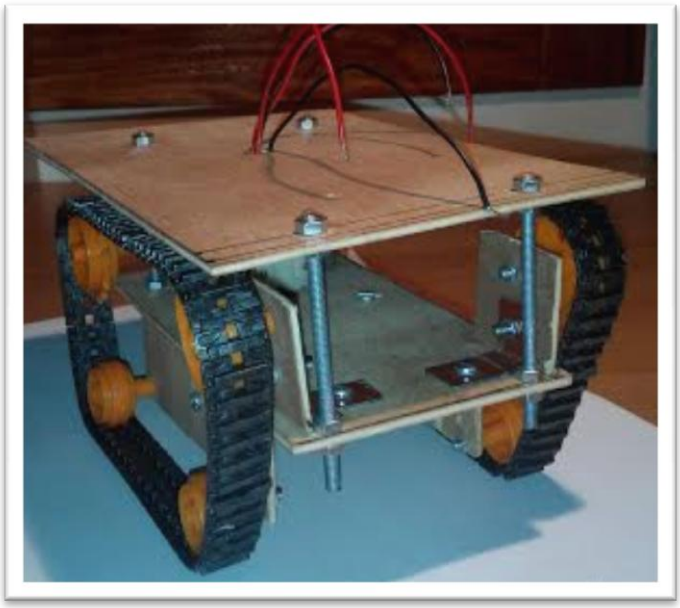


Ilustración 8: Montaje del prototipo.

Los primeros movimientos del prototipo se realizan mediante la programación del microcontrolador en su tarjeta de programación. Donde se programa el microcontrolador en IDE Code Warrior, la programación consiste en la inicialización de los puertos y las interrupciones de los PWM que controlaran los motores, así como un sencillo programa principal que transfiere un valor constante a los PWM1 y PWM0 que activa sus correspondientes pines y estos son transmitidos a los del integrado L293D que hace que los motores se pongan en marcha. De momento el prototipo, en esta fase del proyecto, no podía cambiar su dirección, solo se movía hacia delante.

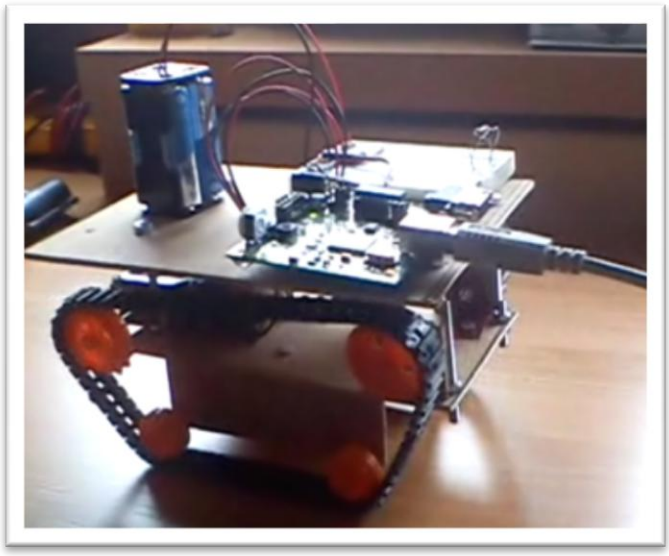


Ilustración 9: Primeros movimientos con el microcontrolador.

El siguiente paso fue la adhesión de Raspberry Pi al prototipo. Este paso implicaba el control del prototipo desde la Raspberry Pi, lo cual suponía que había que conectar mediante algún tipo de comunicación la Raspberry Pi con el microcontrolador. Para ello se eligió la conexión vida bus IIC ya que ambos elementos disponían de él, este paso era fundamental ya que la Raspberry, desde el principio iba a ser el maestro en este proyecto.

Se ha conectado la Raspberry Pi mediante cable Ethernet, para así acceder a ella mediante terminal. En este paso se programa una interfaz con el usuario y la programación del bus IIC tanto en la Raspberry como en el microcontrolador, además se programa mediante la librería `ncurses.h` la lectura de una tecla sin tener que esperar a pulsar “intro”. Asi como la impresión en pantalla de texto en color que permite que los errores sean destacados en el modo texto en la consola.

Todo esto facilita el control del movimiento de vehículo por parte del usuario, con las limitaciones de tener que estar conectado a Internet mediante cable. Por esto, el siguiente paso fue la desconexión del cable Ethernet para así conectar la Raspberry a la red wifi.

La conexión de la Raspberry a la wifi se realizo mediante la consola modificando el archivo “interface”, donde se introduce el ssid de la red y la psk (password).

La conexión de la Raspberry a la red wifi mediante un dongle wifi, supondrá que al desconectarla de la red eléctrica el regulador de tensión a utilizar tendrá que suministrar al menos 1.5 amperios para que el dongle wifi funcione de forma correcta.

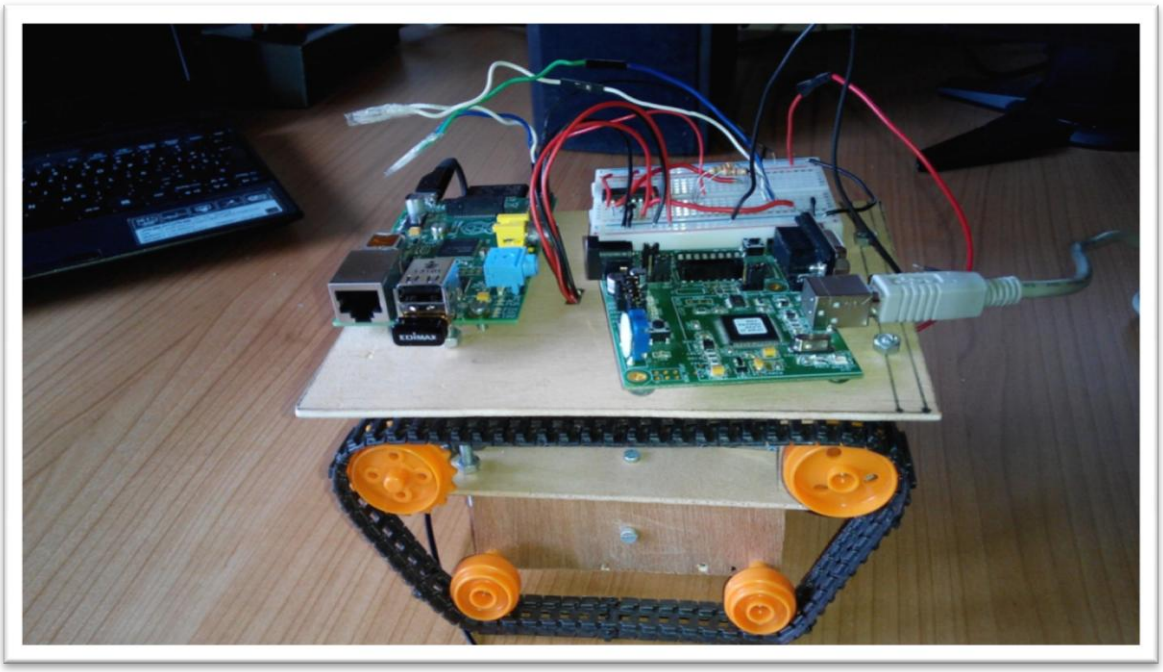


Ilustración 10: Conexión wifi del prototipo.

El siguiente paso fue la desconexión de la red eléctrica de la Raspberry Pi y de la alimentación del microcontrolador mediante el cable USB fuera de la placa de programación del mismo.

La alimentación de la Raspberry Pi se ha realizado 6 baterías tipo AA LR6, colocadas en un zócalo para baterías. Para el ajuste la tensión se utiliza el regulador de tensión MC78T05CT, se eligió este regulador por los 3 amperios que suministra ya que los puertos USB destinados a la webcam y el dongle wifi necesitan al menos 1.5A.

El microcontrolador se extrae de forma sencilla de su placa de evaluación y se pincha sobre la placa protoboard. La alimentación del microcontrolador es sencilla, solo hay que conectar la patilla Vdd a 3v y Vss a GND.

Por ultimo la fabricación de la placa que sustitulla a la placa protoboard, para ello se diseña mediante el programa Eagle.

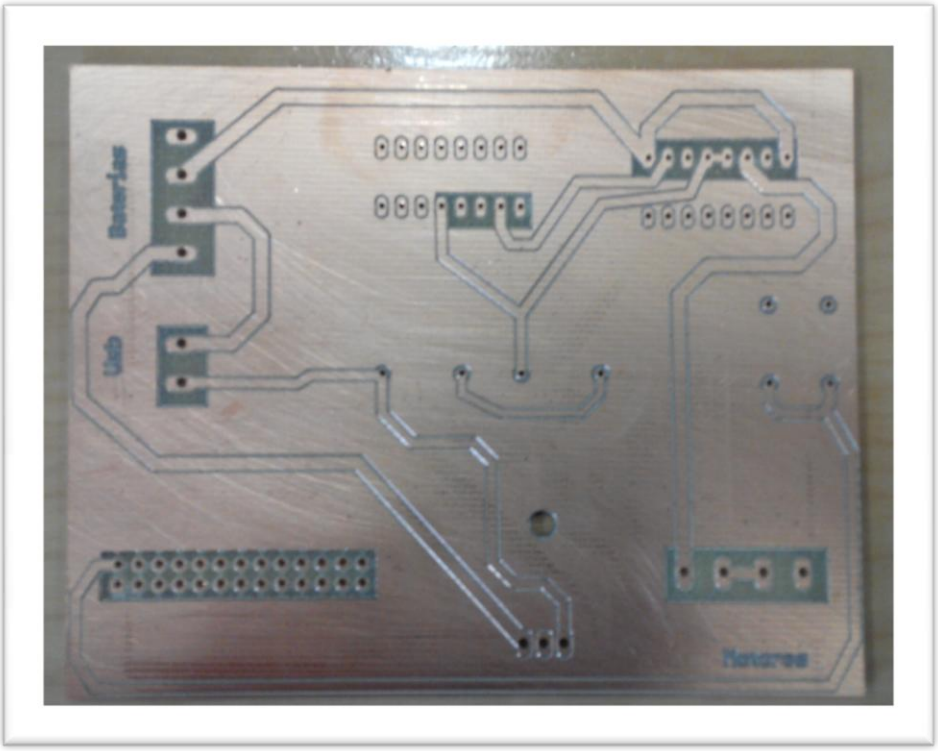


Ilustración 11: Placa sin componentes.

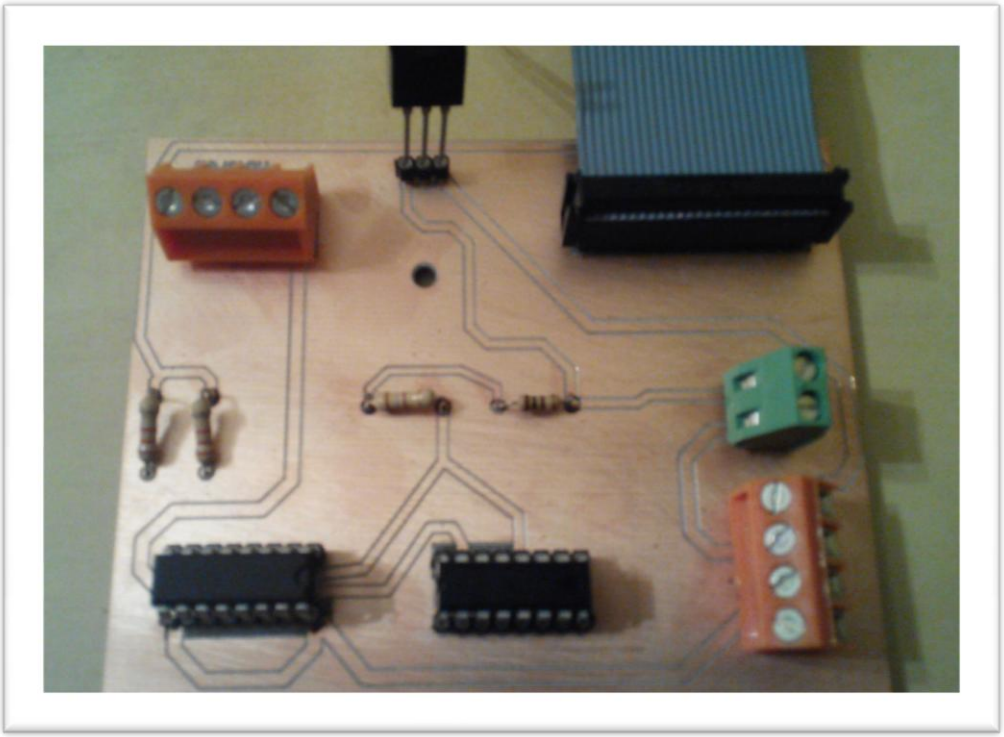


Ilustración 12: Placa con componentes

Por último la conexión de la webcam al prototipo, se realiza conectando la webcam al puerto USB, e instalándola. Además hay que instalar el programa de emisión via streaming.

De esta forma el prototipo estaría terminado. Solo quedaría fijar todos los componentes a la estructura mediante algún tipo de fijador o agarre.

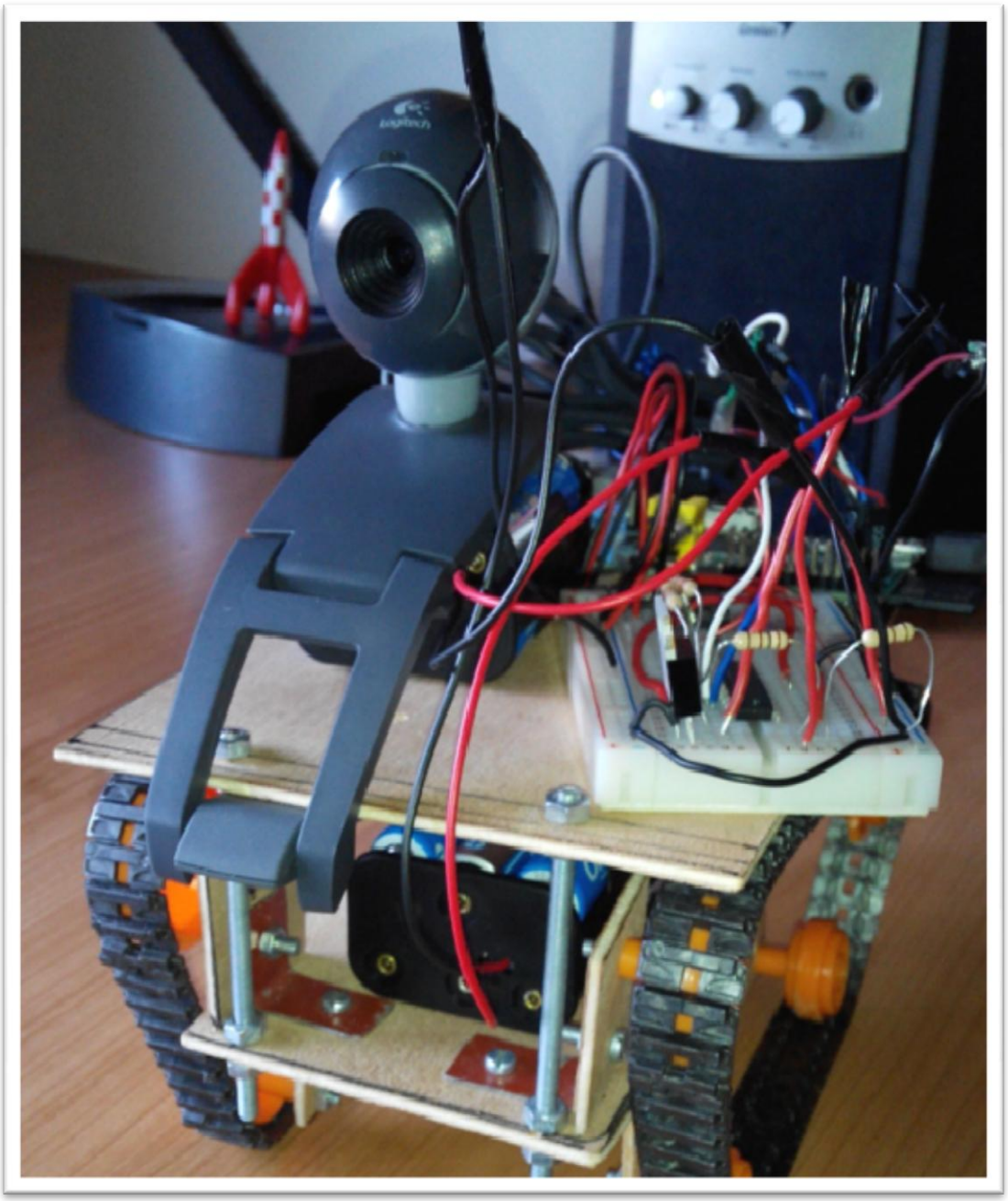


Ilustración 13: Prototipo casi terminado.

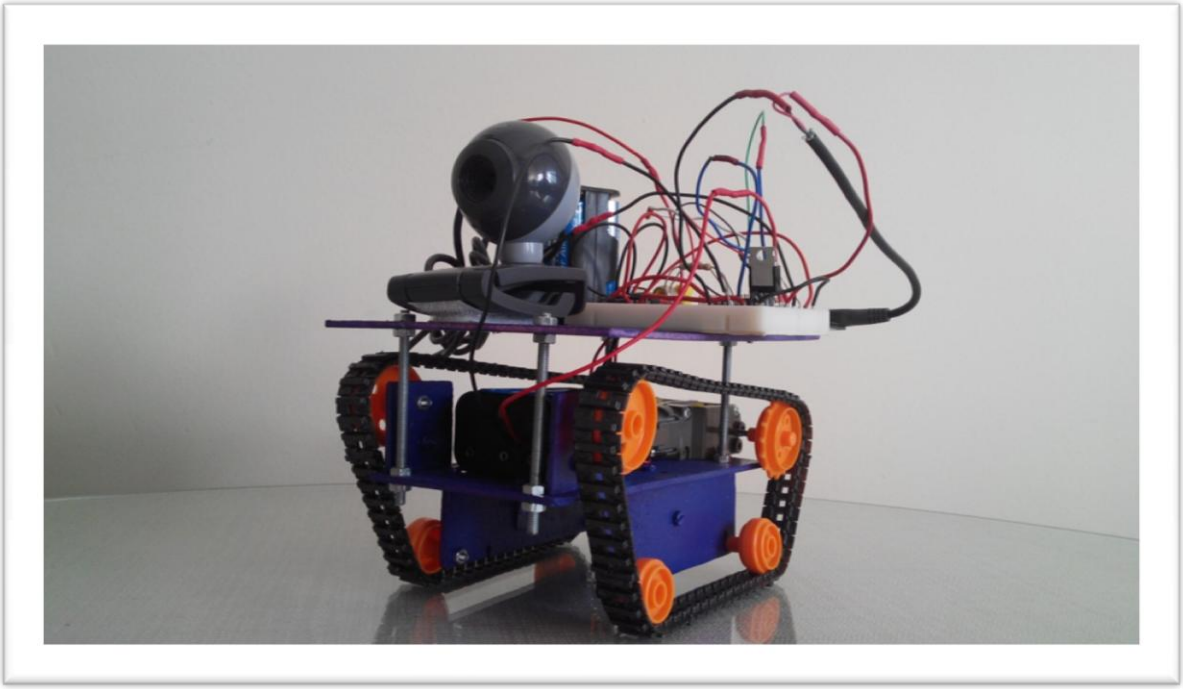


Ilustración 14: Prototipo terminado con placa protoboard.

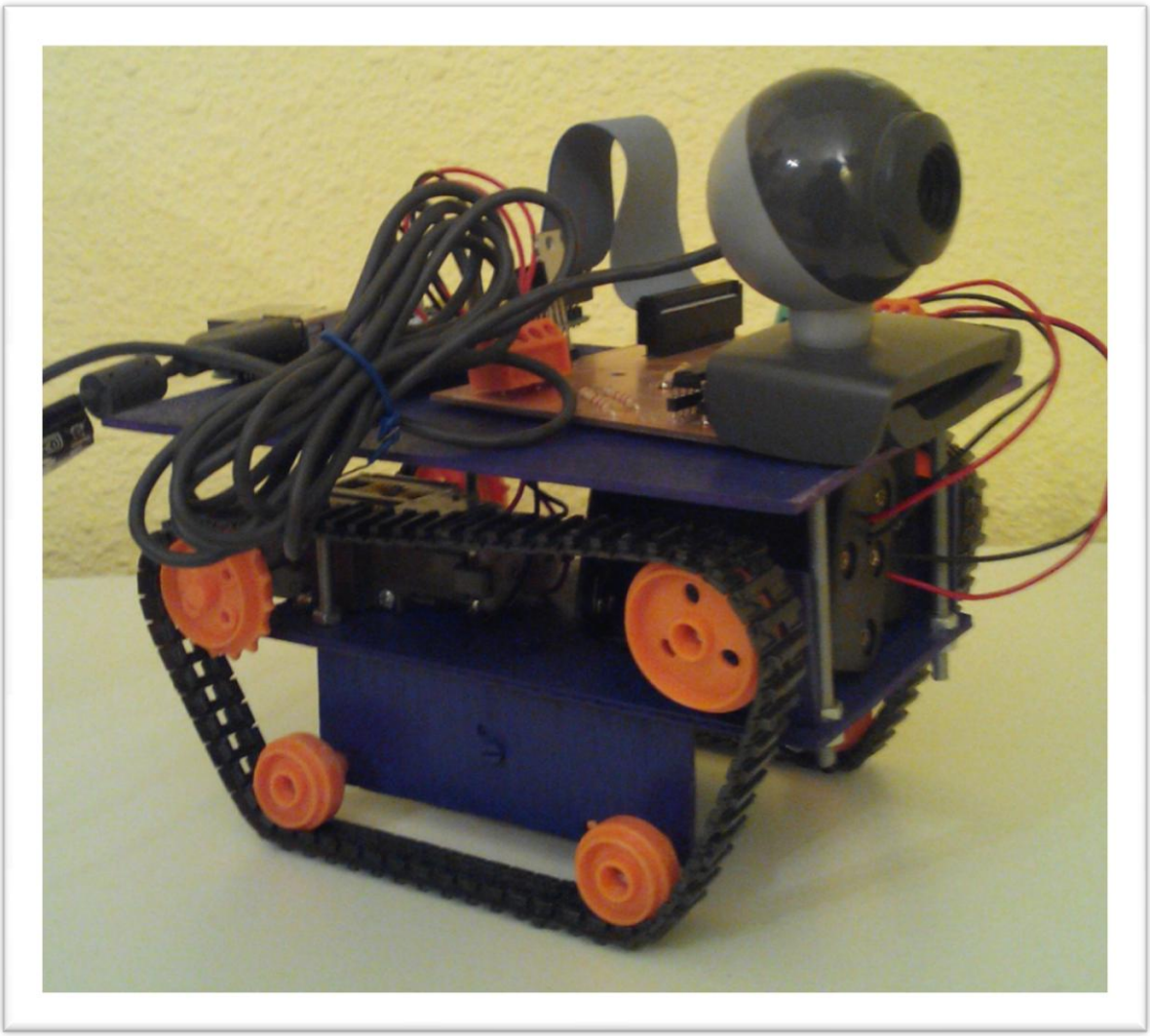


Ilustración 15: Prototipo terminado.

El interfaz hombre maquina se realiza mediante consola y la imagen transmitida vía streaming se ve en el explorador de internet. En la aplicación parecen tres pantallas desde donde se controla el vehículo.

1. Pantalla principal:

En la pantalla principal aparece una pequeña explicación de lo que consiste este proyecto y el menú de selección. Según la tecla que se pulse se irá hacia una pantalla u otra. Si se pulsa “1” se abre la pantalla del funcionamiento de la cámara, si se pulsa “2” la pantalla de conducción del vehículo y por ultimo si se pulsa la barra espaciadora se sale de la aplicación.

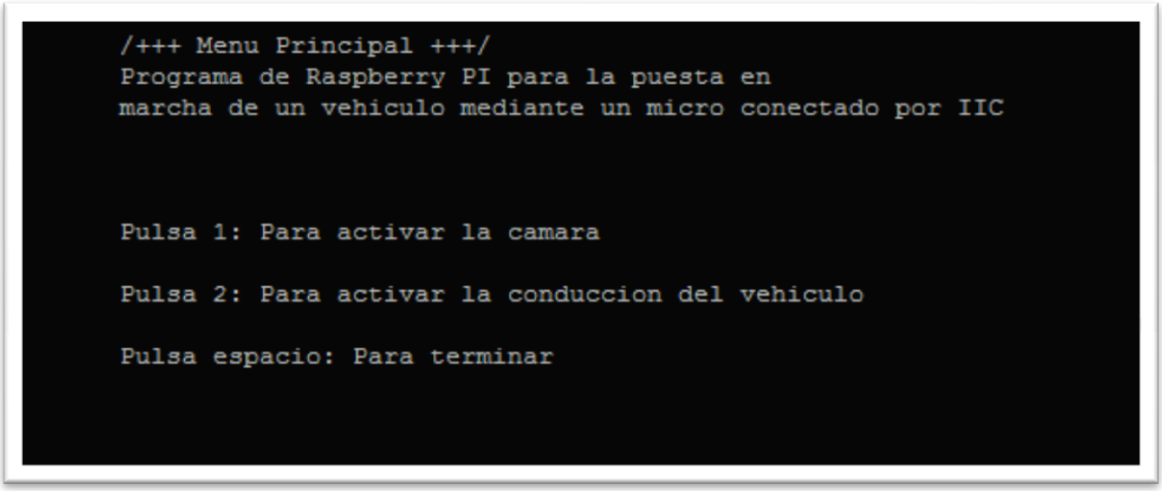


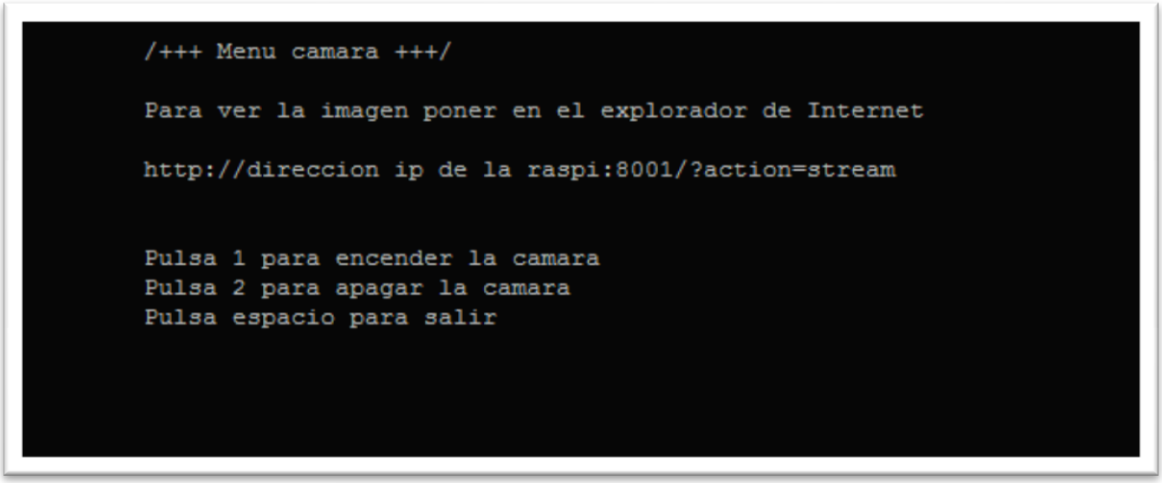
Ilustración 16: Consola: Pantalla principal.

2. Pantalla de funcionamiento de la cámara:

En la pantalla de la cámara se puede seleccionar mediante una selección de teclas, permiten ejecutar el programa MJPG Streamer o cerrar dicho programa. Si se pulsa “1” se ejecuta el programa MJPG Streamer en un segundo plano lo que permite seguir controlando la Raspberry Pi. Es en este punto cuando el usuario puede abrir el explorador de internet para poder ver la imagen que está emitiendo el vehículo, el usuario solo tiene que abrir la siguiente página web para ver la imagen “http://direccion IP de la Raspberry:8001/?action=stream”.

Si se pulsa “2” se cierra este programa, la imagen se deja de transmitir y por tanto el usuario no ve ninguna imagen.

Si se pulsa la barra espaciadora se regresa al menú principal.



3. Pantalla de conducción:

En la pantalla de conducción aparece una explicación de cómo utilizar el vehículo de forma correcta, la conducción se realiza mediante teclado utilizando las teclas siguientes.

Para ir hacia delante hay que pulsar la tecla “w” lo que el programa entiende como la dirección del vehículo número 1, la Raspberry Pi envía mediante IIC al micro un 1, en el programa principal del microcontrolador pone el valor del PWM al máximo en el PWM0 y PWM1, de esta forma las dos salidas del integrado L293D se activan y los dos motores se mueven a la vez, provocando el movimiento del vehículo.

Para parar el vehículo se pulsa la tecla “s”, en este caso la dirección del vehículo es la número 0. El proceso es igual que el caso anterior pero en este caso el valor del PWM es 0, y el micro escribe este valor en el PWM0 y PWM1.

Si el vehículo se tiene que mover a la derecha se pulsa la tecla “d”, lo que le programa entiende como dirección del vehículo número 3. Proceso de envío de datos es igual que en los casos anteriores pero en este caso el micro pone el valor máximo del PWM al PWM1.

Si el vehículo se tiene que mover a la izquierda se pulsa la tecla “a”, lo que le programa entiende como dirección del vehículo número 2. Proceso de envío de datos es igual que en los casos anteriores pero en este caso el micro pone el valor máximo del PWM al PWM0.

Si se pulsa espacio se regresa al menú principal.

En esta pantalla también aparece la lectura que hace la Raspberry del micro para comprobar que la dirección en la que se mueve el vehículo es la deseada del usuario.

La escritura y lectura de datos en el microcontrolador se realiza mediante interrupción, esto significa que en el programa del microcontrolador tiene que estar habilitada la interrupción del bus IIC. Es por esto que la programación de la lectura y escritura no aparece en el menú principal del microcontrolador.

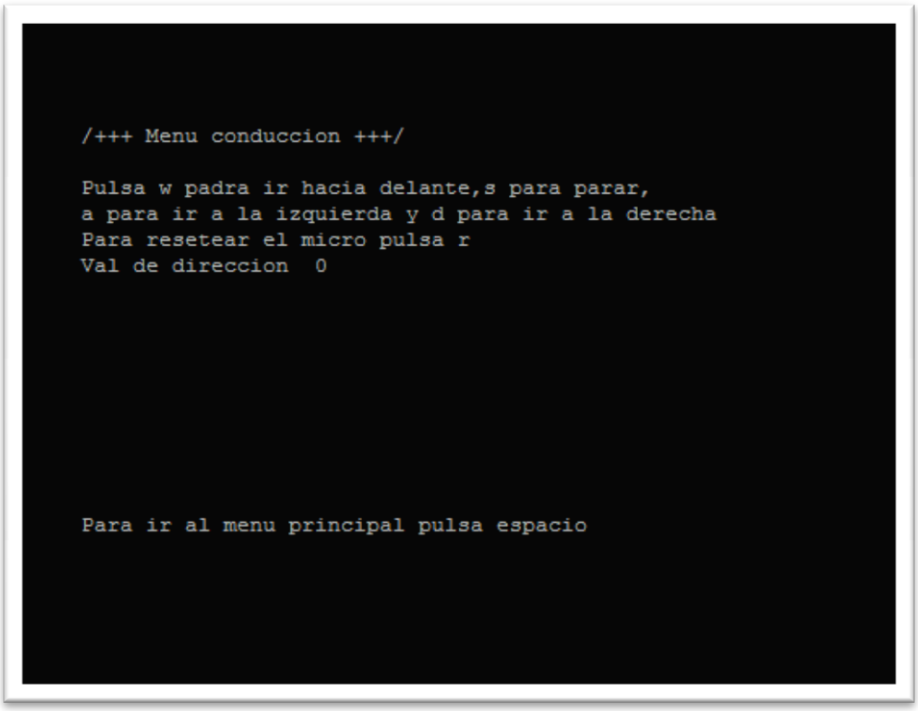


Ilustración 17: Pantalla conducción.

Cuando existe algún error de tipo hardware en la aplicación, aparece un texto en rojo advirtiéndote del tipo de error que es. Son problemas de comunicación ya que o bien la Raspberry Pi no puede escribir en el microcontrolador o bien no puede leer es debido a que le microcontrolador se ha desconectado o se ha apagado, también aparece un error en la lectura del valor de la dirección en el micro. El valor leído es 112, este error es debido a una lectura errónea.

Cuando existe un error de este tipo hay que resetear el microcontrolador o reiniciar la aplicación, ya que hay inicializar los dispositivos de nuevo.

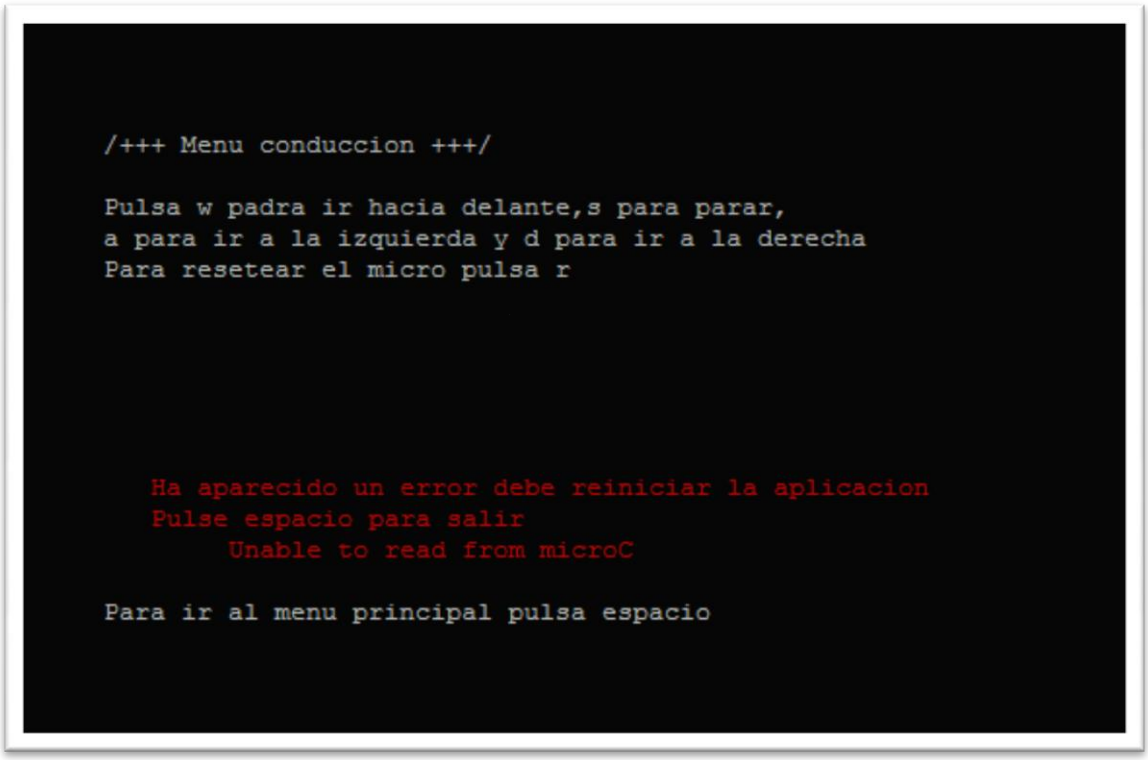


Ilustración 18: Error asociado a la lectora del microcontrolador.

Una interface hombre-máquina adecuada para este proyecto se aleja de los objetivos de este proyecto, debido a que los conocimientos para ello superan los conocimientos adquiridos en el Grado de Ingeniería Electrónica siendo más cercanos a la Ingeniería Informática.

Si se realizara una aplicación de tipo java o android y manteniendo las tres pantallas,las variables con las que tendrían que interactuar las dos aplicaciones serian las siguientes variables:

- `Tecla`: tecla leída en la pantalla principal.
- `Teclacamara`: tecla leída en la pantalla de la cámara.
- `Teclaconduccion`: tecla leída en la pantalla de la conducción.
- `direccion`: valor de la dirección en la que se mueve el vehículo.
- `reset`: variable utilizada para resetear el microcontrolador.
- `error`: variable destina a la detección de errores de comunicaciones en el bus IIC.
- `buf`: utilizada en la escritura y lectura en el bus IIC
- Programa MJPG Streamer para la aparición de las imágenes en la misma aplicación.

La pérdida del control del vehículo una vez puesto en marcha se puede dar por varios motivos:

- Pérdida del control por la pérdida de comunicación entre la Raspberry y el micro para solucionar este problema, se puede reiniciar el micro pulsando la tecla “r” y reiniciar la aplicación para restablecer la configuración inicial.
- Pérdida de control por la pérdida de comunicación entre el usuario y la Raspberry Pi. Este problema se ha solucionado instalando un watchdog en la Raspberry, el watchdog instalado es el “modprobe bcm2708_wdog”, el cual enviara un latido cada 10 segundos y si no recibe esta señal la Raspberry se reiniciara.

3.6 Componentes y costes.

La siguiente tabla los componentes empleados en el proyecto con sus respectivos costes y proveedores. Los componentes fabricados y piezas como cables, tornillos, etc, están incluidos en otros.

Material	Cantidad	Coste unidad(€)	Coste total(€)	Proveedor
Placa protoboard	1	2,02	2,02	Amazon
Raspberry Pi	1	26,06	26,06	Element14
Dongle wifi	1	9,50	9,50	Amazon
Micro MC9S08QG8	1	1,60	1,60	Element14
Integrado L293D	1	2,80	2,80	Element14
Regulador 78T05CT	1	1,96	1,96	Amidata
Resistencias	4	0,01	0,04	Amidata
Track and Wheel set	1	15,00	15,00	Ardumania
Conjunto motor reductor	1	15,00	15,00	Ardumania
Zócalo baterías	2	5,80	11,60	Element14
Baterías AA LR6	12	0.20	2.40	Ferretería
Webcam	1	5,00	5,00	Amazon
Marquetería	900 cm2	1,00	1,00	Ferretería
Otros	1	10,00	10,00	
Total €			**103.98**	

4 Conclusiones.

4.1 Conclusiones del desarrollo del proyecto.

- Los objetivos del proyecto se han alcanzado de manera satisfactoria.
- El tiempo dedicado a este proyecto con los objetivos finalmente alcanzados creo que es el adecuado para la dificultad de este tipo de proyectos.
- La programación C tanto del programa de Raspberry Pi y del microcontrolador resultaron ser, en algunas cuestiones, más difícil de lo planteado en un principio.

- La configuración del programa MJPG Streamer para que se ejecutara en un segundo plano en la Raspberry Pi resulto complicado ya que no se deponía de la suficiente información de la configuración en este modo.
- Los problemas más complicados en cuanto al hardware han sido la alimentación de la Raspberry Pi, el microcontrolador y los motores. La Raspberry Pi porque se necesita un regulador de tensión con varios amperios ya que la alimentación de los puertos USB para el dongle wifi y la webcam lo requieren. La alimentación del microcontrolador no es complicada en sí, pero se ha tenido que buscar una solución al no disponer de un regulador de 3v, la mejor solución era aplicar un divisor de tensión. La alimentación de los motores mediante baterías tipo AA supone un hándicap al proyecto debido al poco tiempo que los motores funcionan a máxima potencia.

4.2 Conclusión del trabajo realizado.

- El diseño mecánico y la construcción del prototipo realizado no es demasiado caro, además de ser sencillo lo que resulta muy interesante en este proyecto.
- El diseño del control del vehículo ha resultado más complicado de lo esperado, al final el control del vehículo es total pero se podría mejorar.
- Una de las ventajas de este proyecto es que se puede acceder a la aplicación desde cualquier dispositivo que disponga de un cliente SSH, esto es gracias a que la Raspberry Pi funciona con Linux y te permite acceder al terminal mediante SSH.
- Las comunicaciones del proyecto son otra de las mejores cosas que tiene, ya que en un cuanto a las comunicaciones vía bus IIC entre la Raspberry Pi y el microcontrolador, son adecuadas para este proyecto, además de funcionar perfectamente. La comunicación wifi en este proyecto es un punto fundamental, ya que es lo que hace que este proyecto pueda funcionar en cualquier lugar siempre que haya conexión wifi.
- La correcta programación del PWM en el microcontrolador tiene en este proyecto una gran importancia por la aplicación de conocimientos aprendidos a lo largo de la carrera. Aunque debido a que se pone el valor máximo a la salida del PWM, se podría haber ejecutado el mismo resultado poniendo una patilla del microcontrolador a uno.
- La retransmisión vía streaming es un éxito, ya que cuando se planteo el proyecto parecía bastante complicado la emisión de imágenes y el control del vehículo funcionando al mismo tiempo. Hay que decir que la emisión vía streaming funciona con un poco de retardo lo que en algunos momentos, degradara en parte la experiencia del usuario, esto es debido a las limitaciones de la webcam y del programa MJPG Streamer.

4.3 Trabajos futuros

- Existen varias cosas a mejorar en este proyecto aunque creo que una de las fundamentales es la creación de una aplicación donde se pueda ver la imagen y la conducción del vehículo en la misma pantalla ya que ahora se ven por separado. Además de conseguir que la aplicación no se vea en modo consola, por otro lado creo que se debería seguir utilizando el cliente SSH, ya que esto permite acceder a la aplicación desde cualquier dispositivo que disponga de un cliente SSH.

- Otra de las cosas a mejor es la integración de los componentes hardware como el microcontrolador, el integrado L293D, el regulador 78T05CT y las resistencias en una placa PCB lo que haría el proyecto más limpio al eliminar parte del cableado.
- En cuanto al prototipo se podría cambiar la alimentación de los motores con otro tipo de baterías para que su funcionamiento fuera más adecuado.
- También se podría generar dos salidas más en el microcontrolador para que el vehículo pudiera ir hacia detrás, pero esto implica la modificación de la aplicación ya que habría que buscar forma de que el vehículo pueda parar.

5 Bibliografía

Información sobre definición de robotica “robotica.wordpress.com

Información sobre Raspberry PI “es.wikipedia.org/wiki/Raspberry_Pi

Información sobre Code Warrior
“libroweb.alfaomega.com.mx/catalogo/microcontroladoresfreescale-1/libreacceso/libreacceso/reflector/ovas_statics/motorola/CodeWarrior.pdf”

Información sobre integrado L293D

“robotos-argentina.com.ar/MotorCC_L293D.htm”

Información sobre comunicaciones I^2C

http://robots-argentina.com.ar/Comunicacion_busI2C.htm

Información sobre cámaras web

http://es.wikipedia.org/wiki/C%C3%A1mara_web

Información sobre Mjpg-streamer

http://wolfpaulus.com/jounal/embedded/raspberrypi_webcam/

Información sobre MC9S08QG8 datasheet encontrado en:

www.freescale.com

Información sobre integrado L293D datasheet encontrado en:

http://www.datasheetcatalog.com/

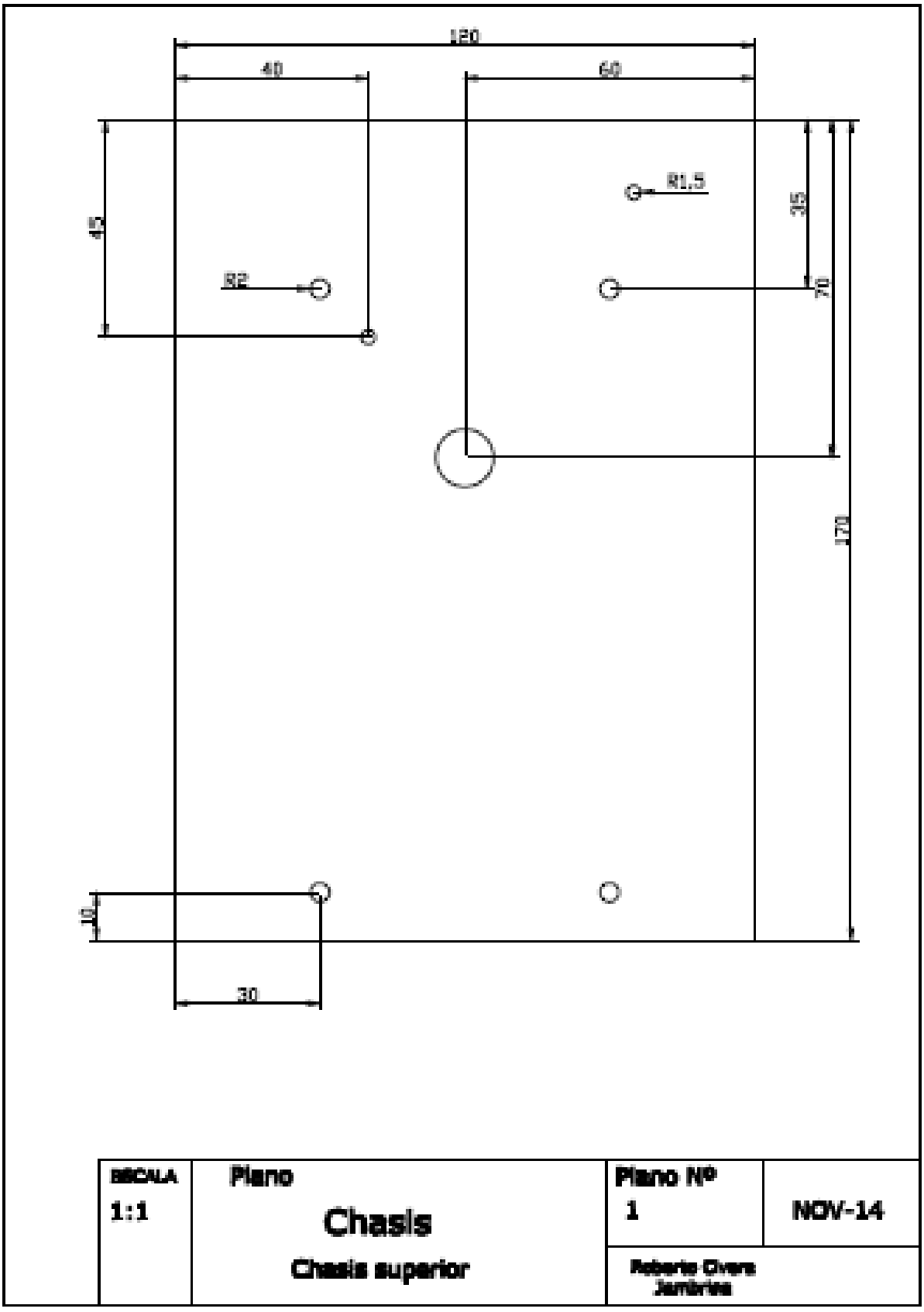
Información sobre MC7805tc datasheet encontrado en:

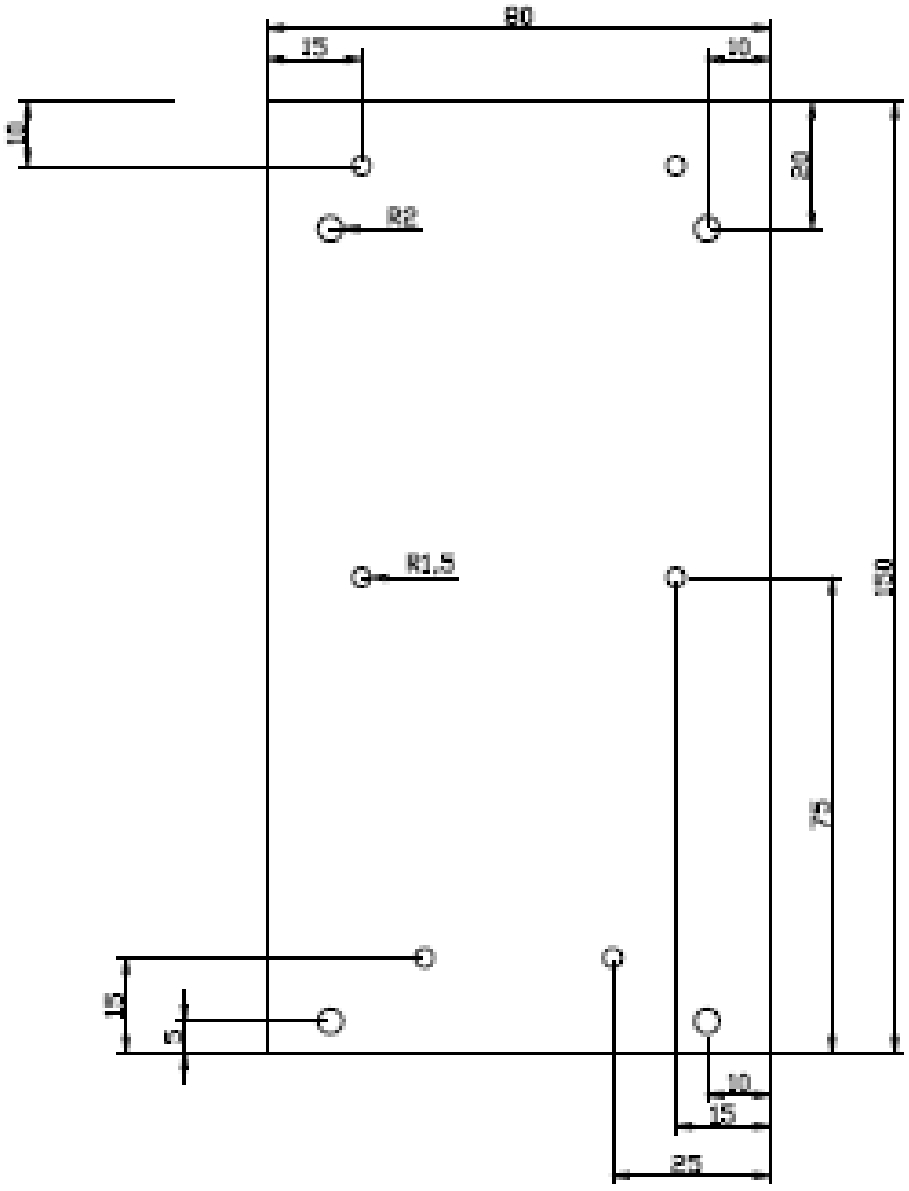
http://www.datasheetcatalog.com/

Información sobre la instalación del watchdog para la Raspberry

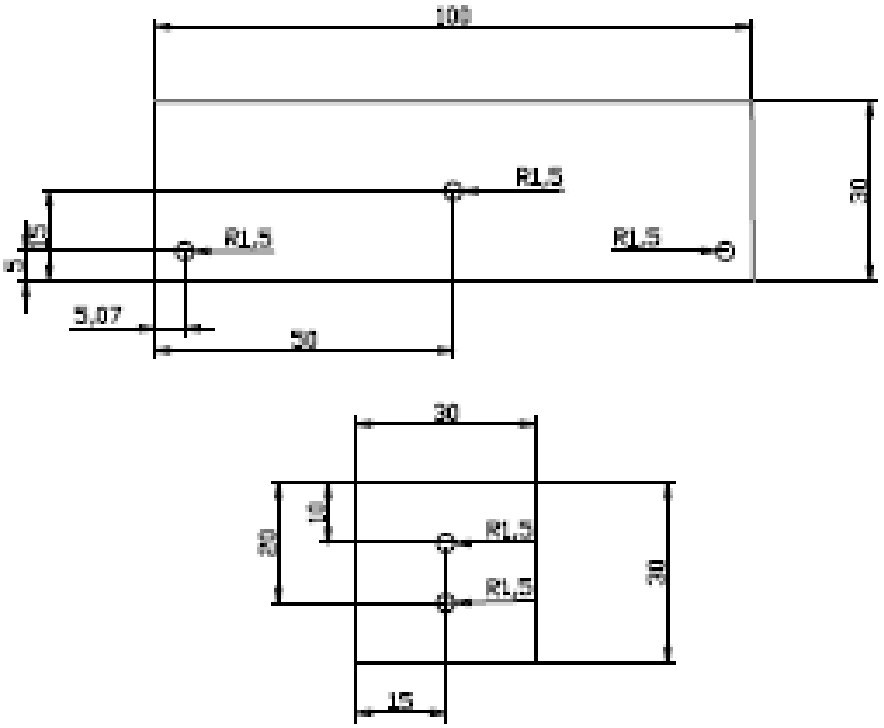
http://raspberrypi.stackexchange.com/questions/1401/how-do-i-hard-reset-a-raspberry-pi

Anexo 1: Planos mecánicos de la maqueta





ESCALA	Plano	Plano Nº	
1:1	**Chasis**	2	NOV-14
	Chasis superior	Roberto Civera Jambrina	



ESCALA	Plano	Plano Nº	
1:1	Chasis Soporte inferior Soporte ruedas delanteras	3	NOV-14
		Roberto Olvera Jambrina	

Anexo 2: Programas aplicación:

```
*********************************************************/
/* Programa de control de VGR                            */
/* Realizado por: Roberto Civera                         */
/* Fecha: 12-9-2014                                      */
/* Funcionamiento: Puesta en marcha de un vehiculo       */
/* controlado por Raspberry Pi y que soporta una camara  */
/* que emite via striming.                               */
/*********************************************************/


#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <linux/i2c-dev.h>
#include <ncurses.h>
#include <time.h>
#include <signal.h>
#include <wiringPi.h>


#define ACABAR ' '


//Define las posiciones de los dispositivos
#define micro_I2C_ADDR 0x38


//Declaracion de variables
       int fd;
```

```
        unsigned int variable;
        unsigned char buf[1];
        unsigned int led=0, error=0;
        unsigned char tecla, direccion;

//Inicializar
void inicializar(){
initscr();
keypad(stdscr,1);
noecho();
curs_set(0);
start_color();
init_pair(1,COLOR_RED,COLOR_BLACK);
timeout(0);
}

//Funcion texto presentacion
void textoconduccion(){
      move(5,7);
      printw("/+++ Menu conduccion +++/");

      move(7,7);
      printw("Pulsa w padra ir hacia delante,s para parar,");
      move(8,7);
      printw("a para ir a la izquierda y d para ir a la derecha");
      move(9,7);
      printw("Para resetear el micro pulsa r ");
      move (20,7);
      printw("Para ir al menu principal pulsa espacio");
}
```

```
//Calcular direccion del vehiculo
void calculardireccion(unsigned char tecla){
switch(tecla){
       case 'w':
        led=1;
        buf[0]=led;
        break;

      case 's':
       led=0;
       buf[0]=led;;
       break;

      case 'a':
       led=3;
       buf[0]=led;
       break;

      case 'd':
       led=2;
       buf[0]=led;
       break;
        }
}

//Operacion de escritura de la raspi en el micro
void operacionescritura(){
 if(write(fd, buf, 1)!=1){
                 error=1;
                 attron(COLOR_PAIR(1));
```

```
                    move(16,15);
                    printw(  "Unable to write from raspi");
                    attroff(COLOR_PAIR(1));
                    //fprintf(stderr, "Unable to write from raspi");
              }
              else{
              //printf("Ahora esta escribiendo \n"); //Para saber cuando
esta escibiendo


              }
}
void operaciondelectura(){
 if (read(fd, buf, 1) != 1)
           {
            error=1;
            move(18,15);
             attron(COLOR_PAIR(1));
                printw("Unable to read from microC");
             attroff(COLOR_PAIR(1));
             //fprintf(stderr, "Unable to read from microC");
           }
           else
           {
                          //printf("Ahora esta leyendo \n");   //Para
saber cuando esta leyendo
                    move(10,7);
                     printw("Val de direccion  %u \n",buf[0]);// Valor
que ha puesto el micro en el PWM
                    if (buf[0]==112){
                          error=1;
                    }
```

```
          }
}
//Abrir puerto i2c
abrirpuertoi2c(){

       if ((fd = open("/dev/i2c-1", O_RDWR)) < 0)
       {
          // Open port for reading and writing

          fprintf(stderr, "Failed to open i2c  autobus\n");

          exit(1);
       }
}

//Inicializacion de dispositivos

    void selectDevice(int fd, int addr, char * name)
    {
       if (ioctl(fd, I2C_SLAVE, addr) < 0)
       {
          fprintf(stderr, "%s not present\n", name);
      exit(1);
       }
    }

//Texto error
void textoerror(){
attron(COLOR_PAIR(1));
move(16,10);
```

```
printw("Ha aparecido un error debe reiniciar el microcontrolador o la
aplicacion");

move(17,10);

printw("Pulse espacio para salir");

attroff(COLOR_PAIR(1));

}


//Funcion texto funcion principal

void textoprincipal(){

      move(1,7);

      printw("/+++ Menu Principal +++/");

      move (2,7);

      printw("Programa de Raspberry PI para la puesta en");

      move (3,7);

        printw("marcha de un vehiculo mediante un micro conectado por
IIC ");


      move (7,7);

      printw("Pulsa 1: Para activar la camara");

      move (9,7);

        printw("Pulsa 2: Para activar la conduccion del vehiculo");

      move (11,7);

        printw("Pulsa espacio: Para terminar");


}

//Funcion de la funcion principal

void abrirfuncionprincipal(unsigned char teclafuncionprincipal){

            switch(teclafuncionprincipal){

                  case '1':

                  abrirfuncioncamara();

            break;
```

```
            case '2':
                  abrirfuncionconduccion();
            break;
            }
}
//Abrir camara
void abrirfuncioncamara(){
unsigned char teclacamara;
      clear();
      move(1,10);
      printw("/+++ Menu camara +++/");
      move (3,10);
      printw("Para ver la imagen poner en el explorador de Internet");
      move (5,10);
      printw("http://direccion ip de la raspi:8001/?action=stream");
      move(8,10);
      printw("Pulsa 1 para encender la camara");
      move(9,10);
      printw("Pulsa 2 para apagar la camara");
      move(10,10);
      printw("Pulsa espacio para salir \n");
      move(11,10);

      while ((teclacamara = getch()) != ACABAR) {
            switch(teclacamara){

            case '1':
                  system("sudo ./mjpgscript");
            break;
```

```
            case '2':
                  system("sudo ./mjpgscript2");
            break;
            }
      }
      clear();
}

void abrirfuncionconduccion(){
unsigned char teclaconduccion;
int pin = 4;
clear();
//Texto de presintacion
      textoconduccion();
      refresh();
      while ((teclaconduccion = getch()) != ACABAR) {
            //Control del PWM
            calculardireccion(teclaconduccion);//Control del PWM
            usleep(90000);
            //Operacion de escritura de la Raspi en el micro
            operacionescritura();
                  usleep(90000);
            //Operacion de lectura de la Raspi al micro
            operaciondelectura();
                  usleep(90000);


            pinMode(pin,OUTPUT);
            if(teclaconduccion == 'r'){
                  if (wiringPiSetupGpio() == -1){
```

```
                move(10,10);
                printw("Error de inicializacion");
                }
                move(10,10);
                printw("Reset");

                digitalWrite(pin,1);
                delay(250);
                digitalWrite(pin,0);

                buf[0]=0;
                operacionescritura();
                error=0;
                clear();
                textoconduccion();
                }

            if (error ==1){
            textoerror();

            }

      }
buf[0]=0;
operacionescritura();
clear();

}
```

```
//Programa principal
    int main(int argc, char **argv)
    {
//Abrir el puerto IIC
abrirpuertoi2c();
//Inicializar el micro
      selectDevice(fd, micro_I2C_ADDR, "microC");

//Seleccion del micro
          //selectDevice(fd, micro_I2C_ADDR, "microC");

//inicializar
      inicializar();
//Bucle del programa princial
            while ((tecla = getch()) != ACABAR){
            textoprincipal();
            refresh();
            abrirfuncionprincipal(tecla);
      }
    system("sudo ./mjpgscript2");
    buf[0]=0;
    operacionescritura();
    usleep(90000);

    endwin();
    return 0;
 }

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

```
/* Programa de control de PWM                                    */
/* Realizado por: Roberto Civera                                 */
/* Fecha: 20 de junio de  2014                                   */
/* Funcionamiento: Programación del microcontrolador para el */
/* control de dos motores mediante la modulación por anchura */
/* de pulso(PWM), al que el valor del PWM se pasa por IIC    */
/* desde una Raspberry                                           */
/*****************************************************************/



#include <hidef.h>
#include <MC9S08QG8.h>

//Definición del constantes
#define SLV_ADDRESS 0x70 //-> 0111 000 ->  011 1000 -> 0x38

//Defenicion de variables

char count = 0;
char rec_count = 0;
char num_to_rec = 0;
char IIC_Rec_Data[1];
char IIC_TX_Data[1];
unsigned char Val_1[1];
unsigned char Val_0[1];
unsigned char direccion[1];
```

```
//Programción de funciones
void CheckSRW(void){

  //Check for Slave Rec or transmit
  if (IICS_SRW){
     //Slave Transmit
     IICC_TX = 1;
     IICD = IIC_TX_Data[count];     //Data which should be sent
      count++;

   } else {    //Slave Receive
            IICC_TX = 0;
      IIC_Rec_Data[rec_count]=IICD;
      rec_count++;
   }

 }

void SlaveReadStore(void){

      IIC_Rec_Data[rec_count] = IICD;
      rec_count++;

}

void startup(void) {  //Función de inicializción

 SOPT1_COPE=0; //Inhabilita el COP
 SOPT2_IICPS = 0; /*0 SDA on PTA2, SCL on PTA3.    1 SDA on     PTB6,
SCL on PTB7.*/
 ICSC2 = 0;             //CLK and Oscillator config
```

```
 if (ICSC2_EREFS)  {
     while (!ICSSC_OSCINIT){
     }
 }
 ICSC1 = 0x06;    //ICSIRCLK active
 while (ICSC1_CLKS != ICSSC_CLKST){
 }/* wait for clock state to be selected */

PTAPE_PTAPE4 = 0;      //Pull-up resistor
PTAPE_PTAPE5 = 0;

//Init IIC Operation
//This is the address that will be responded to if set up as a slave
IICA = SLV_ADDRESS;
//Multiply factor of 1
IICF_MULT = 1;

//SCL divider of 32
IICF_ICR = 0x00;

//Enable IIC and interrupts
IICC  = 0xc0;

//Slave-Mode
IICC_MST = 0;
num_to_rec = 1;              //expected number of data to recieve

PTBDD=0xFF;    //-- Configurar Puerto B para salida
PTBD=0x00;
```

```
 Init_PWM_0(); // Función de configuracion del PWM canal 0

}

//Programación de las interrupciones necesarias
//Interrupciones de IIC y Timer para PWM

 interrupt 17 void IIC_ISR(void){  //Interrupción del IIC
       if (rec_count == 3){
            rec_count = 0;
            }

      //Clear Interrupt Flag
      IICS_IICIF = 1;

      //Master or Slave?
      if (IICC_MST)
      {//MASTER
      } else {
            //Slave Operation
            if (IICS_ARBL){
                  IICS_ARBL = 1;
                   //Check For Address Match
                   if (IICS_IAAS) {
                                    //rec_count=0;
                                    //count = 0;
                                    //CheckSRW();
                    }
```

```
            }      else{
            //Arbitration not Lost
                        if (IICS_IAAS) {

                              count = 0;

                              CheckSRW();

                        }else {


                        if (IICC_TX) {
                                    //Check for rec ACK


                                    if (!IICS_RXAK) {
                                          //ACK Recieved


                                          IICD = IIC_TX_Data[count];
                                                count++;

                                    } else{
                                                IICC_TX = 0;//IICD;
                                    IIC_Rec_Data[rec_count]=IICD;

                                    rec_count++;

                                    }


                                     } else {

                                           SlaveReadStore();

                                     }

                         }

                        }

            }
}

interrupt 7 void overflow_timer(void){ //Interrupción del timer

  TPMSC_TOF=0;          //Borra el flag

}

interrupt 6 void PWM_canal1(void){//Interrrupción del canal 1 del
timer
```

```
      TPMC1SC_CH1F=0;   //Borra el flag
}
interrupt 5 void PWM_canal0(void){
      TPMC0SC_CH0F=0;   //Borra el flag
}
 void ir_adelante(unsigned char V0){
   Set_Value_0(V0); //Funcion que ejecuta el PWM canal 0
   Set_Value_1(V0); //Funcion que ejecuta el PWM canal 1
  }
 void ir_derecha(unsigned char V0){
   Set_Value_1(V0); //Funcion que ejecuta el PWM canal 1
 }
void ir_izquierda(unsigned char V0){
  Set_Value_0(V0); //Funcion que ejecuta el PWM canal 0
 }
void parar(unsigned char V0){
   Set_Value_0(V0); //Funcion que ejecuta el PWM canal 0
   Set_Value_1(V0); //Funcion que ejecuta el PWM canal 1
}
//Program principal
void main (void){
  DisableInterrupts;
  startup();
  EnableInterrupts;
  direccion[0]=0;
  IIC_TX_Data[0]=IIC_Rec_Data[0];  // Copia lo que recive para
transmitir

 for(;;) {
 //Val_0[0]= IIC_TX_Data[0]; //Asignacion del valor recivido al PWM
```

```
 direccion[0]= IIC_TX_Data[0];
   Val_0[0]=254;
   Val_1[0]=0;
   switch(direccion[0]){
    case 0:

      parar(Val_1[0]);
      break;

    case 1:
      ir_adelante(Val_0[0]);
      break;

    case 2:
      parar(Val_1[0]);
      ir_derecha(Val_0[0]);
      break;
    case 3:
        parar(Val_1[0]);
      ir_izquierda(Val_0[0]);
      break;



   }



  }
}
/*************************************************************/
/* Programa de configuración del PWM                         */
```

```
/* Realizado por: Roberto Civera                                  */
/* Fecha: 20 de junio de  2014                                    */
/* Funcionamiento: Programación del microcontrolador para el */
/* control de dos motores mediante la modulación por anchura */
/* de pulso(PWM), al que el valor del PWM se pasa por IIC    */
/* desde una Raspberry                                            */
/****************************************************************/


#include <hidef.h>
#include <MC9S08QG8.h>
#include "pwm0.h"


void Init_PWM_0(void){
//Valores del iniciales del contador y del valor del PWM
  TPMMOD=254;
  TPMC0V=00;
//Configuración del registro de control del canal 0
  TPMC0SC_CH0F=0;
  TPMC0SC_CH0IE=1;
  TPMC0SC_MS0B=1;
  TPMC0SC_MS0A=0;
  TPMC0SC_ELS0B=1;
  TPMC0SC_ELS0A=0;

  TPMC1V=00;
//Configuración del registro de control del canal 1
  TPMC1SC_CH1F=0;
  TPMC1SC_CH1IE=1;
```

```
  TPMC1SC_MS1B=1;
  TPMC1SC_MS1A=0;
  TPMC1SC_ELS1B=1;
  TPMC1SC_ELS1A=0;

 //Configuracion del registro de control del timer
  TPMSC_TOF=0;
  TPMSC_TOIE=1;
  TPMSC_CPWMS=0;
  TPMSC_CLKSB=0;
  TPMSC_CLKSA=1;
  TPMSC_PS0=1;
  TPMSC_PS1=1;
  TPMSC_PS2=1;

}



void Set_Value_0(unsigned char V){   //Función donde se le da valor al
registro del PWM 0
 TPMC0V= V;
}

void Set_Value_1(unsigned char V){   //Función donde se le da valor al
registro del PWM 1
 TPMC1V= V;
}
```