UNIVERSIDAD NACIONAL AUTONOMA DE MÉXICO

FACULTAD DE INGENIERÍA

“ROBOT LIMPIADOR DE PLAYA”

TESIS

QUE PARA OBTENER EL TITULO DE:
INGENIERO ELÉCTRICO Y ELÉCTRONICO E
INGENIERO EN COMPUTACIÓN
PRESENTAN:

CORTES PICHARDO DANIEL
GAMERO GONZÁLEZ MARÍA DE LOS ANGELES

ASESOR:
DR. JESÚS SAVAGE CARMONA

CIUDAD UNIVERSITARIA 2009.

Agradecimientos.

Agradezco a Dios por concederme su luz y amor los cuales avivan la llama de mi fe.

Dedico esta tesis a mi madre Bertha Pichardo quien es todo un ejemplo a seguir, quien durante toda su vida me ha dado su amor y apoyo incondicional en todo momento de mi vida y gracias a su grande esfuerzo puede culminar esta parte importante en mi vida profesional.

A Nancy Lili y a Víctor Hugo mis hermanos, que me han dado todo su apoyo y cariño que siempre estuvieron conmigo en todos los momentos importantes durante la carrera.

A mis abuelitos por sus consejos y cariño que siempre me han brindado.

A mi compañera de carrera y novia María de los Ángeles, por su cariño y apoyo tan grande en la realización de esta tesis.

A mi gran amigo Asterix quien siempre me dio un gran apoyo en los momentos difíciles de la carrera y que gracias a el y su cariño aportaron en mi formación profesional y hoy lo recuerdo con mucho amor.

A mis primos y tíos por darme su apoyo y amistad.

A mi asesor Dr. Jesús Savage Carmona por haber creído en nosotros al darnos su confianza, su apoyo y su amistad durante la realización de este trabajo, que gracias a él he podido crecer en lo profesional y le estoy muy agradecido.

También quiero agradecer al Departamento de Posgrado de la Facultad de Ingeniería en especial al Laboratorio de Biorobótica por darnos las facilidades de la realización de este proyecto.

A mis profesores de la Facultad de Ingeniería que gracias a sus consejos y ejemplo he podido crecer profesionalmente y tener siempre el espíritu de superación y excelencia que todo profesional debe cultivar.

A la UNAM por haberme recibido con los brazos abiertos y haberme dado todas las facilidades para crecer como persona y en lo profesional.

Finalmente a todos mis amigos y amigas que me permitieron contar con el privilegio de su amistad y compartir momentos de felicidad.

.

Agradecimientos

Agradezco a Dios por todo lo que me ha brindado en estas etapas de mi vida; por su amor y caridad que son fuente de la fe que le tengo.

A mi madre Josefina González, por que gracias a ella y a su apoyo he podido culminar con esta etapa de mi vida. Gracias por tu cariño y comprensión.

A mis hermanos José David y Carlos por su apoyo.

A Francisco por apoyar a mi madre y a nosotros, en esta etapa.

A mis abuelitos por su cariño.

A mis primos y tíos por darme su apoyo.

A mi compañero de la escuela y de la vida Daniel Cortes Pichardo por creer en mí al darme la confianza, el apoyo y la oportunidad de realizar este trabajo con él; le agradezco a su familia por haberme abierto las puertas de su corazón.

A mi asesor Dr. Jesús Savage Carmona por haber depositado su confianza en nosotros en la realización de este trabajo.

También quiero agradecer al Departamento de Posgrado de la Facultad de Ingeniería en especial al Laboratorio de Biorobótica por darnos las facilidades de la realización de este proyecto.

A mis profesores de la Facultad de Ingeniería que gracias a sus consejos y ejemplo he podido crecer profesionalmente y tener siempre el espíritu de superación y excelencia.

A la UNAM por haberme recibido con los brazos abiertos y haberme dado todas las facilidades para crecer como persona y en lo profesional.

Finalmente a todos mis amigos y amigas que me permitieron contar con el privilegio de su amistad y compartir momentos de felicidad, gracias por el apoyo que me otorgaron a lo largo de la carrera.

Indice.

1

1. CAPÍTULO I INTRODUCCION 1
1.1 Objetivos Específicos. 1
1.2 Metas. 1
1.3 Justificación. 1
1.4 Microcontroladores 1
1.5 Sistemas embebidos de visión 3
1.6 Panorama de la Tesis 4

2

2. CAPÍTULO II EL ROBOT 5
2.1 Introducción 5
2.1.1 Descripción del robot 5
2.2 Parte mecánica 9
2.2.1. Mecanismo que da movimiento al robot 10
2.2.2 Brazo mecánico 10
2.3 Parte eléctrica 11
2.3.1 Tarjeta controladora 12
2.3.2 Introducción al Microcontrolador PIC16F877A 12
2.3.3 Comunicación Serial 18
2.4 Actuadores 22
2.4.1 El motor de DC 22
2.4.1.1 Puente H 22
2.4.1.2 Puente H con relevadores 24
2.4.1.3 El L293 como controlador de motores de DC 27
2.4.2 Servomotores 28
2.5 Configuración del IDE MPLAB para Windows 30
2.6 Ejemplos de programación en MPLAB 30
2.6.1 Programa que recibe y envía datos por el puerto serial 30
2.6.2 Programa que imprime el estado de los bumpers conectados en el puerto A 30

3

3. CAPÍTULO III EL SISTEMA EMBEBIDO DE VISION 31
3.1 Introducción 31
3.2 Sistemas embebidos 31
3.2.1 Sistema embebido de visión 31
3.3 Introducción a la CMUcam3 32
3.3.1 Hardware de la CMUcam3 32
3.3.1.1 Alimentación de la CMUcam3 33
3.3.1.2 Puerto Serial 34
3.3.1.3 BUS de la cámara 35
3.3.1.4 Puerto para servomotores 35
3.3.1.5 Botón ISP 36
3.3.2 Software de la CMUcam3 37
3.3.2.1 Instalación del software para Windows 37
3.3.2.1.1 Instalación de Cygwin 37
3.3.2.1.2 Instalación de GNU ARM GCC 40
3.3.2.1.3 Instalación de LPC210x FLASH Utility 41
3.3.2.1.4 El cc3 source tree 41
3.3.2.2 Como crear un proyecto en cc3 42
3.3.2.3 Programación de la CMUcam3 44
3.4 Programas de prueba 45
3.4.1 Programa para recibir y enviar datos mediante el puerto serial de la CMUcam3 45
3.4.2 Programa que detecta el color rojo 45
3.4.3 Programa CMUcam2 y la utilidad CMUcam3 Frame Grabber 46

4

4 CAPÍTULO IV DESARROLLO 48
4.1 Introducción 48
4.2 Estructura mecánica 48
4.3 Sistema eléctrico 49
4.3.1Tarjeta controladora con PIC16F877A 49
4.3.2 Programación de la tarjeta controladora 53
4.3.3 Tarjetas de potencia 54
4.3.4 Montaje y conexión de la tarjeta controladora al sistema 56
4.3.5 Control de los actuadores 59
4.4 El sistema embebido de visión CMUcam3 61
4.4.1 Programación de la CMUcam3 62
4.4.2 Detección de los colores rojo, verde y azul 62
4.4.3 Calibración de los colores mediante la utilidad CMUcam Frame Grabber 63
4.4.4 Montaje y conexión de la CMUcam3 al sistema 64
4.5 Algoritmos de programación del robot limpiador de playa 65

5

5. CAPÍTULO V PRUEBAS Y RESULTADOS 80
5.1 Introducción 80
5.2 Configuración de las pruebas 80
5.2.1 Configuración de las pruebas para la tarjeta controladora. 80
5.2.1.1 Comunicación con la tarjeta controladora 81
5.2.1.2 Control de movimiento del robot 81
5.2.1.3 Control de movimiento del brazo mecánico 82
5.2.2 Configuración y pruebas con la CMUcam3 85
5.2.2.1 Comunicación con la CMUcam3 85
5.2.2.2 Seguimiento de colores 85
5.2.3 Pruebas entre la comunicación de la CMUcam3 y la tarjeta controladora 86
5.2.3.1 Envío de caracteres a la tarjeta controladora desde la CMUcam3 86
5.2.3.2 Sincronización de la comunicación entre la CMUcam3 y la tarjeta controladora 90
5.2.4 Pruebas con el sistema completo 90
5.2.4.1 Búsqueda de residuos 90
5.2.4.2 Evasión de obstáculos 91
5.2.4.3 Recolección de residuos 91
5.2.4.4 Depósito de residuos 91
5.3 Conclusiones 91

6

6. CAPÍTULO VI CONCLUSIONES Y TRABAJO A FUTURO 93
6.1 Conclusiones 93
6.2 Trabajo a futuro y recomendaciones 94

A

APÉNDICE A 95
APÉNDICE B 111

B

Bibliografía y Referencias 119

CAPÍTULO I INTRODUCCION

El objetivo general de este proyecto es el desarrollo y la implementación de un sistema autónomo capaz de recolectar basura en la playa. El proyecto está basado en una araña comercial a la que se le realizarán ciertas modificaciones para ser controlada mediante una tarjeta controladora y un sistema embebido de visión.

1.1 Objetivos Específicos.

- Modificar la araña comercial para poder controlarla por medio de una tarjeta controladora.
- Implementar una tarjeta controladora por medio del microcontrolador PIC16F877A.
- Desarrollar las modificaciones pertinentes a la tarjeta de potencia que da movimiento a la araña para que pueda ser operada por la tarjeta controladora.
- Utilizar un sistema embebido de visión CMUcam3 para detectar los tres diferentes colores de basura y que se comunique a través del puerto serial en la tarjeta controladora.
- Controlar el puente H con relevadores por medio de la tarjeta controladora
- Controlar el arreglo de servo-motores y el motor de DC por medio de la tarjeta controladora con el fin de controlar el brazo mecánico.
- Controlar el sistema por medio de una computadora tipo laptop.
- Verificar el algoritmo de búsqueda, el de sincronización con el brazo mecánico y el de comunicación con el sistema embebido de visión CMUcam3.
- Realizar las pruebas de funcionamiento del sistema.

1.2 Metas.

La meta de este proyecto está definida por los objetivos específicos en los que se basa la realización de este trabajo. El proyecto está enfocado en el desarrollo y la implementación de un sistema autónomo recolector de basura en la playa, que por medio de una tarjeta controladora y un sistema embebido de visión, enviarán las señales que se requieran para manipular el brazo mecánico y el movimiento del robot. El alcance de este proyecto es campo fértil para nuevas investigaciones sobre el área de navegación para los robots móviles.

1.3 Justificación.

La necesidad de mantener limpias las playas a cualquier hora y buscando una forma económica para hacerlo, abre un campo de investigación para la robótica, en el cual se requieren robots limpiadores de playa, los cuales requieren de sistemas autónomos capaces de navegar por un ambiente desconocido, limpiando la basura que encuentren en su camino y evadiendo los obstáculos que impidan su objetivo.

1.4 Microcontroladores

El microcontrolador es el encargado de manipular los actuadores del robot para garantizar la tarea de recolección de basura. El robot cuenta con un microcontrolador como cerebro principal, el cual es un circuito integrado programable, que cuenta con un

microprocesador o unidad de procesamiento central (CPU), una memoria para almacenar el programa, una memoria para almacenar datos y puertos de entrada y salida.

Los microcontroladores PIC son una familia de microcontroladores tipo RISC (Reduced Instruction Set Computer), fabricados por Microchip Technology Inc. [1], los cuales últimamente han tenido un gran auge en la robótica y en el mercado electrónico, debido a su gran variedad y suficiente información para desarrollar sistemas digitales con ellos.

En la actualidad existen circuitos electrónicos comerciales en los que son utilizados los microcontroladores, debido a que permiten reducir el tamaño y el precio de los equipos.

Las áreas en las que son utilizados los microcontroladores son:

- Aparatos electrodomésticos
- Control industrial
- Robótica
- Periféricos de computadoras
- Sistemas de seguridad
- Telefonía celular
- Navegación

En la robótica, para que un robot tenga una idea de su ambiente y pueda reaccionar ante él, de acuerdo a ciertos aspectos ambientales como lo seria el encontrarse con un obstáculo; se necesita de un conjunto de sensores, con los cuales podrá ser capaz de reaccionar de acuerdo a la tarea específica para la cual fue construido o para reaccionar de acuerdo al ambiente en el que esté, y de un microcontrolador para regir el comportamiento del robot. El microcontrolador por medio de sus periféricos se encargará de recibir las señales digitales y en ocasiones señales analógicas provenientes de los sensores, y de acuerdo a estas señales el microcontrolador responderá enviando las señales necesarias a los actuadores que regirán el movimiento del robot.

En el presente trabajo se desarrolló una tarjeta controladora, la cual va montada en el robot y se encarga de controlar todo el sistema. Para desarrollar la tarjeta controladora se utilizó el microcontrolador PIC16F877A fabricado por Microchip Technology. Las principales razones por las cuales se eligió a este microcontrolador fueron tres:

- La facilidad de ser programado mediante una interfaz serial.
- El número de puertos suficiente para implementar el control.
- El empleo de un compilador C para su programación.

El compilador de C para PIC’s que se empleó es el CCS [4], el cual le da alta portabilidad y legibilidad a nuestros programas en comparación con el lenguaje de programación de ensamblador.

En la tarjeta controladora se conectaron todos los sensores y actuadores que el robot requirió, en algunos casos se necesitó de una interfaz para poder conectarlos a la tarjeta, como lo fue en la parte de potencia, donde se requirió de una interfaz capaz de controlar a los dos motores de DC que tiene el robot, para esta función se utilizaron dos tarjetas:

la primera controla los motores de DC que hacen que la araña se mueva, en base a dos puentes H construidos con 2 relevadores cada uno y activados mediante el circuito integrado ULN2003, mientras que la segunda tarjeta es también un controlador para motores de DC pero con menor potencia, construido con el circuito integrado L293D, el cual se encarga de controlar las pinzas del robot.

Dentro de los sensores que se conectaron a la tarjeta se encuentran: un arreglo de cuatro bumpers, el cual le permite al robot evadir obstáculos y una cámara para la visión, que le permite al robot identificar figuras y colores. Este último sensor requiere la interfaz serial del microcontrolador, por la cual se envían valores hexadecimales codificados a la tarjeta controladora en caso de que la cámara vea alguno de los colores que sean de nuestro interés.

1.5 Sistemas embebidos de visión

Para poder dotar al robot de un sistema con visión, es necesario utilizar una cámara digital, la cual es considerada como el sensor más complejo utilizado en la robótica.

Anteriormente las cámaras digitales no eran utilizadas en sistemas embebidos, debido a que requieren de una alta velocidad de procesamiento y memoria, por ello se desarrollaron sistemas embebidos de visión compactos, entre los cuales se encuentran el eyebot [3] y la CMUcam3 [2], esto con el fin de proporcionar a nuestros sistemas una visión de su entorno. El presente trabajo fue desarrollado empleando la CMUcam3, el cual es un sistema embebido de visión para robots, desarrollado por la Universidad de Carnegie Mellon y compuesto por una cámara en formato digital y un sistema de desarrollo de código abierto. La CMUcam3 es un sistema embebido de visión basado en el ARM7TDMI [6], que tiene como procesador principal el NXP LPC2106 [7], de 32 bits, el cual va conectado al módulo del sensor de la cámara CMOS, creado por omnivision [8]. La CMUcam3 es programada con librerías de código abierto de GNU TOOLCHAIN [9], que trae herramientas para escribir código en lenguaje C para ella, así como un IDE que nos generará el código ejecutable y la ventaja de programarlo por el puerto serie sin necesidad de hardware adicional. Las características principales de la CMUcam3 son:

- Sensor RGB en color con una resolución de 352 x 288.
- Ambientes de desarrollo para Windows y Linux.
- Conector para tarjeta SD o MMS con soporte FAT 16
- Cuatro Puertos para controlar servos
- Cargar imágenes en memoria a 26 tramas por segundo
- Compresión por software en modo JPEG
- Emulación de la CMUCAM2

El robot utiliza el sensor RGB de la CMUcam3 y los colores que deberá ver son el rojo, el verde y el azul, por lo que la CMUcam3 quedará bien para nuestros fines.

Para conectar a la tarjeta controladora, se utilizó el puerto serial de la tarjeta y de la CMUcam3, de tal manera que cuando la CMUcam3 detecte alguno de los colores programados en ella, enviará a la tarjeta controladora la posición codificada del centroide del objeto que ve. De esta manera la tarjeta controladora se encargará de

mandar las señales necesarias a los actuadores correspondientes. Debido a las diferentes velocidades en que trabajan los sistemas, es posible que en algún momento dado la CMUcam3 sature el buffer de recepción del microcontrolador, por lo que se necesitará utilizar una técnica de sincronización para evitar que se sature el buffer de información, esta técnica se explicará en capítulo 5.

1.6 Panorama de la Tesis

La tesis esta conformada por seis capítulos

Capítulo I

Se da un panorama general de lo que trata el proyecto y se da una breve introducción de los Microcontroladores PIC y el sistema embebido de visión que se emplearon en este proyecto.

Capítulo II

En este capítulo se describen las partes que componen al robot desde el material empleado para su construcción hasta el software empleado para darle movimiento. Se describe la tarjeta controladora basada en un microcontrolador PIC16F877A [5], haciendo énfasis en sus características empleadas en el proyecto así como el software empleado para la programación de este microcontrolador.

Capítulo III

Aquí se habla de los Sistemas embebidos de visión, los diferentes sensores que se emplean en la robótica y el actuador empleado en este proyecto. Se proporcionan las características principales de la CMUcam3 [2] y sus modos de operación.

Capítulo IV

Se plantea el desarrollo y la implementación del robot limpiador de playa, las pruebas realizadas y se proporciona a detalle las conexiones entre las diferentes unidades utilizadas en este proyecto.

Capítulo V

Se plantean las pruebas realizadas una vez terminado el robot, probando la visión, sensores y actuadores para la tarea de recoger basura.

Capítulo VI

En este capítulo se escriben las conclusiones encontradas durante el desarrollo del sistema, así como recomendaciones para futuras modificaciones al robot y para un trabajo futuro.

CAPÍTULO II EL ROBOT

2.1 Introducción

En este capítulo se describe el funcionamiento del robot, tanto en su parte mecánica como eléctrica, y se mencionan las herramientas empleadas para la programación, como el IDE MPLAB, el cual nos permite desarrollar código fuente en C y compilarlo a fin de generar código ejecutable para descargar en nuestra tarjeta controladora.

2.1.1 Descripción del robot

Nuestro robot es un sistema basado en una araña con 6 patas (Figura 2.1a y figura 2.1b), cuya parte locomotriz la conforman dos sistemas de engranes, cada uno de los cuales es movido por un motor. El sistema de engranes en conjunto con los motores le dan el movimiento a las seis patas de la araña, donde cada sistema de engranes mueve tres patas de la araña correspondientes a un juego de patas izquierdas y un juego de patas derechas. Para poder mover a la araña solo se necesita un puente H por cada motor. Para nuestro robot, se diseñaron dos tarjetas de potencia, que corresponden a los controles de los dos motores mediante dos puentes H construidos con relevadores.

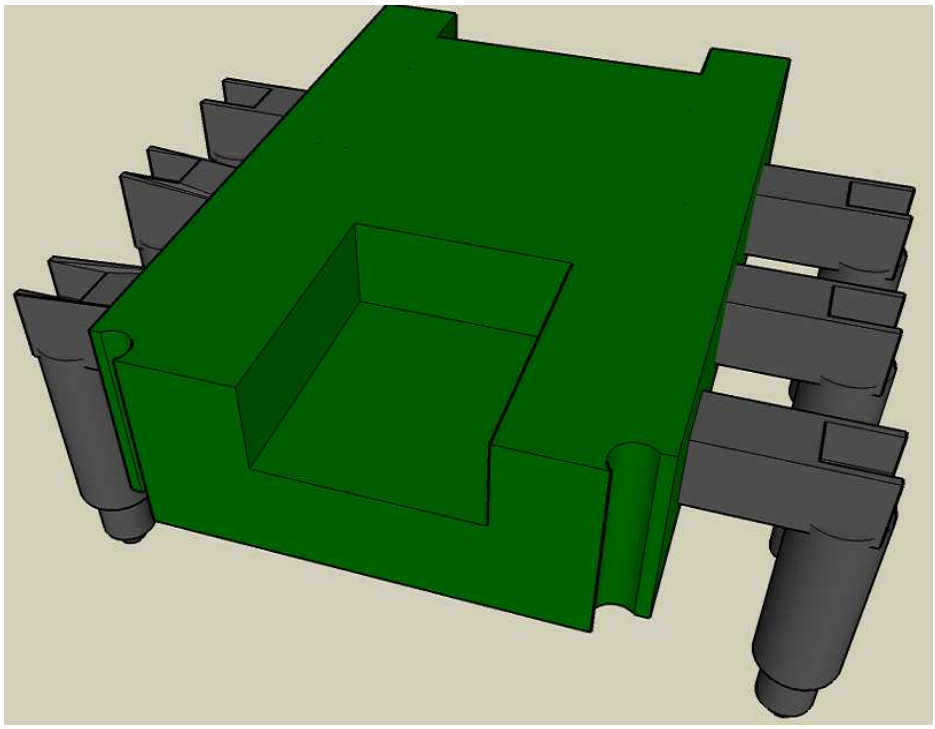


Figura 2.1a Araña con 6 patas

Figura 2.1b Araña comercial [18]

En la figura 2.1c se muestra un esquema de bloques del sistema principal de la araña, donde podemos visualizar los elementos que conforman este proyecto para su correcto funcionamiento y realización de sus principales tareas: evadir obstáculos y búsqueda de basura.

Los bloques de esta figura serán descritos de la siguiente manera:
Por la parte izquierda tenemos:

- Driver del motor 1: es el encargado de mover el juego de patas izquierdas de la araña.

La parte central esta conformada por los siguientes elementos:

- Baterías 1 y 2: son las encargadas de darle la energía a los drivers de los motores y el driver de la pinza.
- Microprocesador: es el cerebro de nuestro robot, contiene el programa que le da el movimiento al robot y genera las órdenes necesarias para los drivers 1 y 2, los servomotores 1 y 2, el estado de los bumpers y el sistema de visión; permitiendo que estos funcionen de la manera correcta.
- Servomotores 1 y 2: son los encargados de darle la posición de los juegos de patas tanto del lado izquierdo (servomotor 1) y del lado derecho (servomotor 2).
- Sistema de visión: este se encarga de darle visión a nuestro robot, es decir, con este sistema el robot es capaz de encontrar el color que anteriormente le fue programado.
- Bumper's 1, 2, 3 y 4: son los encargados de percibir si existe un obstáculo o no y este sensado se envía al microprocesador el cuál mandará las instrucciones adecuadas para evadir el obstáculo.
- Driver para motor de la pinza: su tarea es la de abrir y cerrar la pinza.

Parte derecha:

- Driver motor 2: es el encargado de mover el juego de patas derecha de la araña.

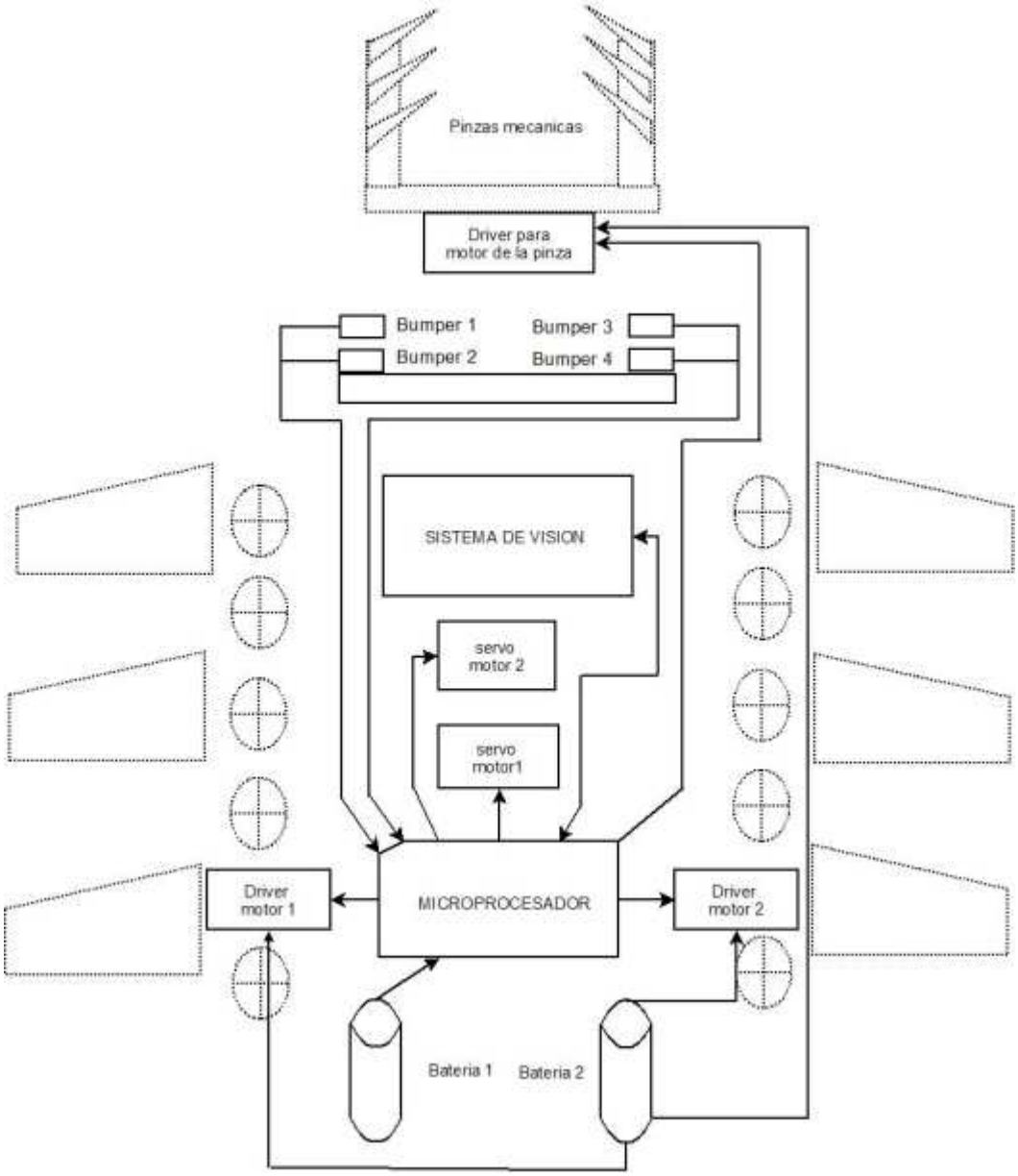


Figura 2.1c Diagrama de bloques de la estructura general de la araña.

El robot para poder recolectar la basura cuenta con un brazo mecánico montado en la parte superior del robot, como se muestra en la figura 2.2.

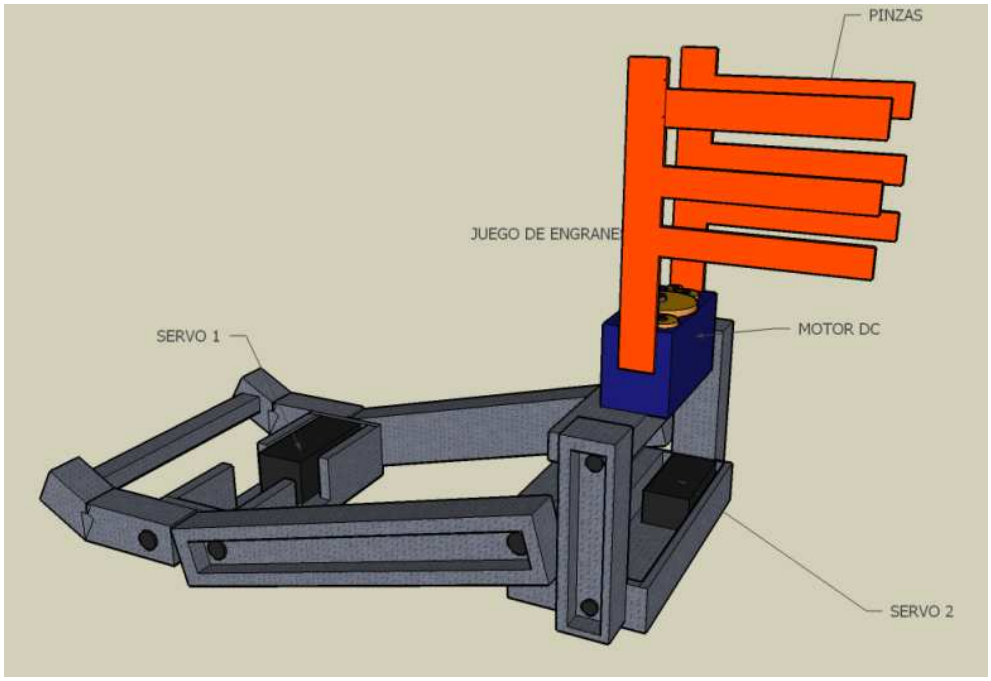


Figura 2.2 Brazo mecánico.

Para el control del brazo mecánico se requirió el empleo de dos servomotores para mover las articulaciones y un motor de DC para mover las pinzas del robot. El motor de DC requirió un puente H para su control, para el cual se empleó el circuito L293D.

El robot también cuenta con un parachoques, el cual es un arreglo de bumpers como muestra la figura 2.3, cuyo fin es el de detectar obstáculos que la CMUcam3 no alcance a ver.

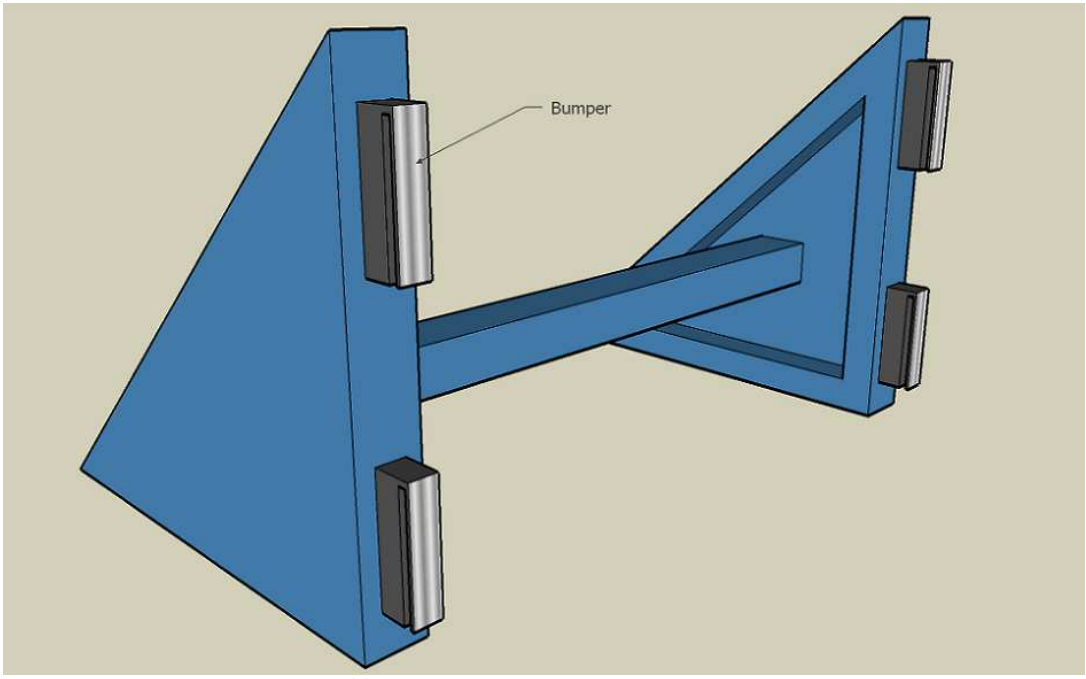


Figura 2.3 Parachoques

Para la visión del robot, se utilizó el sistema embebido de visión CMUcam3, el cual es el sensor que se encarga de identificar los colores y objetos que el robot deberá recolectar.

Fue colocado en la parte frontal de la araña con una protección especial para que no se dañara, como lo podemos ver en la figura 2.4.

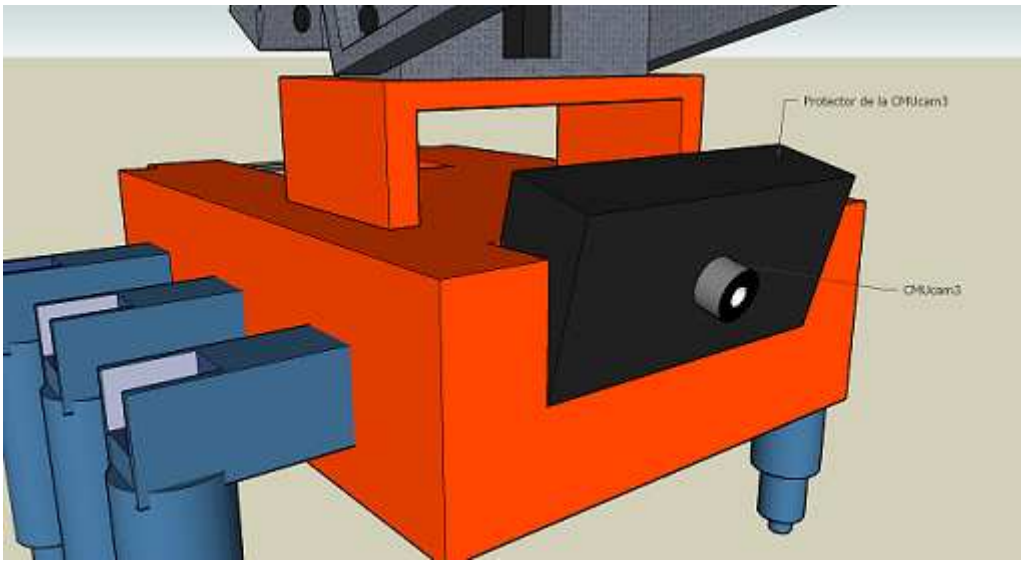


Figura 2.4 Compartimiento de la CMUcam3.

2.2 Parte mecánica

La parte mecánica esta conformada por dos sistemas de engranes, los cuales se encargan de darle movimiento al robot, y un brazo mecánico montado en la parte superior del robot para realizar la tarea de recoger basura. La figura 2.5 muestra la arquitectura mecánica que conforma al robot, cuyo modelo esta basado en una araña mecánica comercial (Figura 2.1b), la cual fue modificada para ser controlada con una tarjeta controladora.

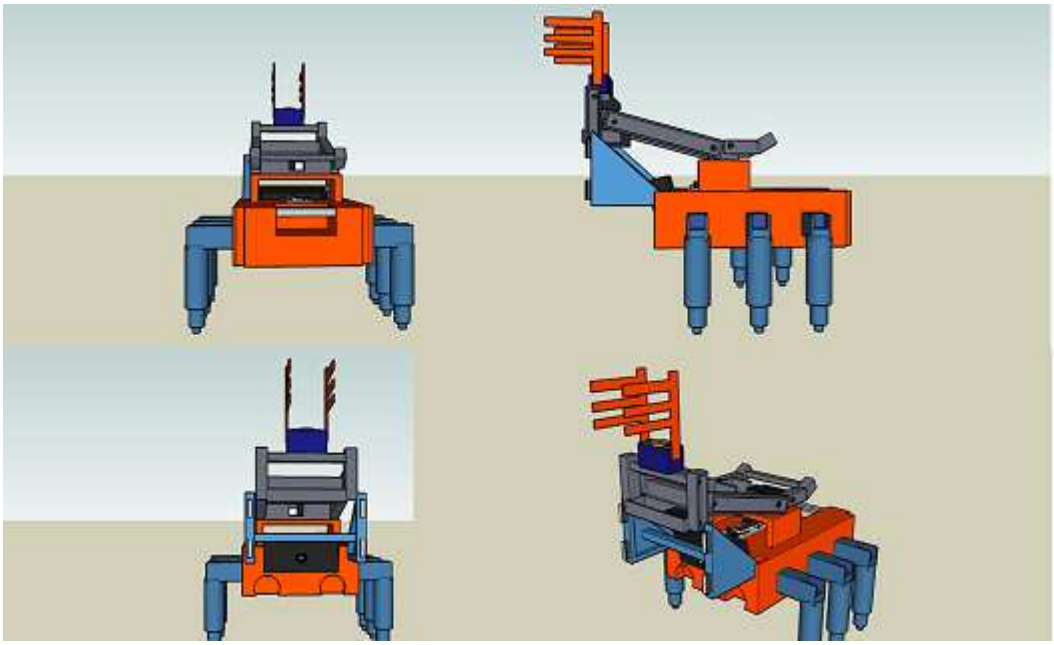


Figura 2.5 Arquitectura general de la araña.

2.2.1. Mecanismo que da movimiento al robot

El mecanismo que da movimiento al robot son dos sistemas de engranes, cada uno movido por un motor; tanto los dos sistemas de engranes, como los motores, venían incluidos en la araña y se utilizaron tal cual sin modificación alguna.

2.2.2 Brazo mecánico

El brazo mecánico es un complemento que se le hizo a la araña con el fin de realizar la tarea de recolección de basura y está hecho de aluminio, por ser un material durable y ligero. El brazo consta de tres movimientos que son mostrados en las figuras 2.6a, 2.6b y 2.6c

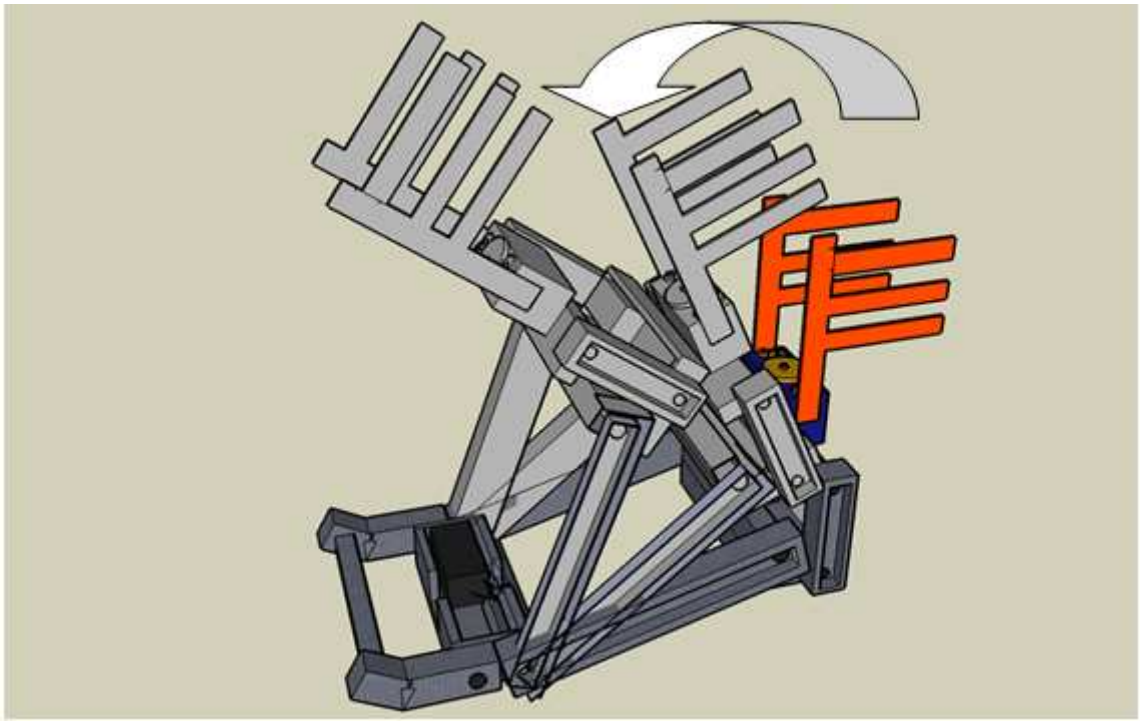


Figura 2.6a Movimiento de la primera articulación.

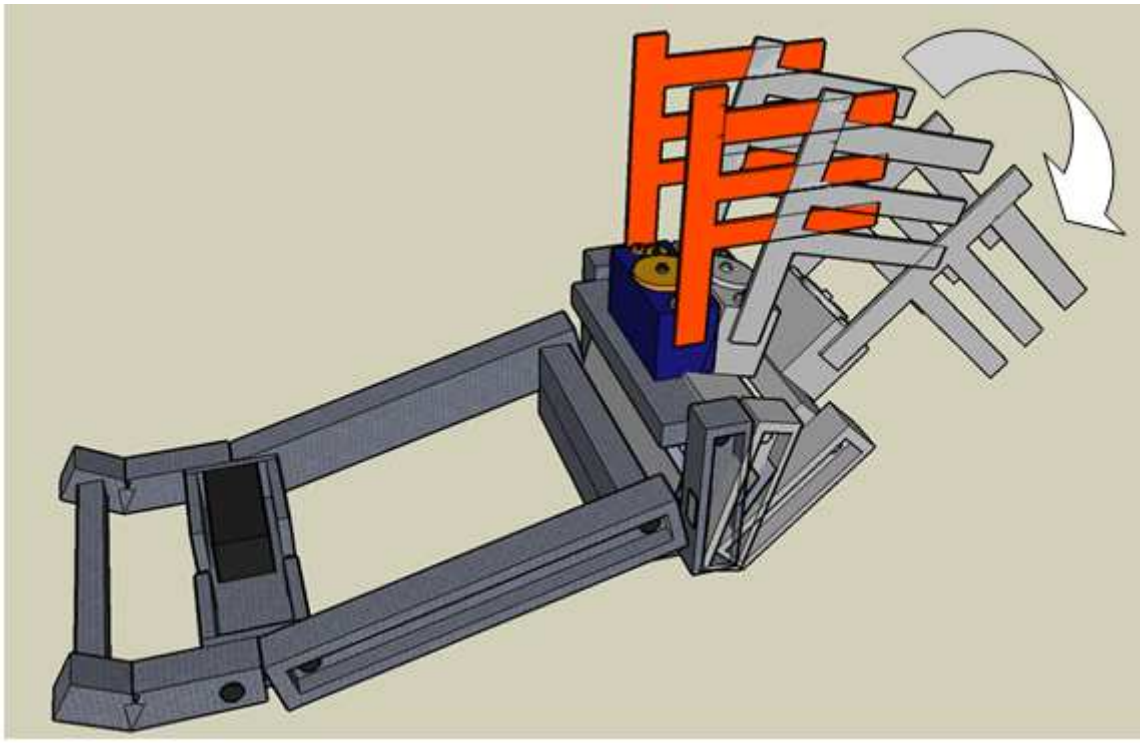


Figura 2.6b Movimiento de la segunda articulación

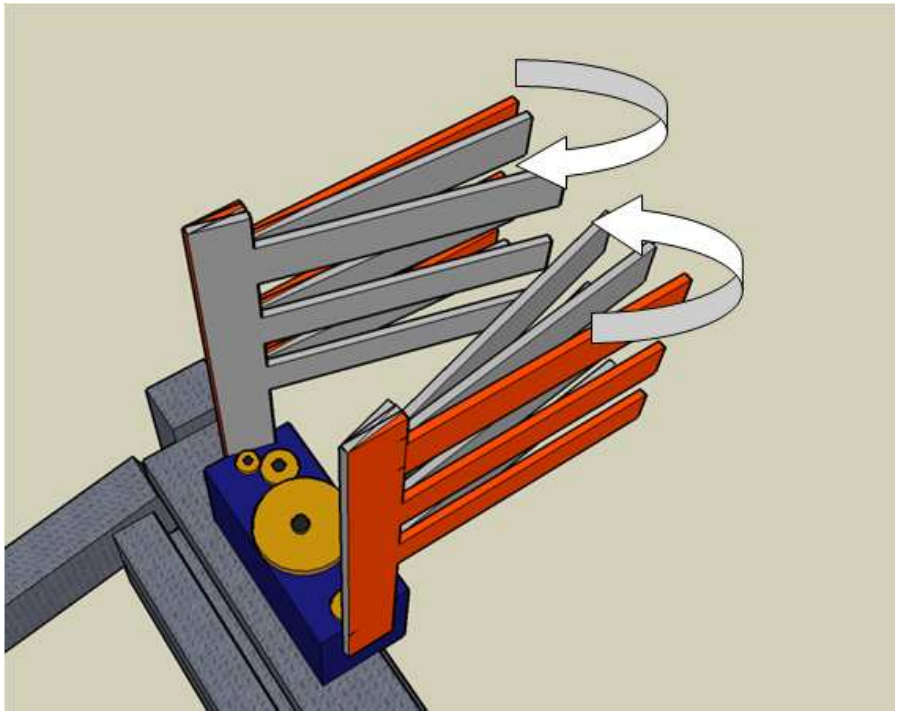


Figura 2.6c Movimiento de las pinzas del brazo mecánico.

2.3 Parte eléctrica

La parte eléctrica del robot la conforman la tarjeta controladora, la tarjeta de potencia y la CMUcam3. Las siguientes secciones describen a detalle cada uno de los elementos eléctricos que conforman al sistema.

2.3.1 Tarjeta controladora

El robot cuenta con una tarjeta controladora basada en un microcontrolador PIC16F877A, que es el cerebro del robot y está encargada de controlar todas las acciones que el robot requiera para completar la tarea de recolección de basura. Los elementos de la tarjeta controladora se describen en los próximos capítulos así como también la forma de programarla.

2.3.2 Introducción al Microcontrolador PIC16F877A

Los microcontroladores son computadores digitales integrados en un chip que cuentan con un microprocesador o unidad de procesamiento central (CPU), una memoria para almacenar el programa, una memoria para almacenar datos y puertos de entrada/salida.

El microcontrolador PIC16F877A de Microchip, empleado como cerebro del robot, pertenece a una gran familia de microcontroladores de 8 bits (bus de datos) que tienen las siguientes características generales:

- Arquitectura Harvard
- Tecnología RISC
- Tecnología CMOS

- Soporta modo de comunicación serial.
- Alta memoria para datos y programa.
- Memoria reprogramable.

Estas características se conjugan para lograr un dispositivo altamente eficiente en el uso de la memoria de datos y programa y por lo tanto en la velocidad de ejecución, además de que estas características hacen que este dispositivo se distinga de las demás familias.

En especial, el microcontrolador PIC16F877A, que es el empleado en este proyecto, cuenta con las siguientes características:

- Procesador de arquitectura RISC avanzada.
- Memoria de Programa tipo Flash 8Kx14
- Memoria de Datos 368 bytes
- EEPROM 256 bytes
- 33 pines de Entrada/Salida

Este microcontrolador viene en un encapsulado de 40 pines tipo DIP y trabaja con un voltaje de 2.0 hasta 5.5 VDC y a una frecuencia de 20 MHz

Los periféricos con los que cuenta este microcontrolador son los siguientes:

- 1 conversor A/D de 10-bits (8 canales)
- 2 Módulos CCP (Captura, Comparador, PWM)
- 1 Módulo I^2C
- 1 USART (Puerto Serie)
- 2 Timers de 8 bits
- 1 Timer de 16 bits

El siguiente diagrama muestra como están distribuidos los pines que conforman el encapsulado:

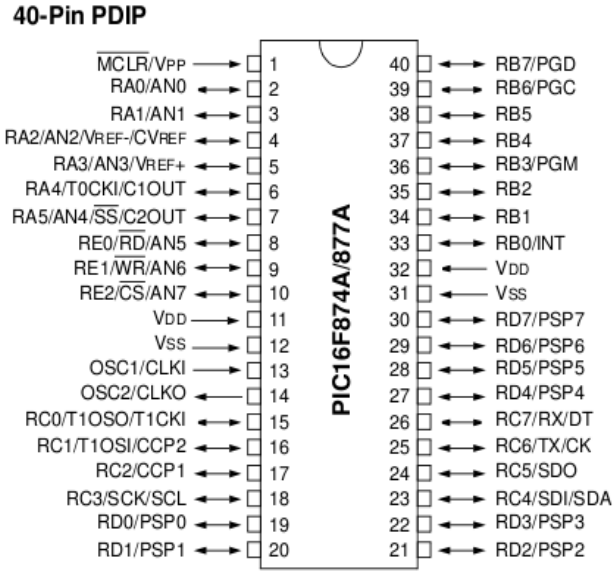


Figura 2.7 Diagrama de pines del PIC16F877A [5]

Donde:

PIN	**DESCRIPCION**
OSC1/CLKIN(9)	Entrada para el oscilador o cristal externo.
OSC2/CLKOUT (10)	Salida del oscilador. Este pin debe conectarse al cristal o resonador. En caso de usar una red RC este pin se puede usar como tren de pulsos o reloj cuya frecuencia es 1/4 de OSC1
MCLR/VPP/ THV(1)	Este pin es el reset del microcontrolador, también se usa como entrada o pulso de grabación al momento de programar el dispositivo.
RA0/AN0(2)	Puede actuar como línea digital de E/S o como entrada analógica del conversor AD (canal 0)
RA1/AN1(3)	Similar a RA0/AN0
RA2/AN2/VREF-(4)	Puede actuar como línea digital de E/S o como entrada analógica del conversor AD (canal 2) o entrada negativa de voltaje de referencia
RA3/AN3/VREF+(5)	Puede actuar como línea digital de E/S o como entrada analógica del conversor AD (canal 3) o entrada positiva de voltaje de referencia
RA4/T0CKI (6)	Línea digital de E/S o entrada del reloj del timer 0. Salida con colector abierto
RA5/SS#/AN4(7)	Línea digital de E/S, entrada analógica o selección como esclavo de la puerta serie síncrona.
RB0/INT(21)	Puerto B pin 0, bidireccional. Este pin puede ser la entrada para solicitar una interrupción.
RB1(22)	Puerto B pin 1, bidireccional.
RB2(23)	Puerto B pin 2, bidireccional.
RB3/PGM(24)	Puerto B pin 3, bidireccional o entrada del voltaje bajo para programación
RB4(25)	Puerto B pin 4, bidireccional. Puede programarse como petición de interrupción cuando el pin cambia de estado.
RB5(26)	Puerto B pin 5, bidireccional. Puede programarse como petición de interrupción cuando el pin cambia de estado.
RB6/PGC(27)	Puerto B pin 6, bidireccional. Puede programarse como petición de interrupción cuando el pin cambia de estado. En la programación serie recibe las señales de reloj.
RB7/PGD(28)	Puerto B pin 7, bidireccional. Puede programarse como petición de interrupción cuando el pin cambia de estado. En la programación serie actúa como entrada de datos
RC0/T1OSO/ T1CKI(11)	Línea digital de E/S o salida del oscilador del timer 1 o como entrada de reloj del timer 1
RC1/T1OSI/ CCP2(12)	Línea digital de E/S o entrada al oscilador del timer 1 o entrada al módulo captura 2/salida comparación 2/ salida del PWM 2
RC2/CCP1(13)	E/S digital. También puede actuar como entrada captura 1,/salida comparación 1/ salida de PWM 1
RC3/SCK/SCL (14)	E/S digital o entrada de reloj serie síncrona /salida de los módulos SP1 e I2C.
RC4/SDI/SDA (15)	E/S digital o entrada de datos en modo SPI o I/O datos en modo I2C
RC5/SDO(16)	E/S digital o salida digital en modo SPI
RC6/TX/CK(17)	E/S digital o patita de transmisión de USART asíncrono o como reloj del síncrono
RC7/RX/DT(18)	E/S digital o receptor del USART asíncrono o como datos en el síncrono
RD0/PSP0-RD7/PSP7 (19-22, 27-30)	Las ocho patitas de esta puerta pueden actuar como E/S digitales o como líneas para la transferencia de información en la comunicación de la puerta paralela esclava. Solo están disponibles en los PIC 16F874/7.
RE0/RD#/AN5 (8)	E/S digital o señal de lectura para la puerta paralela esclava o entrada analógica canal 5.
RE1/WR#/AN6 (9)	E/S digital o señal de escritura para la puerta paralela esclava o entrada analógica canal 6.

PIN	DESCRIPCION
RE2/CS#/AN7	E/S digital o señal de activación/desactivación de la puerta paralela esclava o entrada analógica canal 7.
VSS(8,19)	Tierra.
VDD(20,32)	Fuente (5V).

El PIC16F877A de Microchip pertenece al tipo de procesador RICS (Reduced Instruction Set Computer), que se caracteriza por que el número de instrucciones es pequeño y son ejecutadas en la misma cantidad de tiempo. Este tipo de procesadores emplean una arquitectura de tipo Harvard, es decir, que este tipo de arquitectura dispone de dos memorias independientes (datos e instrucciones) las cuales están conectadas mediante dos buses separados e independientes lo que hace que el microcontrolador pueda acceder a los dos tipos de memorias y las instrucciones se ejecuten en menos ciclos de reloj, este trabajo se realiza mediante procesamiento segmentado o pipeline.

El pipeline permite que se realice simultáneamente la ejecución de una instrucción y la búsqueda del código de la siguiente, de esta manera se ejecuta la instrucción en un ciclo de máquina. De manera ilustrativa podemos verlo en el siguiente diagrama:

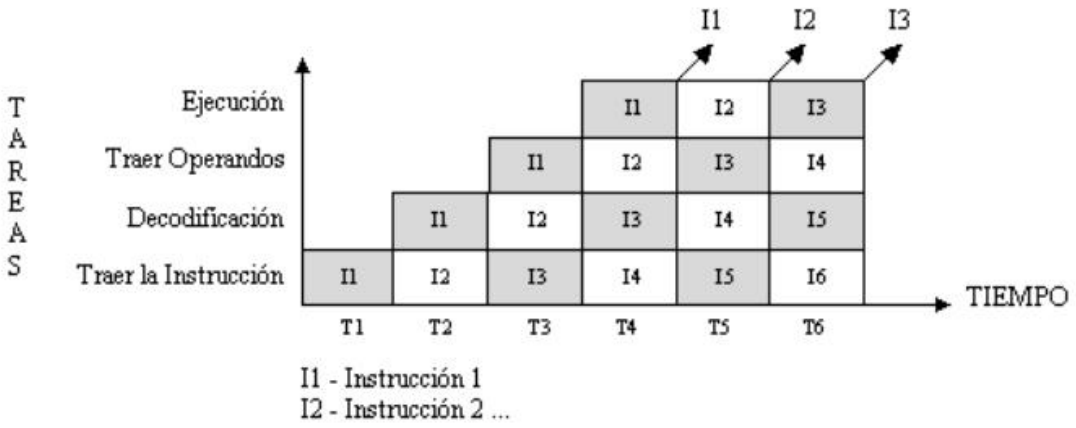


Figura 2.8 Diagrama de bloques del Procesador Pipeline [19]

Se cuenta además de una memoria EEPROM con 256 posiciones, en esta memoria no podemos leer o escribir directamente. Para trabajarla debemos apoyarnos en registros adicionales de tal forma que la usamos indirectamente y estos registros son:

- SFR (Special Function Register): Registros de propósito especial, estos nos ayudan a configurar el hardware interno también nos sirven para escribir o leer valores de los diferentes componentes que constituyen el microcontrolador.
- GFR (General Function Register): Registros de propósito general, son posiciones de memoria que podemos usar para almacenar valores que emplean nuestro programa.

Estos se pueden ver en el siguiente diagrama.

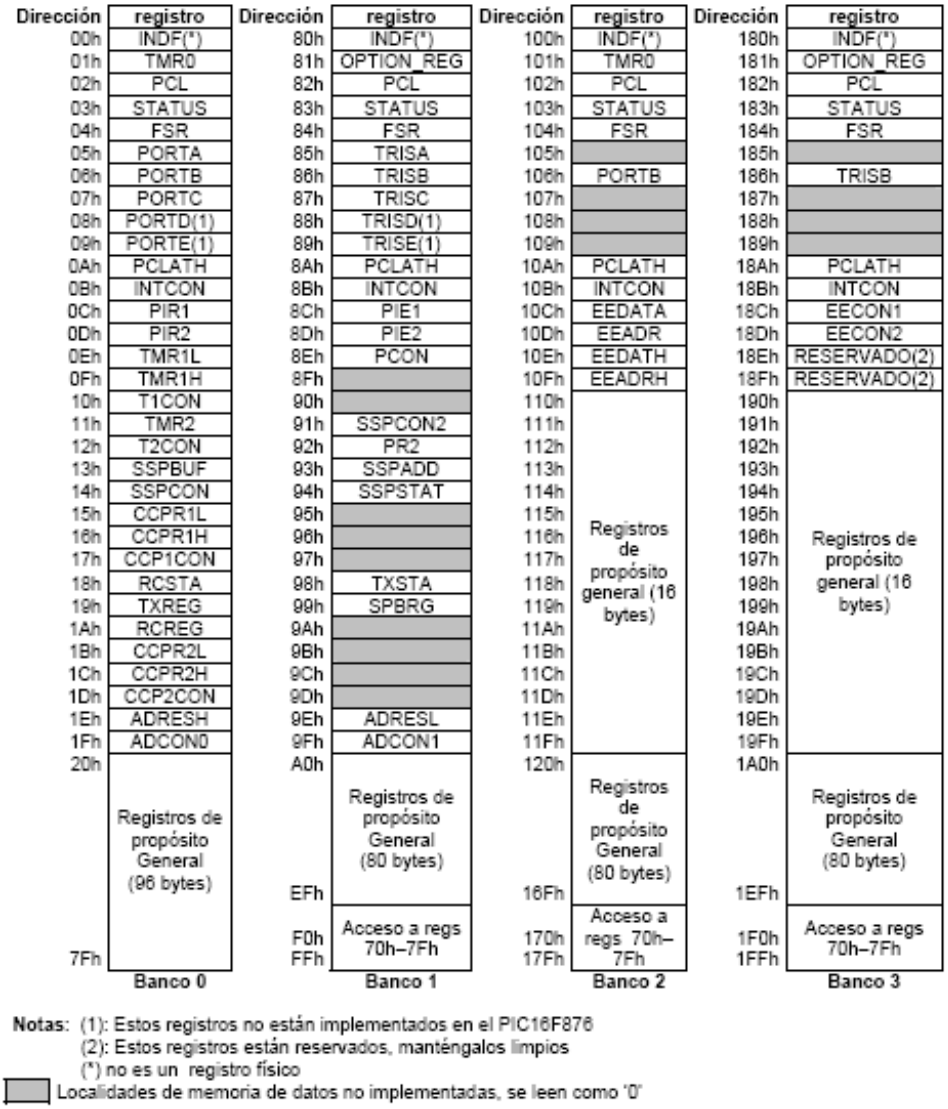
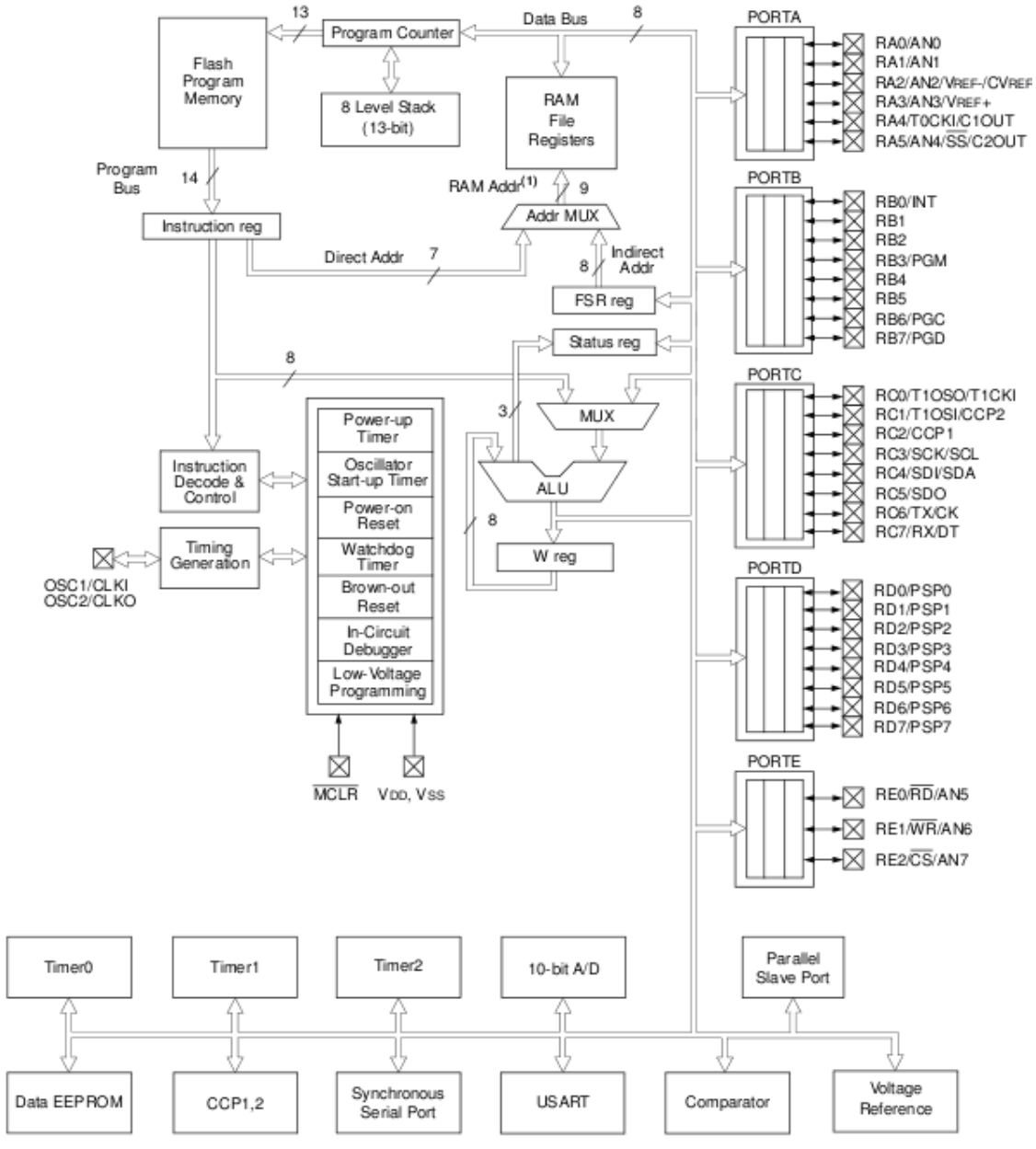


Figura 2.9 Registros del microcontrolador PIC16F877A [5]

Ya hemos mencionado que este microcontrolador posee varias características en forma interna y podemos verlo más claramente en el siguiente diagrama de bloques:



Device	Program Flash	Data Memory	Data EEPROM
PIC16F874A	4K words	192 Bytes	128 Bytes
PIC16F877A	8K words	368 Bytes	256 Bytes

Note 1: Higher order bits are from the Status register.

Figura 2.10 Diagrama interno del microcontrolador PIC16F877A [5]

En el presente diagrama podemos observar como está conformado el microcontrolador, podemos identificar la memoria del Programa en la parte superior izquierda con 8K posiciones por 14 bits, la memoria de datos (RAM) de 368 posiciones por 8 bits, la memoria EEPROM 256 posiciones x 8 bits. Además podemos observar que el procesador esta formado por la ALU (unidad aritmética lógica) el registro de trabajo W, los periféricos de entrada/salida (Port A, B, C, D, E), el TMR0 (temporizador contador de eventos), TMR1 y TMR2 entre otros módulos. También cuenta con un registro de instrucción que se carga cada vez que la ALU solicita una nueva instrucción a procesar.

En la parte intermedia se encuentra el Status que es el registro de estado encargado de anotar el estado actual del sistema. Existe un registro de vital importancia que se llama el Program Counter o contador de programa este registro indica la dirección de la instrucción a ejecutar. También observamos el bloque de la pila y el FSR reg., éste apunta a una dirección de la RAM.

Cabe mencionar que este microcontrolador cuenta con una arquitectura ortogonal, esto quiere decir que la salida de la ALU puede ir al registro W o a la memoria de datos con lo cual, cualquier instrucción puede utilizar cualquier elemento de la arquitectura como fuente o destino.

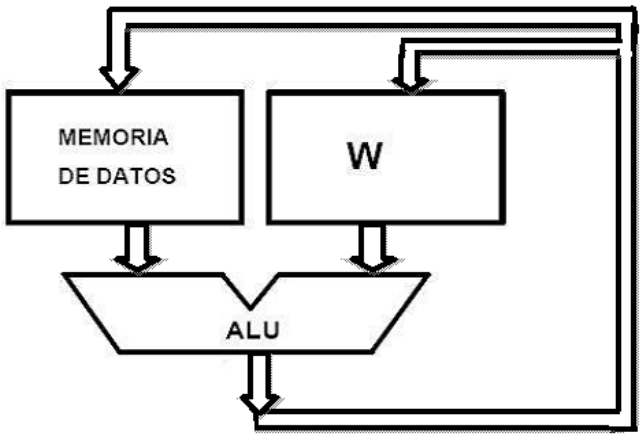


Figura 2.11 Arquitectura Ortogonal.

Una de las herramientas de software que se empleó en este proyecto para generar los archivos .hex para el PIC, fue el Compilador de CCS C Compilers [10] el cual nos permitió compilar código escrito en lenguaje C. En el apartado 2.5 se dará una referencia de como configurar el IDE MPLAB para compilar código escrito en lenguaje C.

2.3.3 Comunicación Serial

La comunicación serial es el proceso de envío de un bit a la vez de modo secuencial a través de bus serial de la computadora. Existen dos tipos de comunicación serial: la comunicación serial síncrona y la comunicación serial asíncrona.

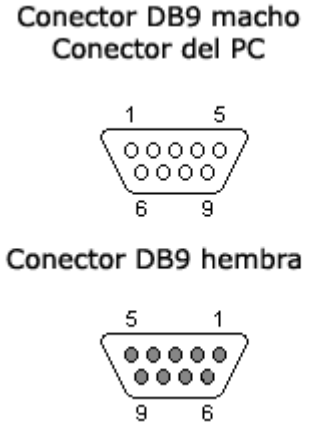
La comunicación serial síncrona necesita de una línea que contenga los pulsos de reloj, el cual indicará cuando el dato es válido y una línea que contenga los datos que se transmitirá. Algunos ejemplos de este tipo de comunicación son:

- I^2C
- ONE WIRE
- SPI

En la comunicación serial asíncrona, los pulsos de reloj no son necesarios, la duración de cada bit es determinada por la velocidad con que se transfieren los datos. En este tipo de comunicación es necesario que el dispositivo receptor sepa cuando se va a iniciar la

recepción de los datos, para ello se tiene la necesidad de delimitar los bits de datos con un bit de inicio y con un bit de parada para saber el inicio y el final del dato enviado; también se utiliza un bit de paridad que es colocado delante del bit de parada con el fin de detectar errores de transmisión. Se puede elegir entre tres tipos de paridad: paridad impar (el número de datos con los que se cuenta es impar), paridad par (el número de datos es par) y sin paridad. Es importante mencionar que tanto el transmisor y el receptor deben de estar sincronizados, para el envío de datos y que ambos deben de tener los mismos parámetros de velocidad, paridad, número de bits del dato transmitidos y el bit de parada.

El puerto utilizado para que el microcontrolador se comunique con un ordenador es el puerto serie RS232, el cual permite la comunicación con otros dispositivos y es compatible con el estándar RS232 o EIA232 Standard. Esta norma establece dos tipos de conectores el DB-25 (con 25 pines) y el DB-9 (con 9 pines), machos y hembras.



PIN	SEÑAL
1	Detección de portadora (DCD)
2	Recepción de datos (RxD)
3	Transmisión de datos (TxD)
4	Terminal de datos preparado (DTR)
5	Tierra de señal (GND)
6	Dispositivo preparado (DSR)
7	Petición de envío (RTS)
8	Preparado para transmitir (CTS)
9	Indicador de llamada entrante (RI)

Figura 2.12 Diagrama del conector DB-9

Para propósitos de este proyecto se utilizó el DB-9 para realizar la comunicación serial entre el microcomputador y el ordenador.

La norma RS232 establece los siguientes niveles de voltaje:

- Los datos se transmiten con lógica negativa, es decir, un voltaje positivo en la conexión representa un “0” y un voltaje negativo representa un “1”.
- Para garantizar que se tiene un “0” lógico una línea debe mantener un voltaje entre +3 y +15 [V].
- Para garantizar un “1” lógico la línea debe de estar entre -3 y -15 [V].
- Los voltajes usados son +12 para el “0” y -12 para el “1”.
- Cuando un puerto serie no está transmitiendo mantiene el terminal de transmisión a “1” lógico a -12 V, normalmente.
- La región de transición es la banda muerta comprendida entre los valores de +3 V y -3 V aquí los niveles lógicos no están definidos.

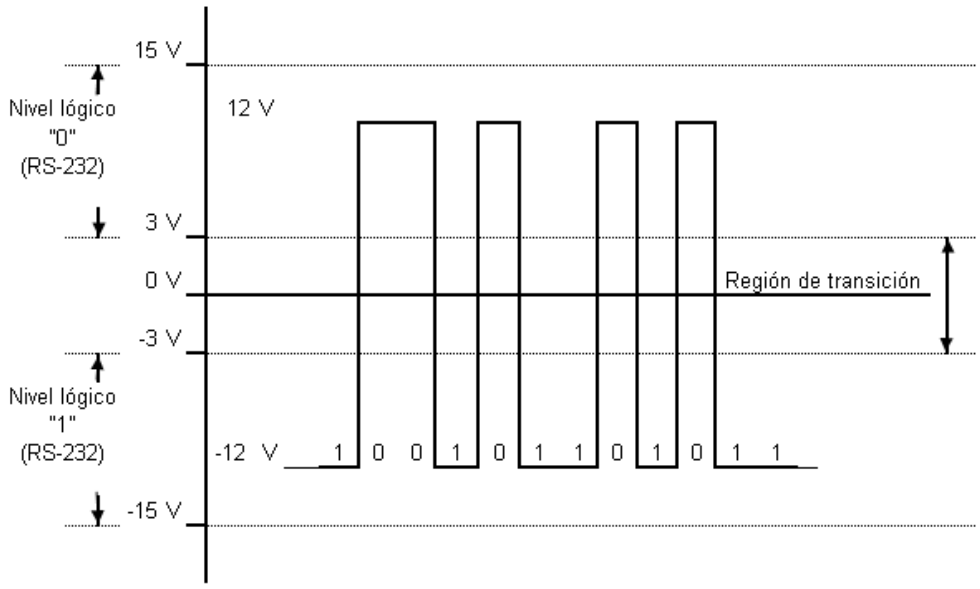


Figura 2.13 Niveles de tensión lógicos para RS-232

Si la velocidad de transmisión de señales de datos aumenta, estas se vuelven susceptibles a pérdidas de voltaje causadas por la capacidad, resistencia e inductancia del cable y aumenta con la longitud del cable.

La comunicación de datos efectuada en un puerto serie RS232 es usado para efectuar comunicaciones asíncronas. Estos datos llegan en paquetes de información normalmente de 8 bits.

El protocolo establecido por la norma RS232 envía la información estructurada en 4 partes:

- Bit de inicio o arranque (Start). Es el paso de un "1" a un "0" lógico en la lógica negativa de la noma RS232. Cuando el receptor detecta el bit de inicio sabe que la transmisión ha comenzado y entonces, debe leer las señales de la línea a distancias concretas de tiempo en función de la velocidad fijada por el emisor y receptor.
- Bits de datos (Datos). Estos son enviados al receptor después del bit start. El bit de menos peso LBS (Least Significant Bit) es transmitido primero y el de mayor peso MBS (Most Significant Bit) el último.
- Bit de parada (Parity). Con este bit se suelen descubrir errores en la transmisión. Puede ser paridad par o impar.
- Bit de parada (Stop). Indica la finalización de la transmisión de una palabra de datos, este protocolo permite 1, 1.5 ó 2 bits de parada.

Lo descrito anteriormente lo podemos ilustrar con un ejemplo, en este se muestra el envío de una palabra de 7 bits (1011010) que corresponde a la letra "Z" en código ASCII, un bit de paridad par y luego dos bits de paro. Como podemos observar en la figura, el microcontrolador trabaja con lógica positiva a diferencia del puerto RS232 que trabaja con lógica negativa.

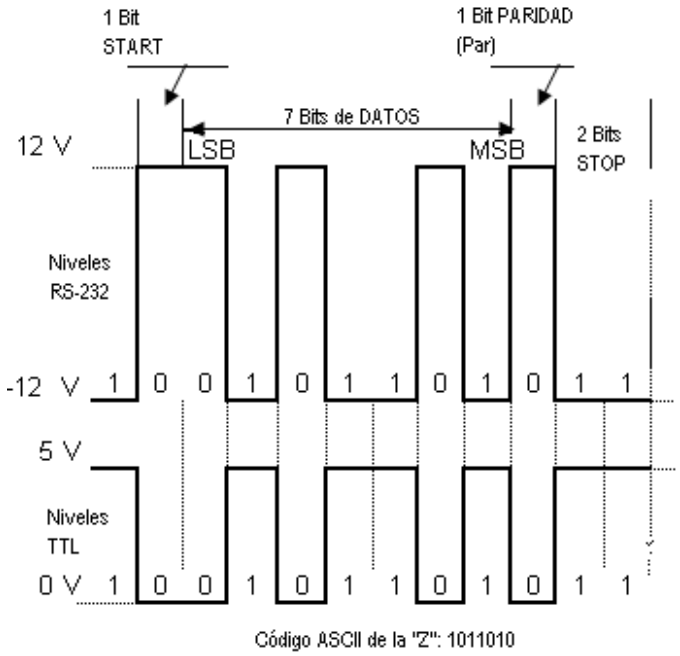
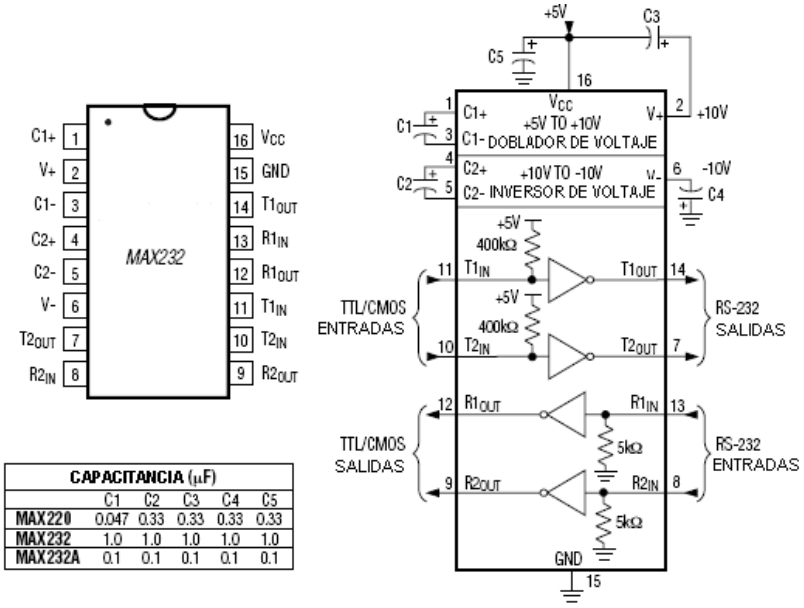


Figura 2.14 Ejemplo de envío de un byte según la norma RS232

La importancia de conocer esta norma radica en que los niveles de voltaje que maneja este puerto, el microcontrolador y otros circuitos son diferentes. Para permitir la adecuada conversión de estos niveles de voltaje se utiliza el transceptor MAX232.



CAPACITANCIA (µF)					
	C1	C2	C3	C4	C5
MAX220	0.047	0.33	0.33	0.33	0.33
MAX232	1.0	1.0	1.0	1.0	1.0
MAX232A	0.1	0.1	0.1	0.1	0.1

Figura 2.15 Diagrama del transceptor MAX232 [20]

Este circuito soluciona el problema de los niveles de voltaje entre la línea RS-232 y los circuitos TTL. Para este proyecto se utilizó este circuito por que solo necesita de una fuente de +5 V para operar y cumple con la norma RS232.

Para comunicarse con el PIC16F877A se utilizan sólo tres patillas del puerto serie. Éstas son:

- Pin de transmisión (TxD).
- Pin de recepción (RxD).
- Pin de masa (SG).

Como ya habíamos mencionado existe el problema de los niveles voltaje entre el puerto RS232 y el microcontrolador PIC16F877A, para resolver este inconveniente utilizamos el transceptor MAX232, el cual como ya se mencionó anteriormente nos permitirá la traducción de estos niveles lógicos de voltaje.

2.4 Actuadores

Un actuador es un dispositivo de funcionamiento eléctrico, neumático o hidráulico, que actúa como un motor para cambiar la posición de aparatos móviles, para los objetivos de este proyecto se utilizaron actuadores eléctricos.

Entre los diferentes actuadores eléctricos podemos encontrar:

- Motores de corriente continua (DC).
- Servomotores.
- Motores paso a paso.

En siguientes apartados se mencionará los actuadores empleados en este proyecto.

2.4.1 El motor de DC

El motor DC o de corriente directa convierte la energía eléctrica en energía mecánica, por medio del movimiento rotatorio.

Una de sus principales características es que su velocidad puede ser regulada.

Un problema que se tiene al emplear este tipo de motores en proyectos con microcontroladores es la forma de alimentarlos ya que la corriente máxima que proporciona el microcontrolador es de 25 mA y el motor necesita de más corriente para trabajar, por lo que es necesario utilizar de dispositivos electrónicos que le puedan proveer la corriente necesaria al motor entre los que podrían ser los transistores, relevadores entre otros. Para el control de un motor de DC se requiere de una configuración en Puente H de estos dispositivos. En la sección 2.4.1.1 se hablará de los tipos de Puente H y del empleado en este proyecto.

2.4.1.1 Puente H

El Puente H es un circuito electrónico que permite controlar al motor de DC, en velocidad y sentido de giro.

Existe gran variedad de tipos de puente H, desde los formados por transistores hasta los formados por relevadores.

El puente H con transistores está formado por cuatro transistores que trabajan en conmutación y se comportan como interruptores controlados por la señal que les llega a las entradas.

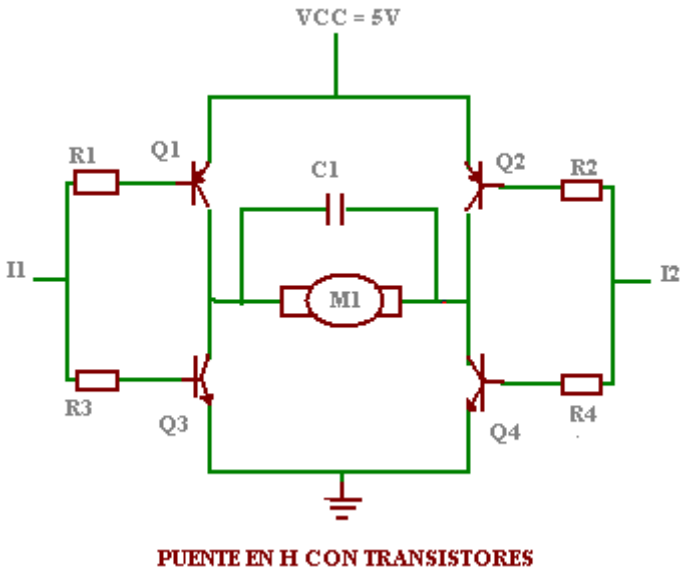


Figura 2.16 Puente en H con transistores.

El sentido de giro de un motor de corriente continua depende de la polaridad que se aplica en las terminales del puente H.

Si lo que deseamos es que nuestro motor gire en un sentido o en el otro debemos tener la siguiente configuración:

- Activar la entrada I1 a nivel alto y la entrada I2 a nivel bajo los transistores Q3 y Q2 entran en saturación mientras Q1 y Q4 entran en corte; así en esta condición el motor girará en sentido contrario al de las manecillas de reloj

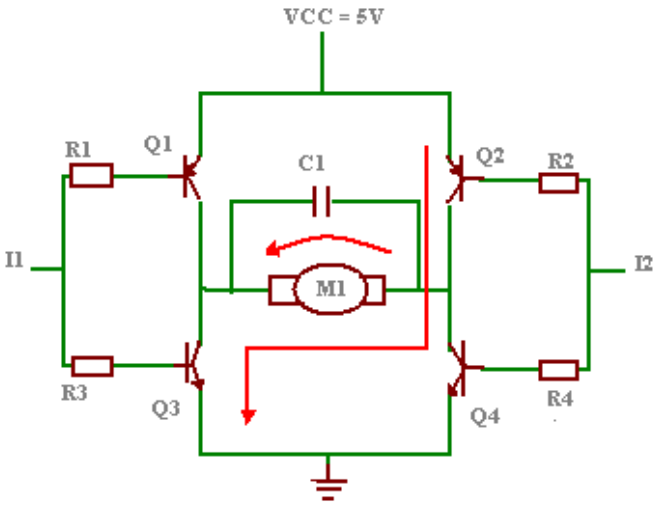


Figura 2.17 Movimiento del motor en sentido antihorario.

- Si activamos la entrada I1 a nivel bajo y la entrada I2 a nivel alto, los transistores que entran en saturación son Q1 y Q4 mientras que Q2 y Q3 entran en corte; es decir realizamos la configuración inversa a la anterior, esto con el fin de que el motor gire en sentido de la manecillas del reloj.

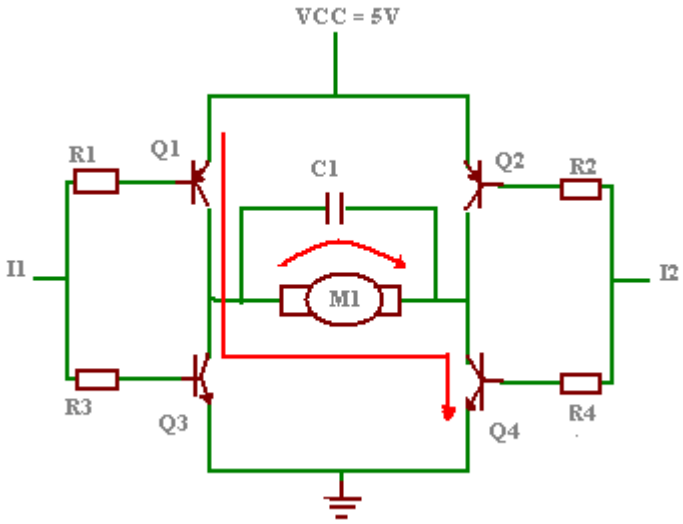


Figura 2.18 Movimiento del motor en sentido horario.

El problema que existe con este tipo de configuración es la caída de tensión real que hay en los transistores y se tendría que compensar con la tensión de alimentación; además de que para fines de nuestro proyecto la circuitería empleada sería demasiado grande para el espacio destinado de el robot.

2.4.1.2 Puente H con relevadores

Un relevador es un dispositivo electromecánico que funciona como un interruptor, el cual es controlado por un circuito eléctrico. Para poder cambiar el estado del interruptor es necesario aplicar un voltaje de activación que depende de la construcción física del relevador (normalmente 5 V). La figura 2.19 muestra el diagrama de un relevador.

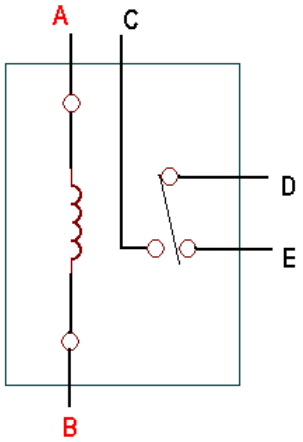


Figura 2.19 Diagrama de un relevador.

El relevador de la figura 2.20 contiene cinco terminales, la terminal A y B corresponden a la bobina del relevador y las terminales C, D y E corresponden a las terminales de los contactos mecánicos, el estado normal del relevador es cuando D y E tiene continuidad y para poder modificar el estado del interruptor se requiere un voltaje igual al voltaje de activación del relevador entre las terminales A y B que a partir de ese momento se active el electroimán y conecta los puntos D y E.

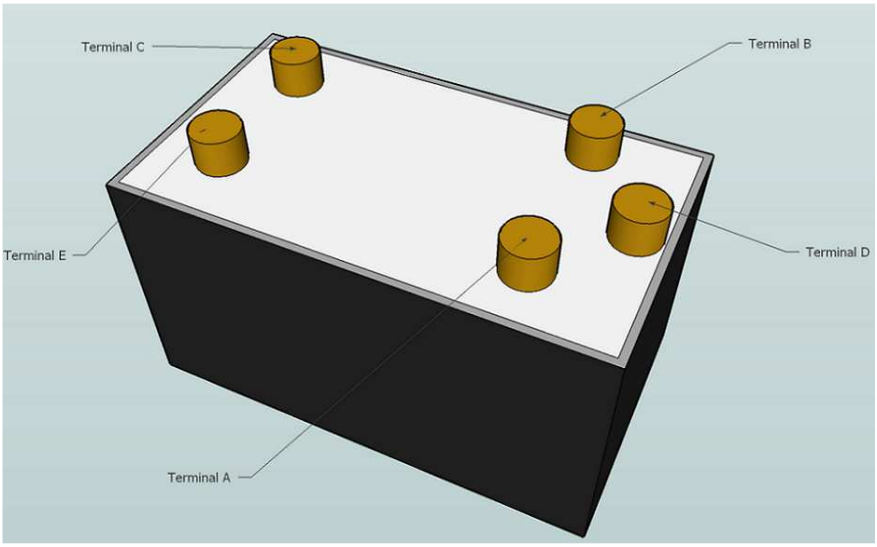


Figura 2.20 Terminales de un relevador.

Un microcontrolador no es capaz de dar la corriente necesaria para activar a un relevador por lo que se requiere de un par de transistores Darlington para dicha activación, la figura 2.21 muestra como conectar un relevador a una salida del microcontrolador. Cuando la salida del microcontrolador proporciona un nivel alto a la base del Darlington, pasa conducción y activa el relevador. Es necesario colocar un diodo en paralelo con la bobina del relevador como una manera para proteger de los picos de fuerza electromotriz producidos por la carga de la bobina en el momento de la conmutación.

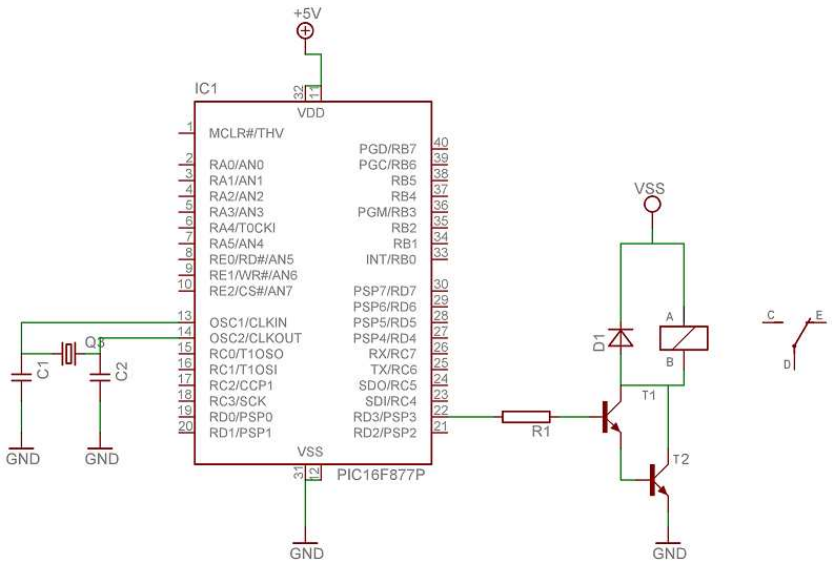


Figura 2.21 Conexión de un relevador a la salida del PIC.

En muchos casos es necesario controlar muchos relevadores, lo que ocasionaría un gran número de componentes conectados al microcontrolador para evitar este problema existe un circuito integrado especializado en el control de relevadores este es el driver ULN2003 [17] que aguanta hasta una tensión máxima de 50 V y puede alimentar cargas de hasta 500 mA además de contar con el diodo de protección, permitiéndonos manejar hasta siete relevadores.

Un puente H con relevadores no conmutan tan rápido como los formados con transistores, pero nos permiten controlar cargas que demanden más de 5 A. La figura 2.22a muestra la configuración del puente H empleando relevadores.

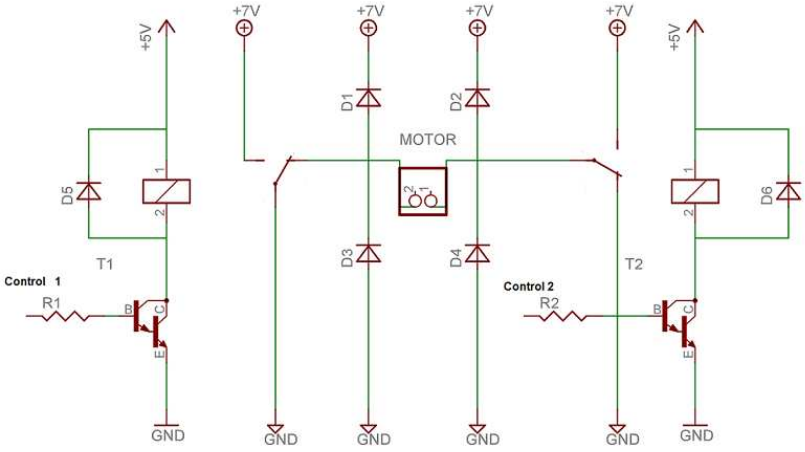


Figura 2.22a Puente H con relevadores.

El funcionamiento del puente H con relevadores es el siguiente:
Cuando la señal de control 1 y la señal de control 2 son iguales a 0 no existe conmutación de los contactos mecánicos de los relevadores y el estado del motor es apagado (ver figura 2.22a), cuando la señal de control es igual a 5 V y la señal de control 2 es igual a 0 V el relevador 1 conmuta originando un flujo de corriente como se muestra en la figura 2.22b lo cual ocasiona que el motor gire en el sentido mostrado en la figura. Para que el motor gire en sentido contrario la señal de control 1 deberá ser igual a 0 V y la señal de control 2 igual a 5 V como lo muestra la figura 2.22c. Para parar nuevamente al motor las dos señales de control deberán ser igual a 0 V y nunca las dos a 5 V.

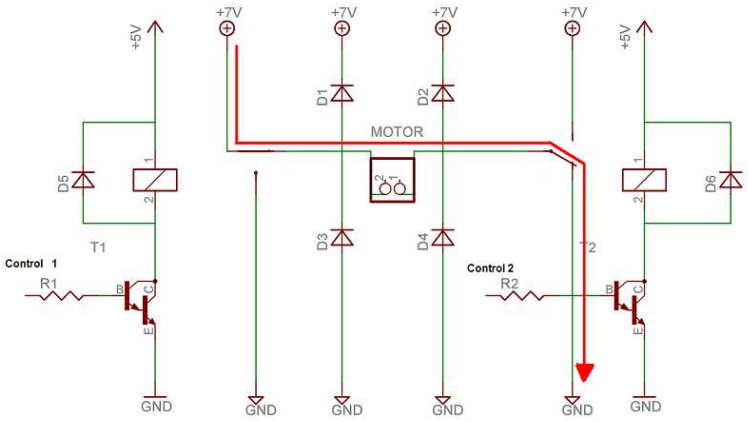


Figura 2.22b Señal de control 1 igual a 5 V y control 2 igual a 0 V

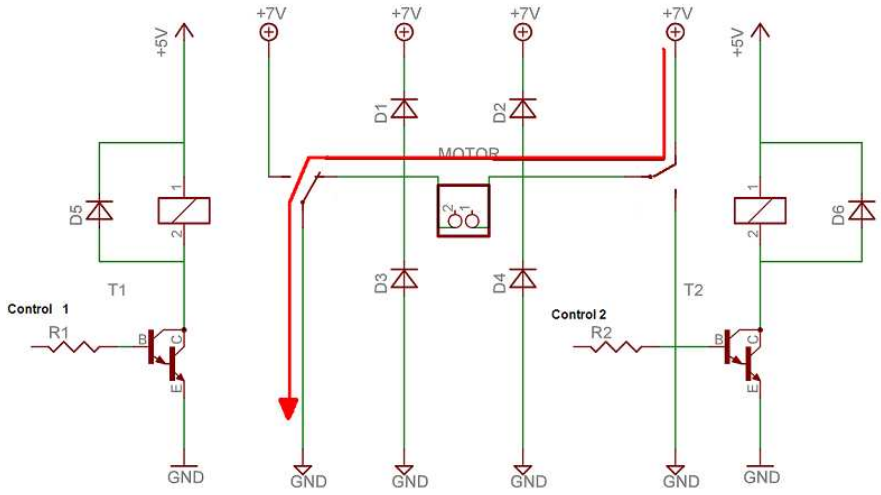
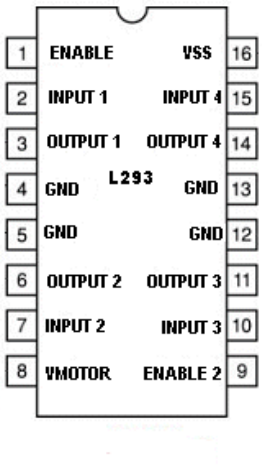


Figura 2.22c Señal de control 1 igual a 0 V y control 2 igual a 5 V

2.4.1.3 El L293 como controlador de motores de DC

El circuito integrado L293 es capaz de proporcionar una corriente de salida de hasta 1 A por canal. Está compuesto por cuatro canales para ingresar las señales de control a los motores y cada uno de estos acepta niveles estándares de lógica TTL y DTL, además de que cada pareja dispone de una señal de habilitación que desconecta las salidas de los mismos.

El chip está incluido en un diseño DIP de 16 pines y es capaz de operar con voltajes máximos de 36 volts.



PIN	NOMBRE	DESCRIPCIÓN
1	Chip Enable 1	Habilitación de los canales 1 y 2
2	Input 1	Entrada del canal 1
3	Output 1	Salida del canal 1
4,5	GND	Masa alimentación
6	Output 2	Salida del canal 2
7	Input 2	Entrada del canal 2
8	VMOTOR	Alimentación de las cargas
9	Chip Enable 2	Habilitación de los canales 3 y 4
10	Input 3	Entrada del canal 3
11	Output 3	Salida del canal 3
12,13	GND	Masa alimentación
14	Output 4	Salida del canal 4
15	Input 4	Entrada del canal 4
16	Vss	Alimentación del chip

Figura 2.23 Diagrama de pines del L293 [20]

Este circuito permite controlar los movimientos del motor ya sea hacia la izquierda, a la derecha, hacia adelante, hacia atrás o pararlo; estos movimientos podemos lograrlos con la previa configuración del chip, es decir, el valor que se le dará a los INPUT y también podemos configurar la velocidad del motor. Para detectar los movimientos de los motores de nuestro robot empleamos las siguientes señales para el funcionamiento de los drivers:

V enable	V input	V output
H	H	H
H	L	L
L	H	Z
L	L	Z

Donde:
H: Nivel alto "1"
L: Nivel bajo "0"
Z: Impedancia

Tabla de funcionamiento del Driver L293.

Símbolo	Parámetro	Valor
Vs	Tensión de alimentación para las cargas	36 V
Vss	Tensión de alimentación de la lógica	36 V
Vi	Tensión de entrada	7 V
Vinh	Tensión de habilitación	7 V
Iout	Intensidad de pico de salida	2 A
Ptot	Potencia total de disipación	5 W

Rangos absolutos del driver L293B

El circuito L293D a diferencia del L293B entrega una corriente máxima de 600 mA, para fines del este proyecto se utilizó el circuito L293D.

2.4.2 Servomotores

Los servomotores son un tipo especial de motores de DC que se caracterizan por la capacidad de posicionarse de forma inmediata en cualquier posición angular específica siempre y cuando este dentro de su intervalo de operación, para realizar esto el servomotor espera un tren de pulsos el cual corresponde al movimiento que se va a realizar.

Un servomotor está formado por un pequeño motor de corriente continua, unas ruedas dentadas, lo que le da una potencia considerable, y una tarjeta de circuito impreso con lo necesario para su control.

La tensión de alimentación de los servos suele estar comprendida entre los 4 y 8 voltios. El control de un servo se limita a indicar en que posición se debe situar, mediante una señal cuadrada TTL modulada en anchura de pulsos PWM. La duración del nivel alto de la señal indica la posición donde queremos poner el eje del motor. El potenciómetro que el servomotor tiene indica al circuito eléctrico del control interno si éste ha llegado a la posición deseada.

La duración de los pulsos indica el ángulo de giro del motor. Cada tipo de servomotor tiene sus márgenes de operación, que corresponden con el ancho de pulso máximo y mínimo que el servo entiende y que no debe de sobrepasar.

Es importante decir que para que un servomotor conserve la misma posición debe de enviarse un pulso con una anchura constante. Estas posiciones son mostradas en la figura 2.24.

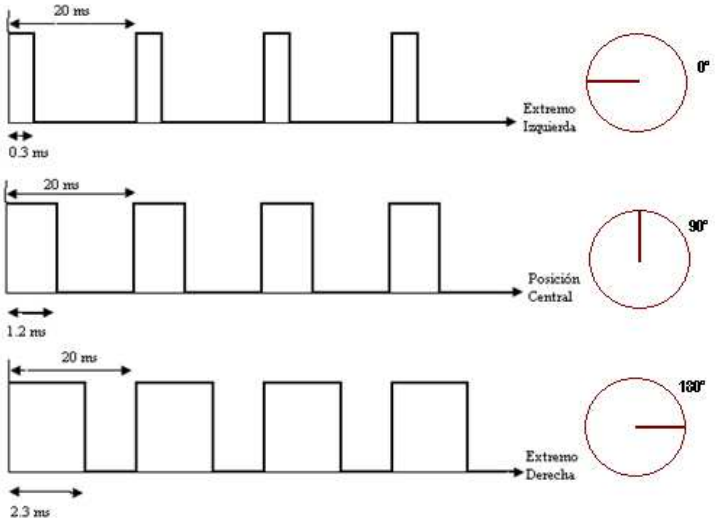


Figura 2.24 Tren de impulsos para el control de un servomotor.

El periodo entre pulso y pulso no es crítico. Se suelen emplear valores entre 10 ms y 30 ms, aunque comúnmente se utilizan 20 ms, esto implica una frecuencia de 50 Hz. Si el intervalo entre pulsos es inferior al mínimo la estructura interna del servo causa un zumbido y si excede los límites máximos el servo pasa a estado inactivo.

Como ya hemos mencionado anteriormente un servomotor es un motor eléctrico que sólo se puede mover aproximadamente en un ángulo de 180 grados, estos disponen de tres terminales:

- Positivo de alimentación unido al cable de color rojo.
- Masa o negativo, que casi siempre es negro.
- Señal por donde se aplica la entrada de impulsos, suele ser de color blanco amarillo o naranja.

La figura 2.25 se muestra las terminales de un servomotor.

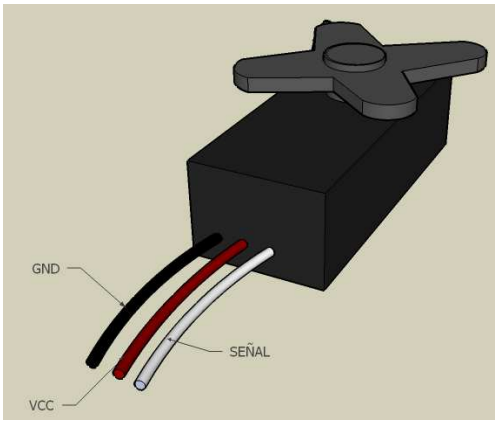


Figura 2.25 Terminales del servomotor.

2.5 Configuración del IDE MPLAB para Windows

Para programar el microcontrolador PIC16F877A se requiere el IDE MPLAB y el compilador CCS, toda la información necesaria para instalar y configurar MPLAB se encuentra en la página de Microchip [1] y la instalación y configuración del compilador CCS se encuentra en la página del compilador [10].

2.6 Ejemplos de programación en MPLAB

En esta sección se muestran dos ejemplos de programación en lenguaje C para PIC's, los cuales son básicos para comprobar el buen funcionamiento de los mismos. Los programas escritos en lenguaje C para PIC's se muestran en el apéndice A.

2.6.1 Programa que recibe y envía datos por el puerto serial

El programa que se muestra a continuación (ver programa 1 del apéndice A) es un ejemplo de comunicación mediante el puerto serial de una computadora y el microcontrolador. Este programa recibe una cadena dada por el usuario mediante una hyperterminal y a continuación el microcontrolador devuelve la misma cadena a la computadora para ser visualizada en la hyperterminal.

2.6.2 Programa que imprime el estado de los bumpers conectados en el puerto A

Los bumper son sensores de contacto que nos sirven para que el robot detecte los obstáculos que se le presenten delante de él y así pueda evadirlos en el caso que exista un contacto físico con un objeto. El siguiente programa (ver programa 2 apéndice A) lee el puerto A del microcontrolador al cual se le han colocado cinco bumper's, en el caso de que un contacto se presione se registrará un cero de lo contrario existirá un uno y será enviado a la hyperterminal del usuario.

CAPÍTULO III EL SISTEMA EMBEBIDO DE VISION

3.1 Introducción

En esta sección se dará una breve introducción a los elementos más importantes que conforman a la imagen digital, para después pasar a ver los sistemas embebidos de visión. Para un estudio detallado de la teoría de la imagen digital puede consultar el libro Digital Image Processing de González, R. C.

Una imagen digital en escala de grises es una matriz de MxN elementos numéricos, cuyos valores posibles van del 0 (negro) al 255 (blanco), siendo este número la intensidad luminosa en un píxel. El píxel es la abreviatura de Picture Element, que es la menor unidad homogénea de color que conforman a la imagen digital. Una imagen digital a colores esta formada por 3 matrices de MxN elementos numéricos, cuyos valores posibles van del 0 (negro) al 255 (blanco), siendo este número la intensidad luminosa en cada una de las bandas espectrales del RGB (Rojo, Verde, Azul), de cada punto o píxel, a diferencia de las imágenes en escala de grises, las imágenes a color requiere de la combinación de las 3 bandas de color, para representar el color de un píxel. El modelo RGB para un color hace referencia a la composición del color en términos de la intensidad de los colores primarios con que se forma el color, siendo estos el rojo, el verde y el azul, con lo que es posible representar un color mediante la mezcla por adición de los tres colores primarios. Para que indiquemos la porción de color que estamos mezclando, se asigna un valor a cada uno de los colores primarios, donde el valor se encuentra nuevamente entre 0 y 255, para 0 se indica que no se aporta nada de ese color y a medida que se va incrementando este número la intensidad para la mezcla de ese color aumentará, cuando se llegue a 255 se tendrá el tono más obscuro del color. Si queremos obtener el color rojo la combinación de colores en el modelo RGB seria (255,0,0) para el verde (0,255,0) para el azul (0,0,255) y en el caso de la ausencia del color (0,0,0) que correspondería al negro y el blanco seria cuando los tres colores primarios están en su máximo nivel (255,255,255).

3.2 Sistemas embebidos

Los sistemas embebidos son sistemas electrónicos hechos para tareas específicas, consisten en un microprocesador cuyo software y hardware están específicamente diseñados para resolver una tarea concreta. En el mercado existen muchos sistemas embebidos para controlar el funcionamiento de dispositivos como los televisores, lavadoras, alarmas, teléfonos, computadoras entre otras. En las computadoras podemos encontrar controladores de disco, controladores de puertos seriales y USB, controladores de red entre otros; todos ellos son sistemas embebidos desarrollados para tareas específicas que proporcionan una interfaz bien definida. Los sistemas embebidos proporcionan interfaces mediante las cuales se pueden comunicar con el exterior, pueden ser puertos de entrada y salida de datos, puertos seriales, puertos paralelos, puertos USB, puertos de red, salidas de potencia, o como en nuestro caso, una interfaz para una cámara digital.

3.2.1 Sistema embebido de visión

Anteriormente no era posible incluir una cámara digital en un proyecto de robótica, ya que esto requiere de una alta velocidad de procesamiento y los microcontroladores no tienen el hardware necesario y la velocidad de procesamiento para poder utilizar una cámara digital con ellos, por ello es que existen sistemas especializados en el procesamiento de imágenes como los sistemas embebidos de visión que contienen el hardware y software necesario para el procesamiento de imágenes, el presente trabajo se enfoca al sistema embebido de visión CMUcam3 del que se hablará en las siguientes secciones.

3.3 Introducción a la CMUcam3

La CMUcam3 es un sistema embebido desarrollado por la Universidad de Carnegie Mellon, con el objetivo de desarrollar un sistema de visión a un bajo costo en su hardware y la posibilidad de crear software libre para su interfaz de programación, está compuesta por una cámara en formato digital y un sistema de desarrollo de código abierto, la figura 3.1 muestra una vista general de la CMUcam3. La CMUcam3 es un sistema embebido de visión basado en el ARM7TDMI [6], su procesador principal es el NXP LPC2106 [7], de 32 bits, que va conectado al módulo del sensor de la cámara CMOS creado por omnivision [8]. La CMUcam3 es programada con librerías de código abierto de GNU TOOLCHAIN [9], que trae herramientas para escribir código en lenguaje C para ella, así como un IDE que nos generará el código ejecutable y la ventaja de programarlo por el puerto serie sin necesidad de utilizar hardware adicional.

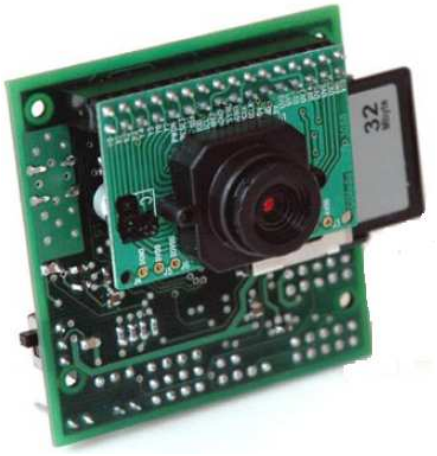


Figura 3.1 La CMUcam3 [2]

Las características principales de la CMUcam3 son:

- Sensor RGB en color con una resolución de 352 x 288.
- Ambientes de desarrollo para Windows y Linux.
- Conector para tarjeta SD o MMS con soporte FAT 16.
- Cuatro Puertos para controlar servos.
- Carga imágenes en memoria a 26 tramas por segundo.
- Compresión por software en modo JPEG.
- Emulación de la CMUCAM2.
- Puerto de comunicación serial.

3.3.1 Hardware de la CMUcam3

El hardware de la CMUcam3 está basado en una arquitectura ARM7TFMI [6] con un procesador NXP LPC2106 y puertos de propósito general. La figura 3.2 muestra la interfaz de hardware que proporciona la CMUcam3.

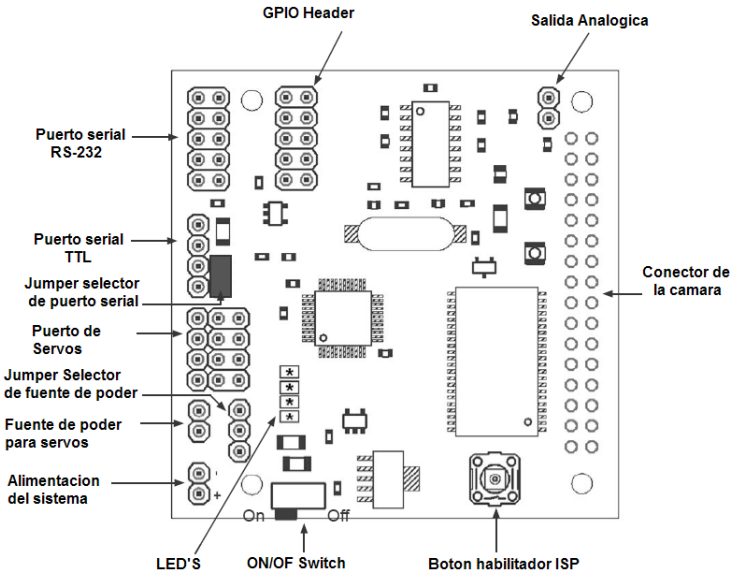


Figura 3.2 Hardware de la CMUcam3 [11]

3.3.1.1 Alimentación de la CMUcam3

La primera conexión de hardware es la alimentación del sistema, ésta proporciona al sistema una alimentación de 5 volts a través de un regulador de voltaje, ideal para conectarle una fuente externa entre 6 y 15 volts de una fuente de DC y que sea capaz de suministrar al menos 150 [mA] de corriente. Para poder alimentar a los servomotores es posible utilizar la alimentación interna de la CMUcam3 o utilizar una alimentación externa, con lo cual se deberá remover el Jumper de selector de fuente de poder, para que de esta manera los servos utilicen alimentación externa. En la figura 3.3 muestra esta configuración para los servos.

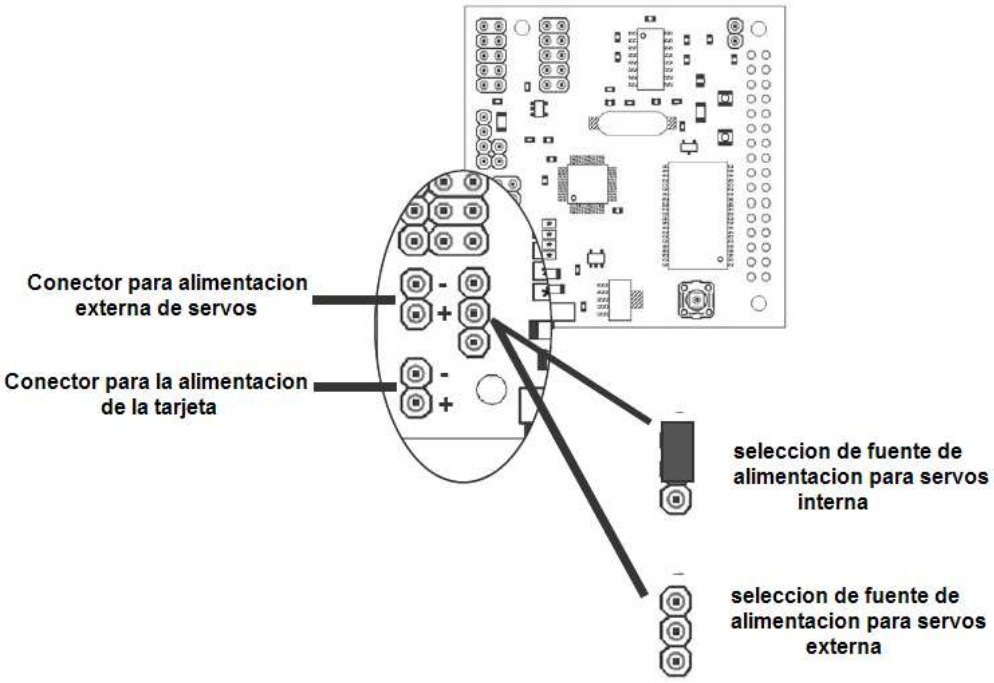


Figura 3.3 Configuración de la fuente de alimentación para servos [11]

3.3.1.2 Puerto Serial

La CMUcam3 cuenta con un puerto de comunicación serial, el cual puede ser configurado mediante hardware para poder comunicarse con una computadora a los niveles de voltaje que marca la norma RS232 o comunicarse a niveles TTL con otro microcontrolador. La figura 3.4 muestra como se debe hacer esta configuración que dependerá del estado del Jumper, por ejemplo si se desea programar a la CMUcam3 o comunicarse con la computadora el Jumper no se deberá mover, por lo contrario si lo que se desea es comunicarse con un microcontrolador a niveles TTL, el Jumper se deberá remover, dejando de esta manera habilitado el puerto serie con niveles TTL. La figura también muestra los tres pines que se utilizan para la comunicación serial siendo estos el pin de transmisión (Tx), el de recepción (Rx) y tierra (GND).

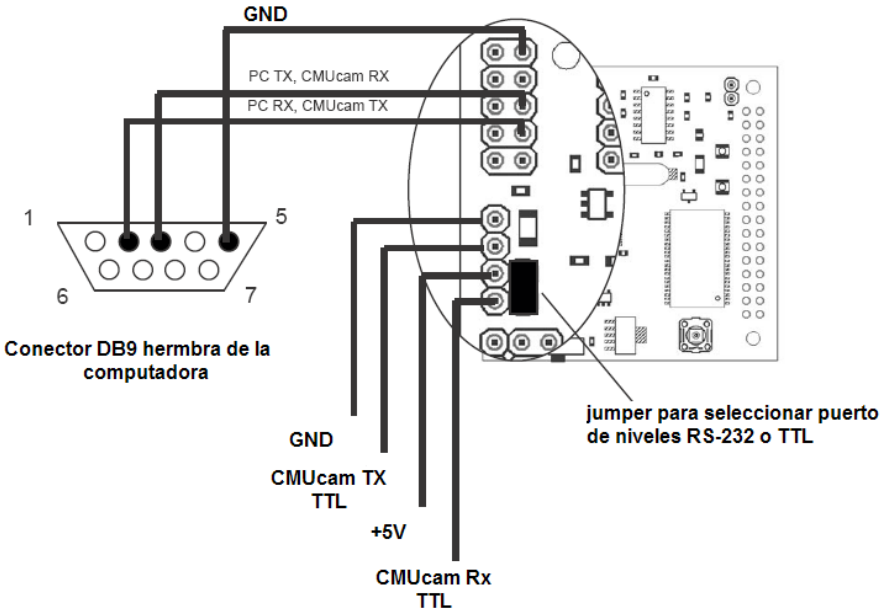
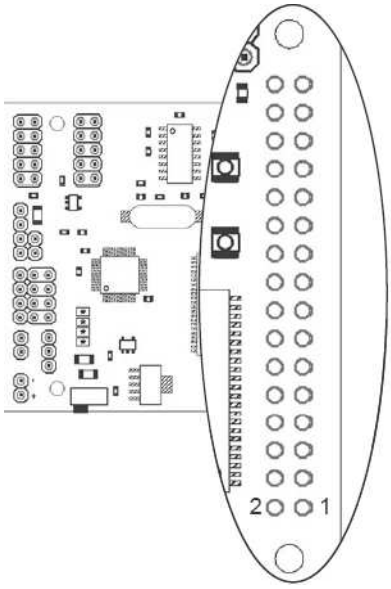


Figura 3.4 Puerto serial de la CMUcam3 [11]

3.3.1.3 BUS de la cámara

La CMUcam3 tiene un puerto destinado para la conexión de una cámara digital por la cual se le pasaran los datos que se requieren para el tratamiento de la imagen digital, las entradas de esta interfaz se detallan en la figura 3.5 donde se describe a cada uno de los pines que lo conforman.



PIN	Nombre	Descripción
1-8	Y0-Y7	Salidas digitales del bus Y
9	PWDN	Modo Power Down
10	RST	Reset
11	SDA	Línea de datos del bus serial I2C
12	FODD	Bandera de campo impar
13	SCL	Reloj del I2C
14	HREF	Referencia horizontal
15	AGND	Tierra analógica
16	VSYNC	Vertical Sync
17	AGND	Tierra analógica
18	PCLK	Reloj de pixel
19	EXCLK	Reloj externo
20	VCC	+5VDC
21	AGND	Tierra analógica
22	VCC	+5VDC
23-30	UV0-UV7	Salidas digitales del bus UV
31	GND	Tierra
32	VTO	Salida de video (75 Ohm)

Figura 3.5 Interfaz de la CMUcam3 para una cámara digital [11]

3.3.1.4 Puerto para servomotores

La CMUcam3 cuenta con un puerto para servos, mediante el cual puede controlar hasta cuatro servos además de tener la opción de elegir la fuente de alimentación interna de la

tarjeta o utilizar una fuente externa para la alimentación de los servos. En la sección 3.3.1.1 se encuentra la descripción de este último punto. En la figura 3.6 se muestra el puerto que posee la CMUcam3 para manejar servos, así mismo se muestra la descripción de cada pin del puerto.

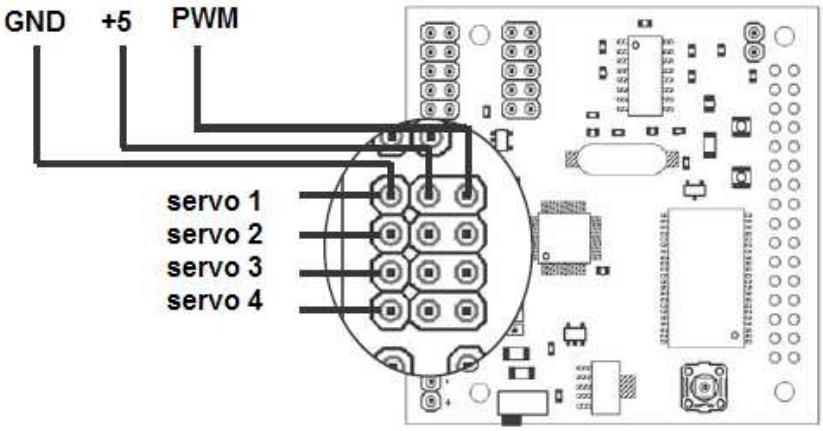


Figura 3.6 Puerto para controlar los servos [11]

La CMUcam3 cuenta con dos puertos más, los cuales son: el puerto analógico de salida y el puerto de expansión GPIO los cuales en este trabajo no se mencionan. Si quiere un estudio detallado de estos puertos véase la referencia número 11 de este trabajo.

3.3.1.5 Botón ISP

El botón ISP nos permite elegir el modo en el que la tarjeta debe iniciarse (ver figura 3.7). Cuando alimentamos a la tarjeta, y el botón ISP no es presionado, inicia automáticamente en el modo de ejecución, donde ejecutará el código previamente instalado. Por lo contrario, cuando se alimenta a tarjeta y a su vez se mantiene presionado el botón ISP, se entrará en el modo boot loader, esto quiere decir que la tarjeta esta lista para ser programada con un nuevo firmware.

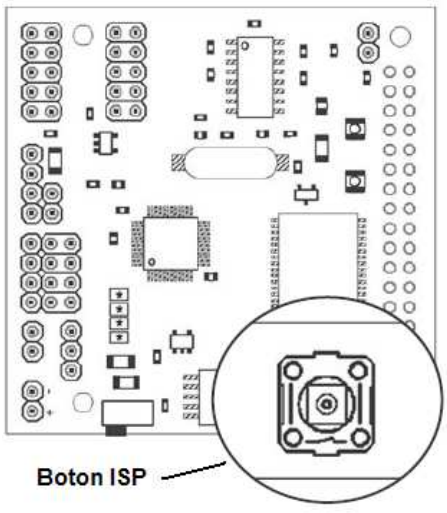


Figura 3.7 Botón ISP [11]

3.3.2 Software de la CMUcam3

Para poder iniciar a escribir código para nuestra CMUcam3 y poder crear la imagen para después descargarla en la tarjeta, primero se requiere tener instalado el IDE cc3, en cuyo directorio guardaremos nuestro código fuente, previo a ser compilado. Para compilar se tiene que tener instalado el compilador ARM compiler. Para el caso de Windows se requerirá tener un emulador de Unix como Cygwin [12]. Cygwin es un emulador de sistema Unix, el cual conjunta una gran gama de herramientas cuyo objetivo es el de soportar software que se ejecuta en sistemas POSIX y poder utilizarlo en Windows mediante una recopilación de sus fuentes. Mediante Cygwin es posible ejecutar comandos que solo son reconocidos en sistemas Unix. Para nuestro objetivo se utiliza Cygwin, para poder emplear el compilador de ARM compiler y utilizar el comando make. Make es una herramienta que se emplea para controlar los procesos de construcción de software en sistemas Unix. Mediante el empleo de reglas de compilación que se encuentran descritas en un archivo llamado "makefile". Para un estudio detallado de esta herramienta se puede consultar la referencia número 13 de este trabajo.

Si se desea descargar un programa en la CMUcam3, se requiere la utilizar Philips LPC2000 Downloader para Windows y LPC21ISP para Linux, además de la comunicación serie con el ordenador.

Para un estudio detallado de la instalación del software de la CMUcam3 véase la referencia número doce de este trabajo.

3.3.2.1 Instalación del software para Windows

Para poder trabajar con la CMUcam3 en Windows se necesita lo siguiente:

1. Windows XP
2. Instalar Cygwing [14]
3. Instalar el compilador GNU ARM GCC [9]
4. Instalar la utilería LPC210x FLASH Utility [2]
5. Instalar CMUcam3 Frame Grab Utility [2]
6. Descargar la carpeta cc3 [2]

3.3.2.1.1 Instalación de Cygwin

Para instalar Cygwing se realizan los siguientes pasos:

1. Descargar el instalador de Cygwing del sitio Web www.cygwin.com/setup.exe**.**
2. A continuación aparece una ventana donde nos preguntara si deseamos ejecutar el programa (ver figura 3.8a), le diremos que deseamos ejecutar el programa.

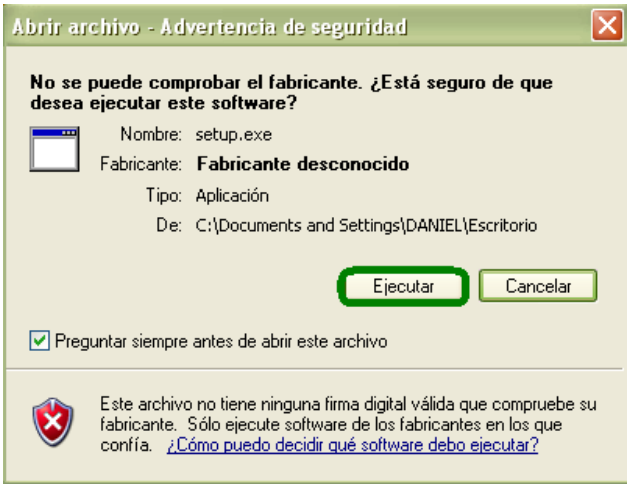


Figura 3.8a Ventana de aviso de Windows

3. Enseguida aparecerá una ventana que nos pregunta desde donde deseamos hacer la instalación, escogemos instalar desde Internet (ver figura 3.8b).

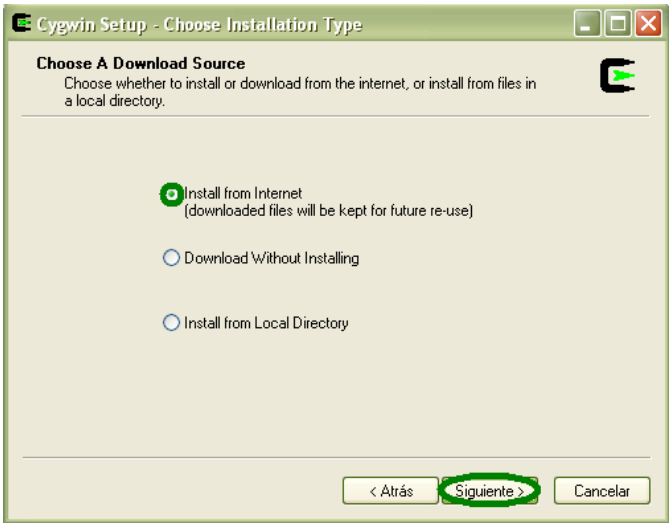


Figura 3.8b Elección de fuente de instalación

4. A continuación se elige el tipo de conexión (ver figura 3.8c)

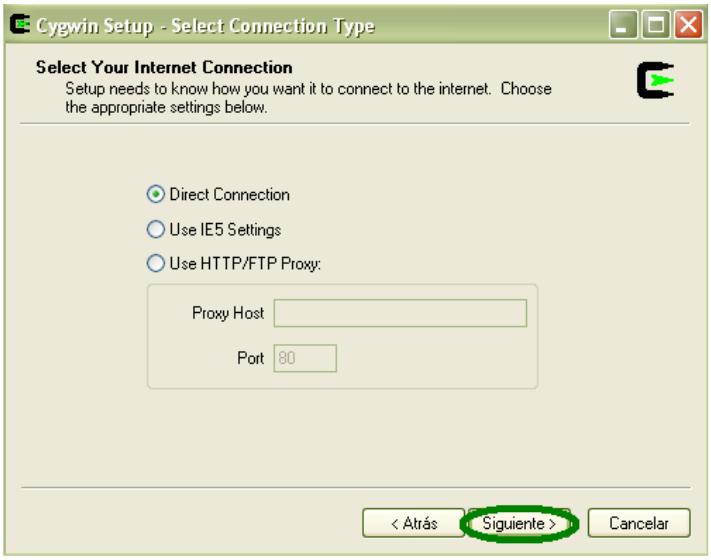


Figura 3.8c Elección del tipo de conexión

5. Después se elige el lugar donde deseamos realizar la descarga, seleccionaremos http//:mirrorskernel.org (ver figura 3.8d).

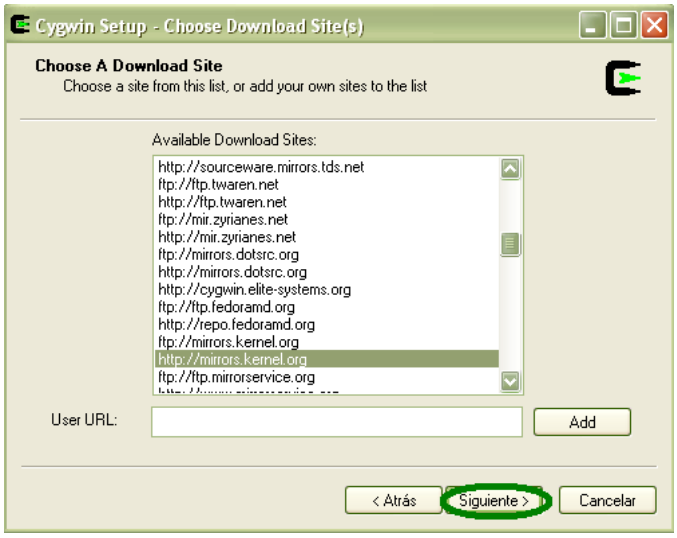


Figura 3.8d Elección del host de descarga

6. En la siguiente ventana (ver figura 3.8e) nos piden las herramientas que deseamos instalar. Primeramente maximizamos la pantalla, después seleccionamos View, a continuación seleccionamos la herramienta de make, por último seleccionamos siguiente.

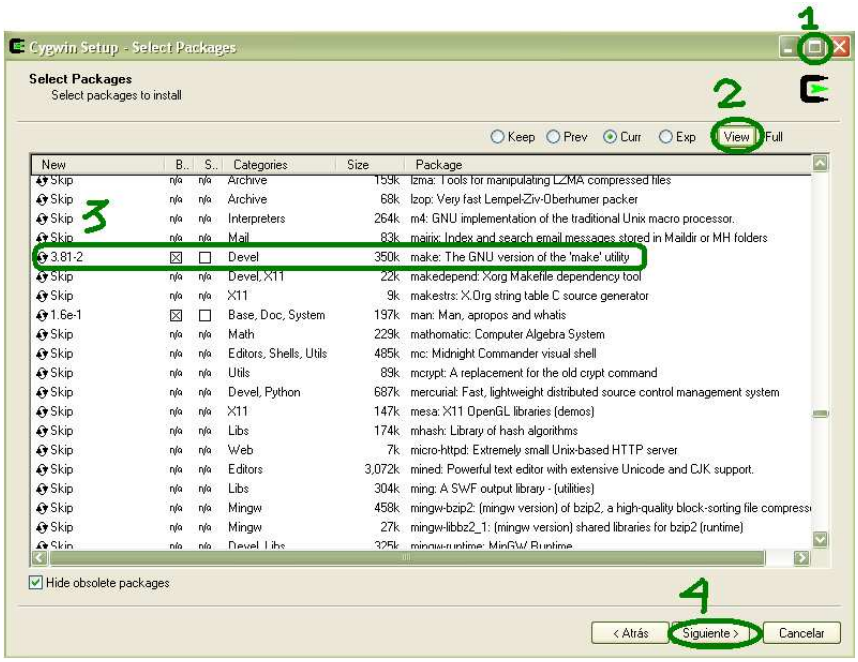


Figura 3.8e Elección de las herramientas a instalar

3.3.2.1.2 Instalación de GNU ARM GCC

El instalador se puede bajar desde la pagina Web "www.cmucam.org" o desde la página www.codesourcery.com/gnu_toolchains.

Los pasos para instalar el compilador GNU ARM GCC se mencionan a continuación:

1. Tras ejecutar el instalador, se muestra una ventana donde se comenzara a instalar el compilador (ver figura 3.13) se recomienda aceptar todas las configuraciones predefinidas.

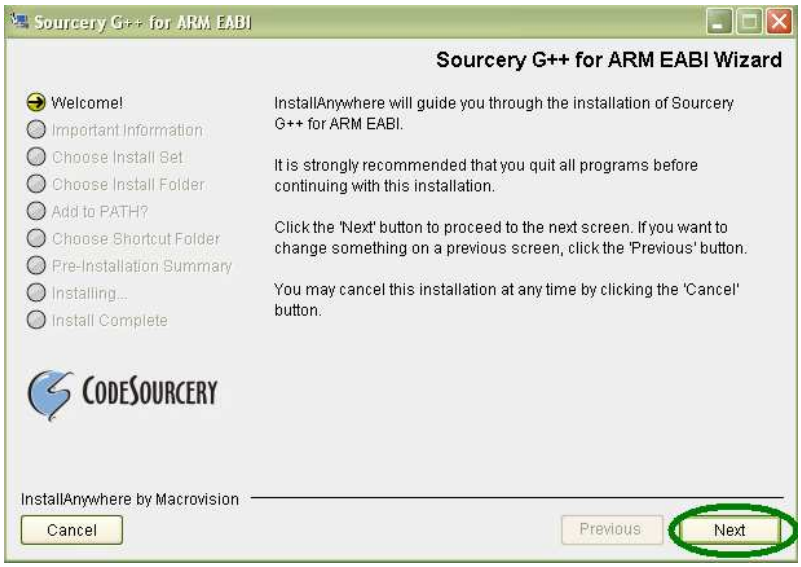


Figura 3.13 Instalación del compilador ARM G++

3.3.2.1.3 Instalación de LPC210x FLASH Utility

Para poder descargar el firmware a nuestra tarjeta es necesario utilizar el LPC210x FLASH Utility [7]. A continuación se muestran los pasos para poder utilizar esta utilidad:

1. Bajar e instalar la utilidad LPC210x FLASH, la cual puede ser descargada desde la página de la CMUcam o desde la pagina de Philips:

   http://www.semiconductors.philips.com
   o
   http://www.cmucam.org/wiki/Downloads

2. Seleccionar el icono de "philips_flash_utility.zip"
3. A continuación seleccionar el icono de Philips Flash installer.
4. Cuando aparezca una ventana (ver figura) de bienvenida, dar siguiente a todas las configuraciones predeterminadas.

Figura 3.14 Ventana de bienvenida del asistente de instalación

3.3.2.1.4 El cc3 source tree

Para poder desarrollar nuestros proyectos, requerimos el directorio cc3, al cual se le conoce como cc3 Source Tree. El directorio cc3 funciona muy parecido a un IDE (Integrated Development Environment), el cual contiene las herramientas necesarias para programar y administrar nuestros proyectos. Para que funcione correctamente el cc3 es necesario colocarlo en el directorio raíz de Windows como lo muestra la figura 3.15.

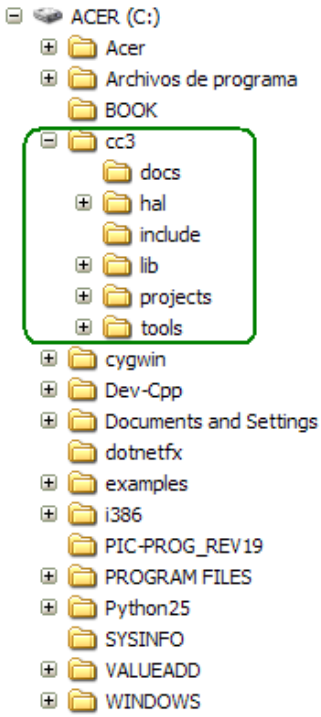


Figura 3.15 Configuración del cc3 source tree

3.3.2.2 Como crear un proyecto en cc3

Para crear un proyecto en cc3, primero se copia alguno de los proyectos que ya vienen en el cc3, nos dirigimos al directorio cc3 y abrimos el subdirectorio projects (ver figura 3.16a), a continuación copiamos el directorio hello-world y lo pegamos en la misma carpeta de projects pero le cambiamos el nombre por ejemplo “nuevo _ proyecto” (ver figura 3.16b).

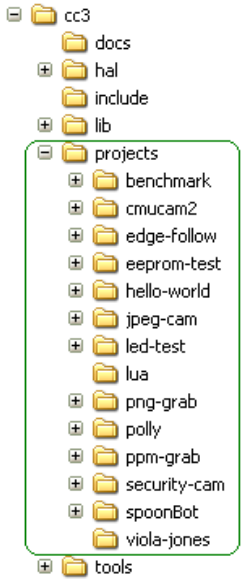


Figura 3.16a Apertura de la carpeta projects

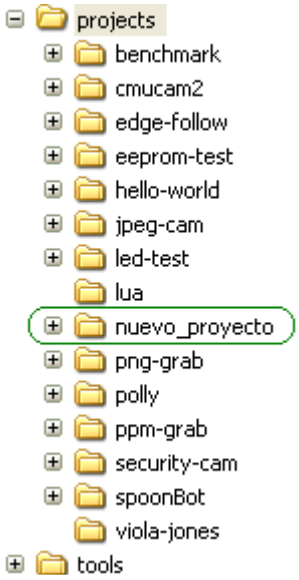


Figura 3.16b Creación de un nuevo proyecto llamado “nuevo _proyecto”

Lo que sigue es modificar el archivo llamado “makefile”, en el que modificamos el nombre del proyecto, cambiando “hello-world” por “nuevo_proyecto” (ver figura 3.16c), de esta manera hemos creado un proyecto y ahora es cuestión de abrir el archivo main.c comenzar a programar.

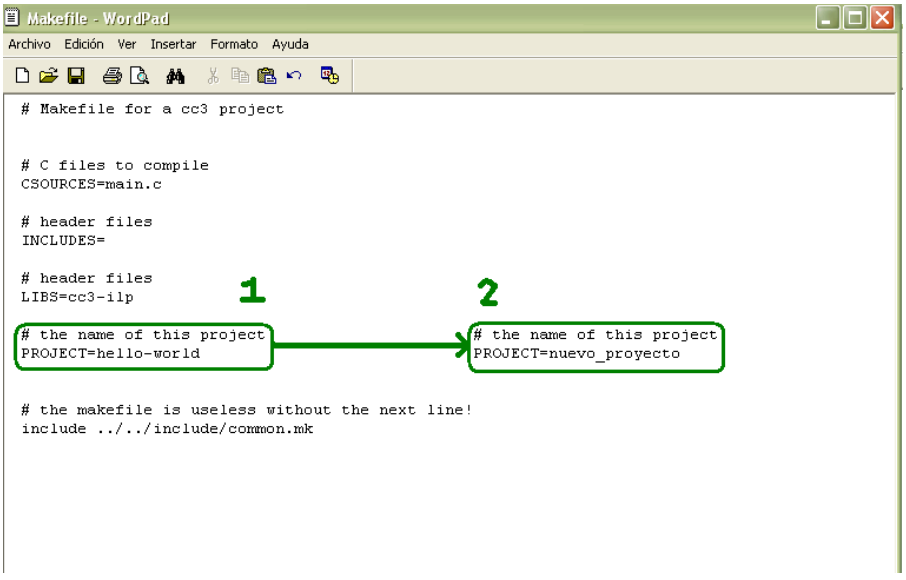


Figura 3.16c Codificación del archivo Makefile

Para compilar nuestro código fuente empleando el sistema cc3, necesitamos del comando make, un archivo incluido en nuestro proyecto llamado Makefile [13] y Cygwin. Los pasos para compilar código fuente se mencionan a continuación:

1. Iniciar Cygwin dando doble clic en el icono de Cygwin.

2. Nos dirigirnos al directorio de c: mediante el comando *cd c:*, después cambiarnos al directorio de nuevo_proyecto mediante el comando cd /cc3/projects/nuevo_proyecto (ver figura 3.17).
3. A continuación invocamos make en el directorio de nuevo_proyecto, con lo que se compilará el proyecto y nos generará el archivo .hex para la CMUcam3 (ver figura 3.17).

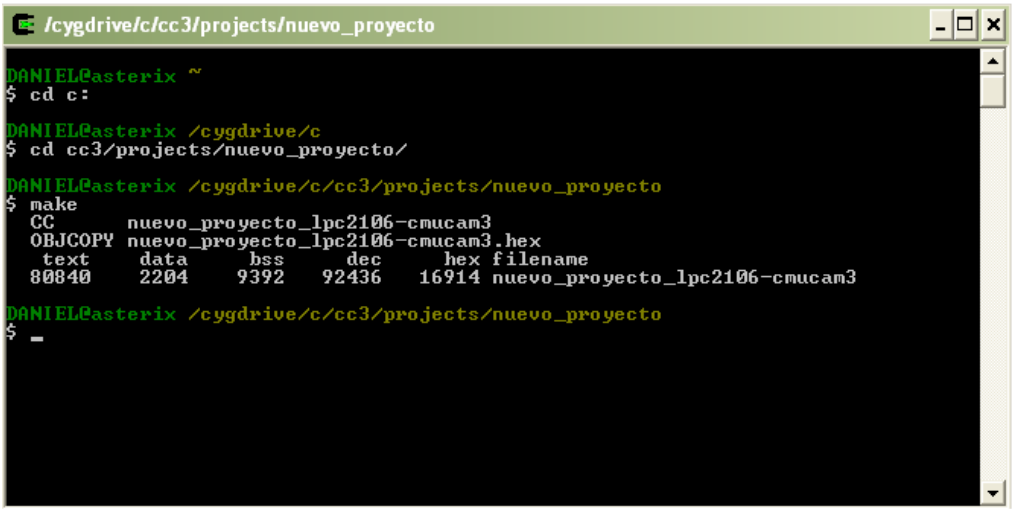


Figura 3.17 Compilación de un proyecto

3.3.2.3 Programación de la CMUcam3

Para programar a la CMUcam3 se necesita el LPC210x FLASH Ulitily, que nos permite descargar en la CMUcam3 un nuevo firmware. La figura 3.18a muestra las configuraciones que debe tener esta utilidad para poder programar a la CMUcam3. A continuación se muestran los pasos necesarios para programar a la CMUcam3.

1. Conectar la CMUcam3 con la computadora mediante el puerto serial.
2. Ejecutar el LPC210x FLASH Utility.
3. Seleccionar el dispositivo LPC2106 y la frecuencia del cristal a 14745[Khz.].
4. Seleccionar el puerto por el que se comunicará la computadora y establecer la velocidad de transmisión a 115200 baud.
5. Seleccionar la imagen del firmware que se desea descargar en la tarjeta.
6. Dar doble clic en Upload Flash.
7. Aparecerá una ventana que nos pedirá reiniciar la CMUcam3 (ver figura 3.18b).
8. Presionar el botón ISP de la CMUcam3 y mantenerlo así.
9. Encender la CMUcam3 mientras se mantiene presionado el botón ISP.
10. Después seleccionamos aceptar (ver imagen 3.18b) y la imagen comenzará a descargarse.

Figura 3.18a Configuración del LPC2000 Flash Utility

Figura 3.18b Mensaje para reiniciar la CMUcam3

3.4 Programas de prueba

En esta sección se muestran los programas básicos que son necesarios para poder implementar un sistema de visión para detectar colores.

3.4.1 Programa para recibir y enviar datos mediante el puerto serial de la CMUcam3

Para comunicarnos con la computadora, el IDE cc3 contiene una librería llamada "stdio.h" la cual nos permite utilizar las funciones printf(), getchar(), putchar(), gets(), entre otras, que ya vienen configuradas para reconocer como su salida estándar el puerto serial.

El siguiente programa esta escrito en C, realiza la comunicación con la computadora. El programa imprime un mensaje de bienvenida y pide al usuario que se comunica mediante una hyperterminal, ingresar una cadena, enseguida la CMUcam3 regresa la cadena como la escribió el usuario (ver programa 1 del apéndice B).

3.4.2 Programa que detecta el color rojo

El siguiente programa nos permite detectar un color rojo intenso, que para aplicaciones simples nos valemos de las API de la CMUcam, que ya tienen llamadas al sistema que

nos facilita la programación y detección de colores. A continuación se presenta el programa de detección de color rojo intenso, el programa obtiene el centroide de una región de color rojo intenso, su envolvente y el número de pixeles que encontró con ese tono de color, para después enviarlos por la hyperterminal. (Ver programa 2 del apéndice B).

3.4.3 Programa CMUcam2 y la utilidad CMUcam3 Frame Grabber

Cuando existen cambios de luz en el ambiente es necesario volver a modificar los valores de los límites del color que deseamos detectar, una manera es mediante el empleo de la utilidad CMUcam3 Frame Grabber (ver figura 3.19). Para poder utilizar esta modalidad es necesario que la CMUcam3 tenga previamente cargado el firmware de la CMUcam2, que se encuentra en la carpeta de proyectos del cc3 con el nombre de "cmucam2_lpc2106-cmucam3.hex". Una vez cargado el firmware abriremos el programa CMUcam3 Frame Grabber y aparecerá una ventana similar al de la figura 3.19, donde configuraremos los parámetros de velocidad de transmisión a 115200 baud, elegiremos el puerto serial deseado, pondremos el Timeout a 5000[ms] y por último nos conectaremos por medio de la CMUcam3 Frame Grabber, que en realidad es una hyperterminal.

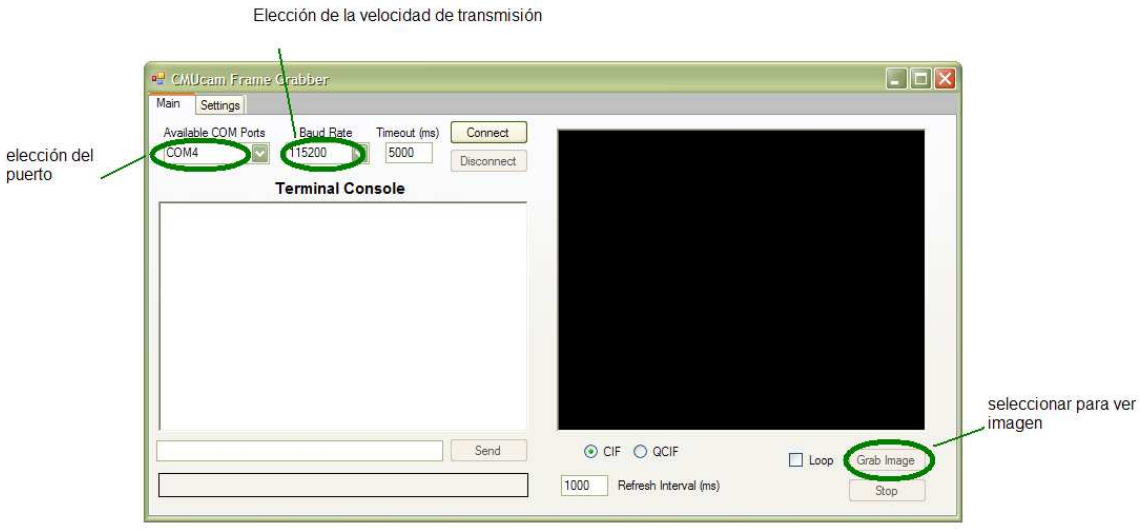


Figura 3.19 CMUcam3 Frame Grabber

Una vez hecho lo anterior, presionaremos en Grab image, y comenzaremos a ver lo que la cámara digital detecte y en ese momento la CMUcam3 estará dispuesta a recibir comandos, los cuales nos ayudarán a encontrar los valores máximo y mínimo del color que deseamos detectar. Algunos de los comandos importantes se mencionan a continuación:

GET_MEAN (GM): nos da el valor promedio del color que se ve en la pantalla.

GET_TRACK (GT): permite guardar los nuevos parámetros del color que se desea seguir, los valores son de 0 a 255 y en el orden RGB.

TRACK_COLOR (TC): una vez establecidos los nuevos parámetros, y tras ejecutar este comando, en la pantalla de la terminal aparecerá el centroide del color que queremos detectar, en caso de no ver el color deseado mandará 0.
Existe una exhaustiva lista de comandos, en este trabajo solo se presentan los comandos utilizados para realizar pruebas. Un estudio detallado de esta lista se puede ver en código fuente del proyecto CMUcam2 o visitando la pagina Web www.cmucam.org.

CAPÍTULO IV DESARROLLO

4.1 Introducción

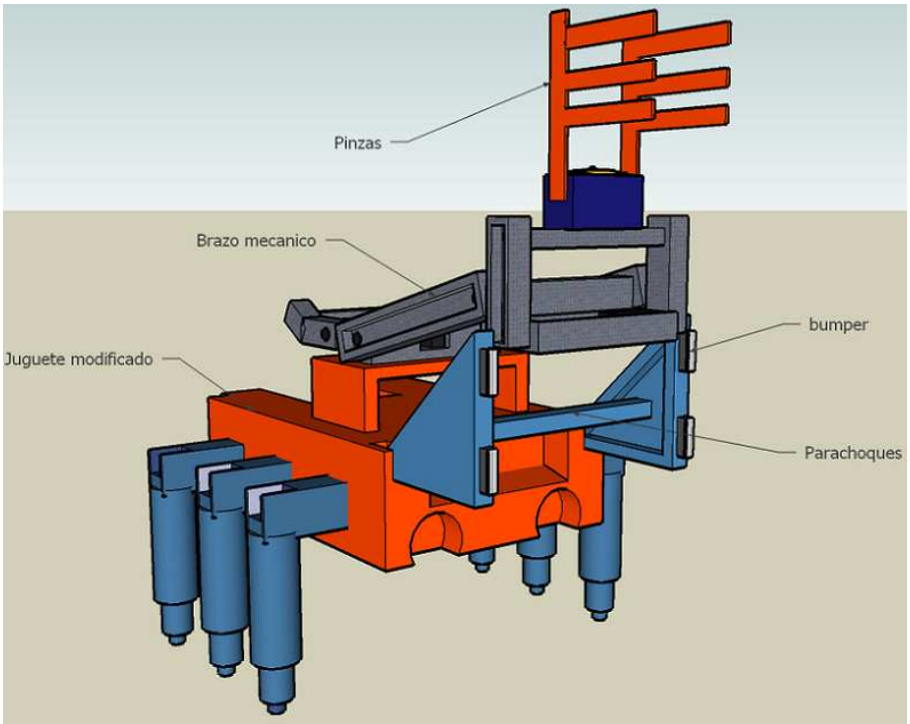
Este capítulo describe en forma detallada la construcción del robot limpiador de playa.

El sistema se divide en dos partes:

a) Estructura mecánica- el cual consta del robot comercial modificado y el brazo mecánico.
b) Sistema eléctrico- el cual consta de la tarjeta de potencia, la tarjeta controladora, la CMUcam3 y sensores de tacto.

4.2 Estructura mecánica

Está conformada por la araña comercial, el brazo mecánico y un parachoques. La araña se modificó para que la podamos controlar con un microcontrolador, únicamente montándole nuestras tarjetas de potencia y así aprovechar de esta manera su sistema motriz que consta de dos sistemas de engranes y un motor por cada sistema. A la estructura de la araña se le agregó un brazo mecánico con el que pueda recolectar los objetos. El brazo esta hecho de aluminio, ya que es ligero y durable. El brazo está situado en la parte de arriba de la araña (ver figura 4.1). En la parte frontal de la araña se le colocó un parachoques que contiene un arreglo de cuatro bumper's, también hecho con aluminio. La figuras 4.1a y 4.1b muestran la estructura ya montada.



Figuras 4.1a y 4.1b Estructura mecánica del robot

4.3 Sistema eléctrico

Para el sistema eléctrico del robot se desarrollaron tres tarjetas, la primera es la tarjeta controladora que está implementada con un microcontrolador PIC16F877A, la segunda es una tarjeta de potencia, que consiste de un puente H construido con relevadores cuyo fin es el de controlar el movimiento del robot, la tercera tarjeta es una tarjeta de potencia, que consiste de un circuito L293D cuyo fin es el de controlar las pinzas del robot. Los siguientes puntos detallan cada uno de los elementos eléctricos del sistema.

4.3.1Tarjeta controladora con PIC16F877A

Para poder controlar el robot, se desarrolló una tarjeta controladora basada en un PIC16F877A [5], esta se encargará de controlar todos los actuadores con los que cuenta el robot. Para el diseño de la tarjeta se utilizó el software Eagle [15] el cual es una herramienta para poder crear plantillas para circuitos impresos. La figura 4.1c muestra el diagrama esquemático de la tarjeta. La tarjeta esta compuesta por los siguientes elementos:

a) Un microcontrolador PIC16F877A.
b) Un circuito integrado ULN2003 para manejar los relevadores.
c) Un circuito integrado MAX232 para cambiar a los niveles RS232.
d) 2 diodos 4N4004.
e) 1 resistencia de 10[kΩ] y 1 resistencia de 330[Ω].
f) 5 capacitores de 1[μF], 1 capacitor de 10 [μF], 2 capacitores de 22[pF] y 1 capacitor de 0.1 [μF].
g) 1 conector mollex de tres vías.
h) 1 conector de alimentación.

La figura 4.1d se muestra el PCB de la tarjeta terminada en Eagle [15]. La interfaz de la tarjeta esta compuesta principalmente por cuatro puertos, mediante los cuales se proporciona las señales de control a los actuadores del robot. La figura 4.1e muestra la tarjeta terminada.

Los pines que utilizaron del PIC16F877A se muestran en la figura 4.1f, donde se menciona el numero de pin y su función en el sistema.

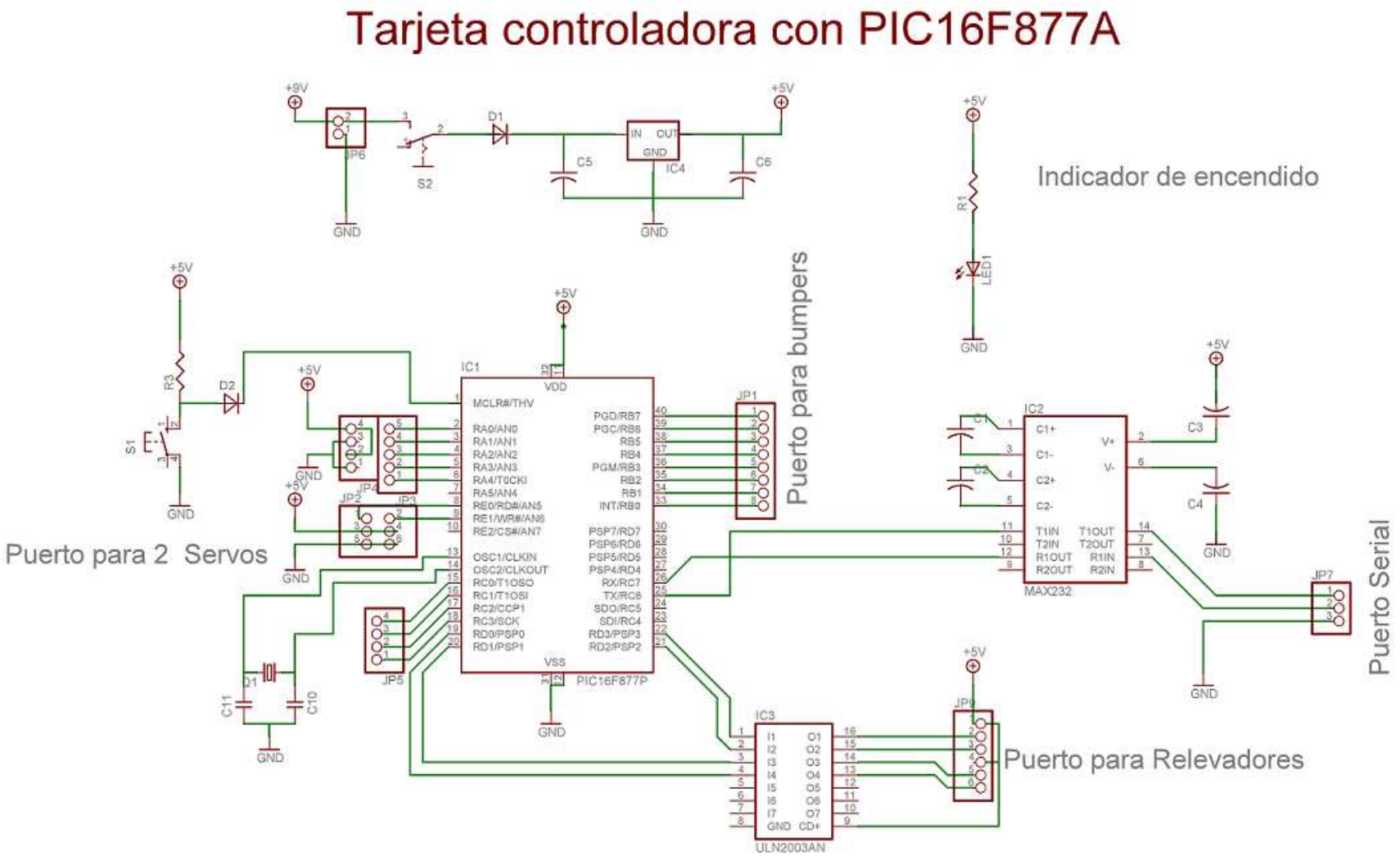


Figura 4.1c Tarjeta controladora

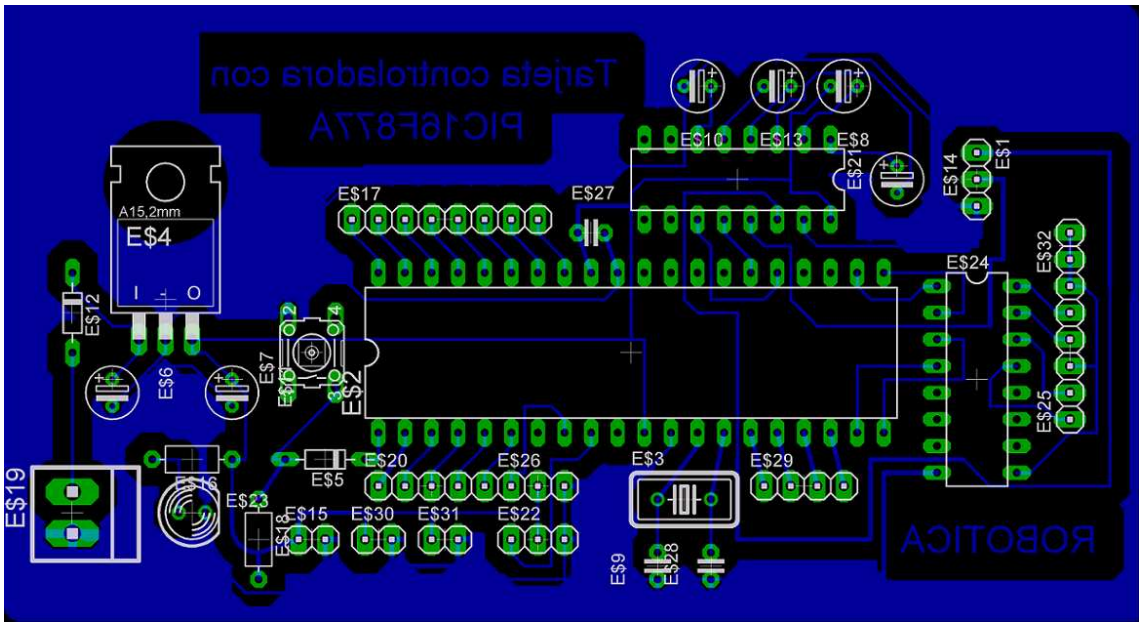


Figura 4.1d PCB de la tarjeta controladora

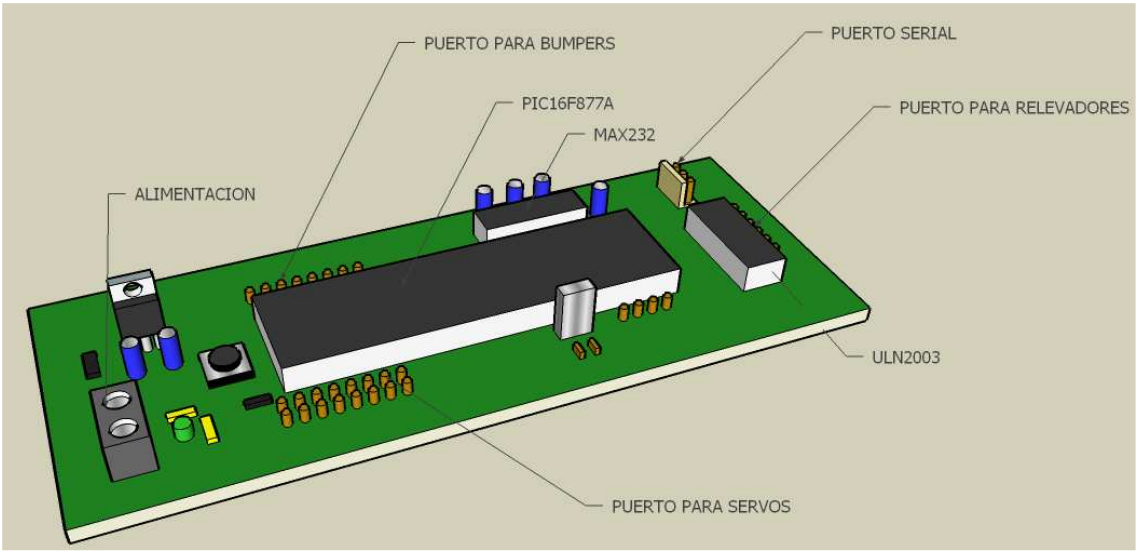


Figura 4.1e Puertos de la tarjeta controladora

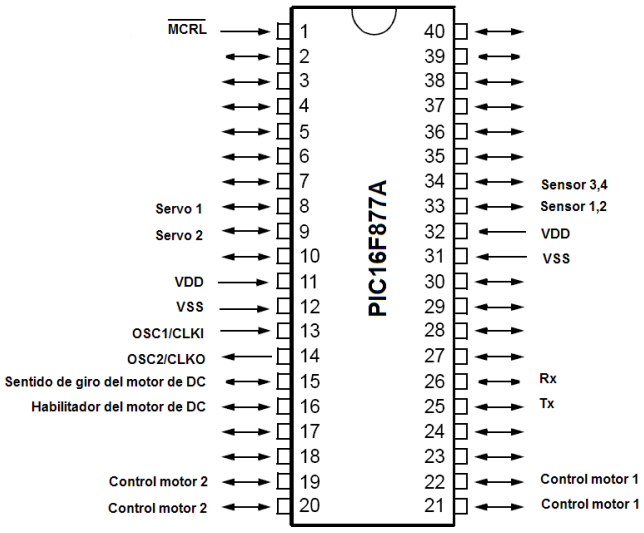


Figura 4.1f Diagrama de los pines utilizados en el robot

4.3.2 Programación de la tarjeta controladora

Para programar la tarjeta, se necesito lo siguiente:

1. Una computadora tipo laptop
2. Un programador de PIC's
3. Un cable USB
4. Un cable convertidor usb-serial

Primeramente se necesito descargar en el PIC16F877A el firmware para poder programarlo mediante el puerto serial [16], utilizando un programador universal de PIC's [1]. Una vez cargado el firmware en la tarjeta, se monto el PIC en la tarjeta controladora y a partir de este momento se puede programar la tarjeta controladora mediante el puerto serial utilizando el programa PIC downloader [16]. La figura 4.2 muestra la conexión típica entre la computadora y la tarjeta controladora mientras es programada.

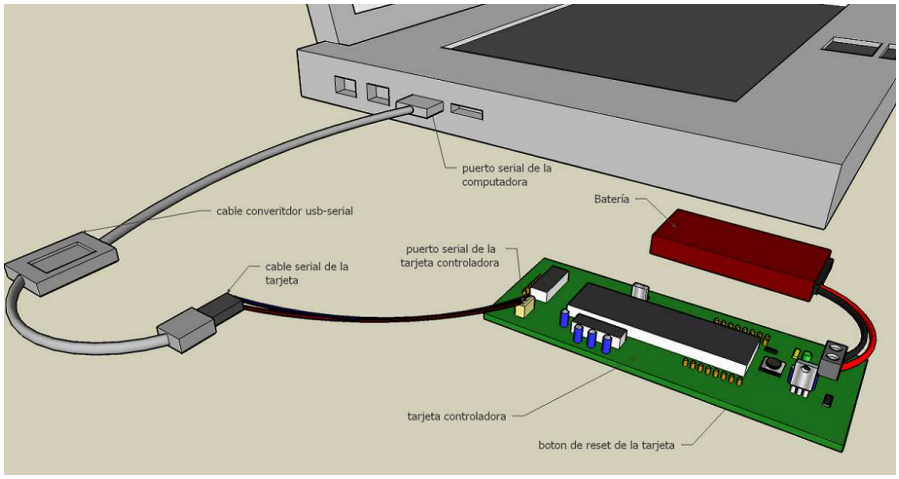


Figura 4.2 Programación de la Tarjeta

4.3.3 Tarjetas de potencia

Para poder mover al robot se requirieron construir dos tarjetas de potencia basadas en un puente H construido con relevadores, la figura 4.3 muestra el esquema del puente H diseñado en Eagle [15].

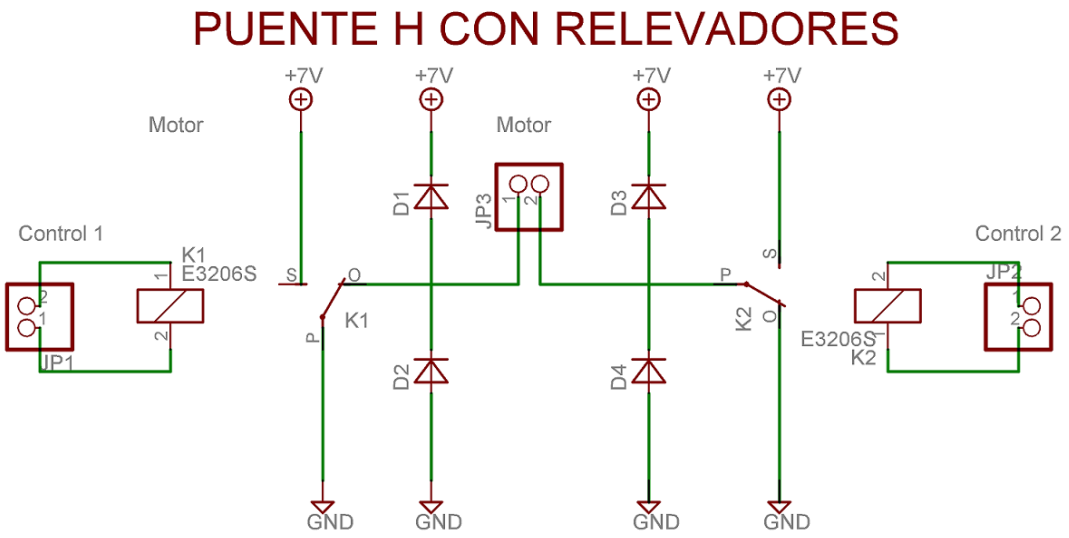


Figura 4.3a Diagrama la tarjeta de potencia para un motor

Las figuras 4.3b y 4.3c muestran la tarjeta de potencia en PCB y la tarjeta terminada respectivamente.

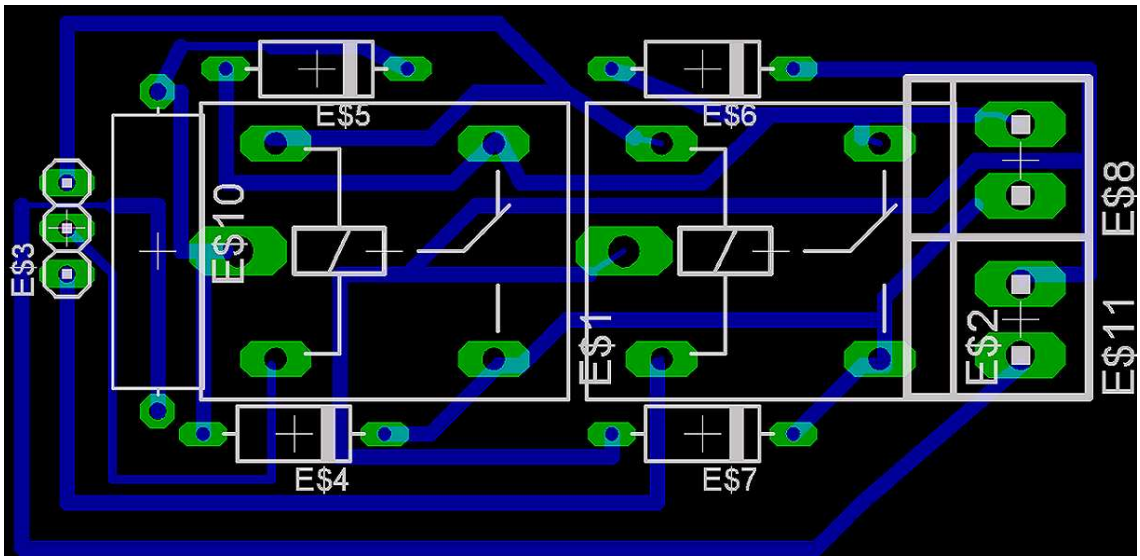


Figura 4.3b PCB de la tarjeta de potencia

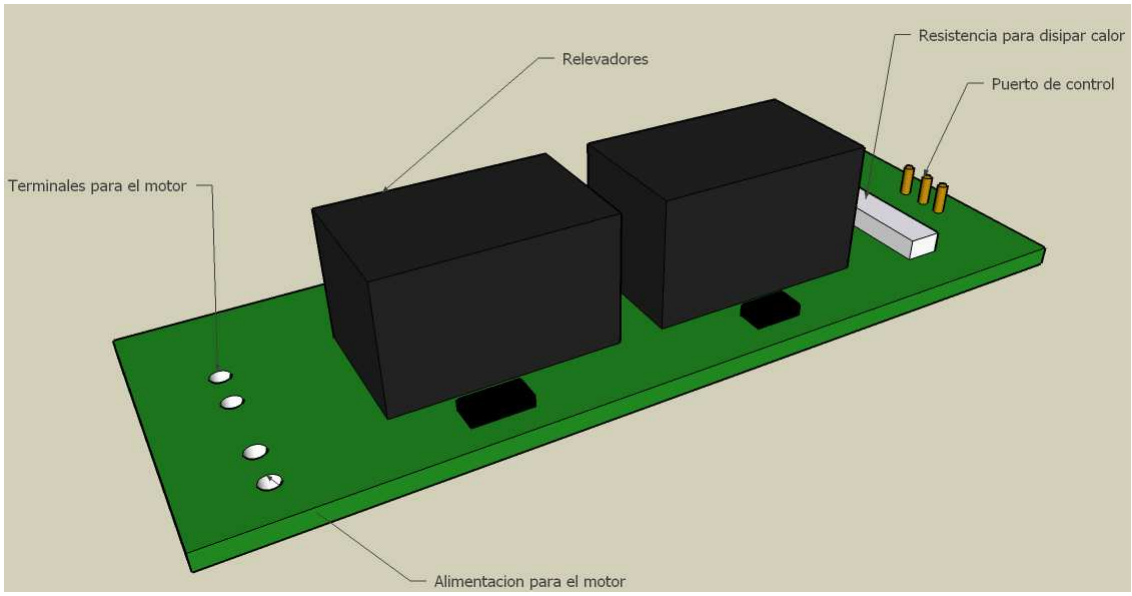


Figura 4.3c Tarjeta de potencia terminada

La segunda tarjeta de potencia que se hizo fue para el motor que controla las pinzas del robot. El esquema en Eagle [15] se muestra en la figura 4.4a.

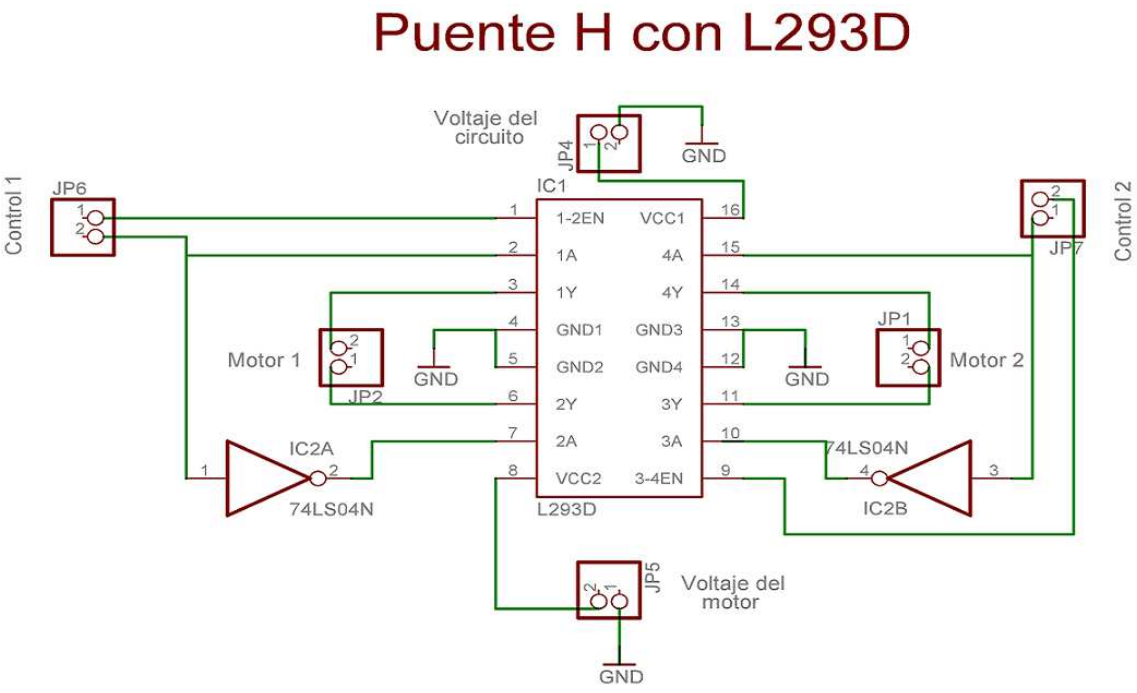


Figura 4.4a Diagrama de la tarjeta de potencia con circuito L293D

Las figuras 4.4b y 4.4c muestran el PCB de la tarjeta y la tarjeta terminada respectivamente.

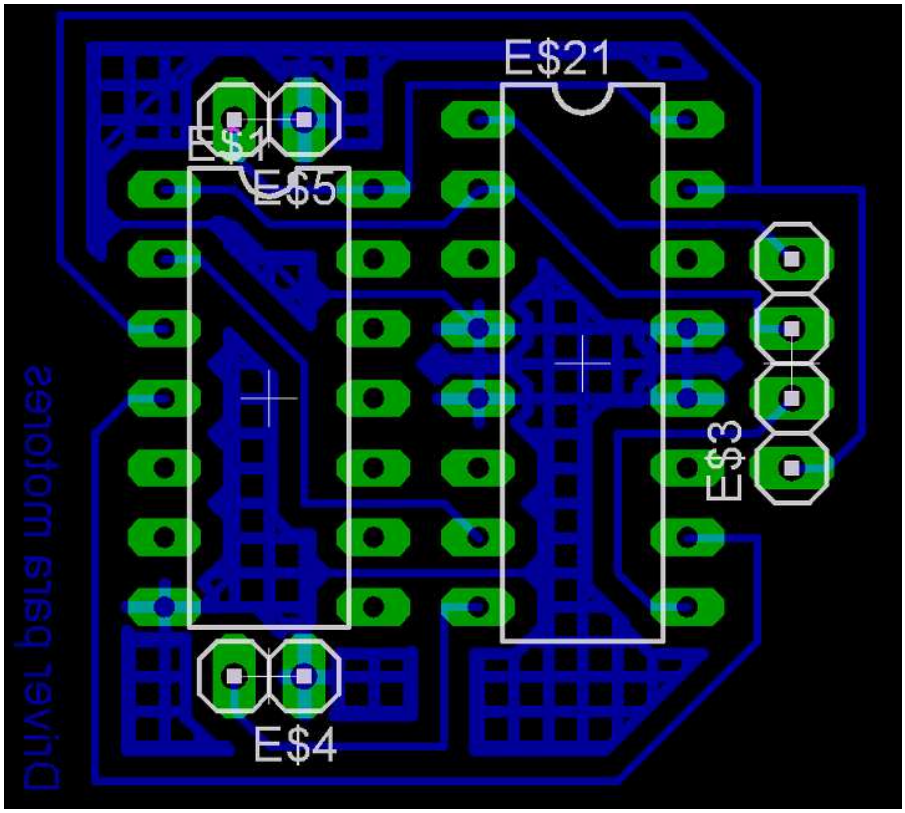


Figura 4.4b PCB de la tarjeta de potencia

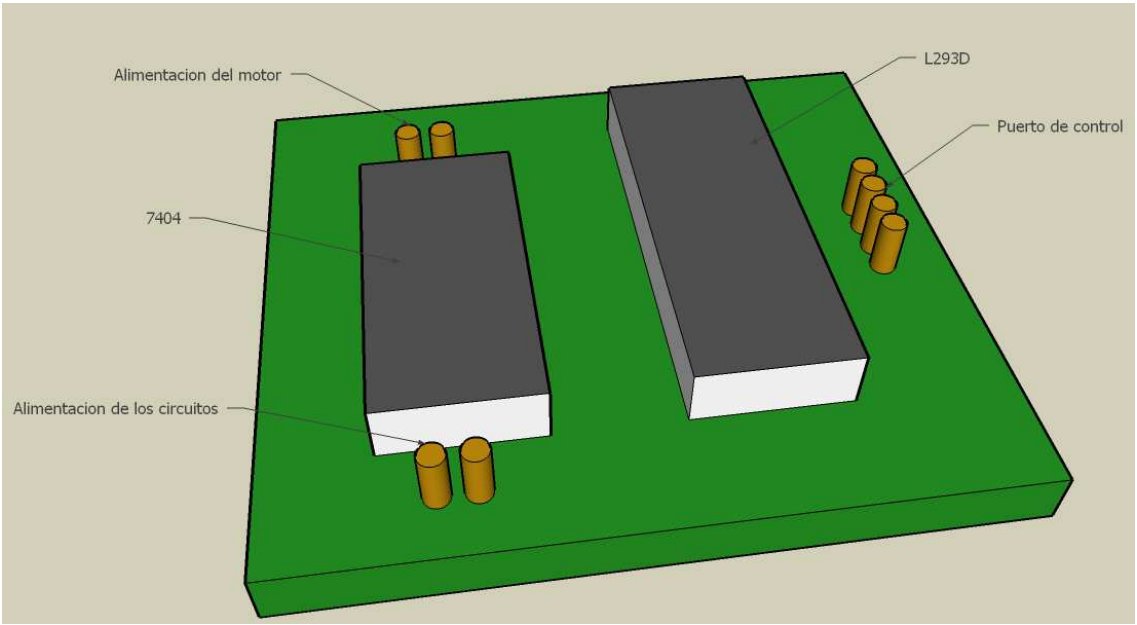


Figura 4.4c Tarjeta de potencia terminada

4.3.4 Montaje y conexión de la tarjeta controladora al sistema

La figura 4.1c muestra los puertos con los que cuenta la tarjeta, en la figura 4.5a se muestra la conexión de la tarjeta con la etapa de potencia, que consta de la tarjeta de

potencia que controla los relevadores y la tarjeta de potencia que controla las pinzas del robot.

Para alimentar a los servomotores y el motor de DC de la pinza del robot, se utilizó una batería de 6 volts, se construyó una interfaz de conexiones, donde se conecta la batería de 6 volts a los dos servos y al motor de DC. La figura 4.5b muestra la interfaz y como conectar los servomotores y el motor de DC de la pinza del robot a la tarjeta.

La figura 4.5c muestra la conexión de los bumper's al sistema, donde no se requirió de utilizar resistencias de pull-up externas ya que se utilizo las resistencias de pull-up internas del PIC16F877A.

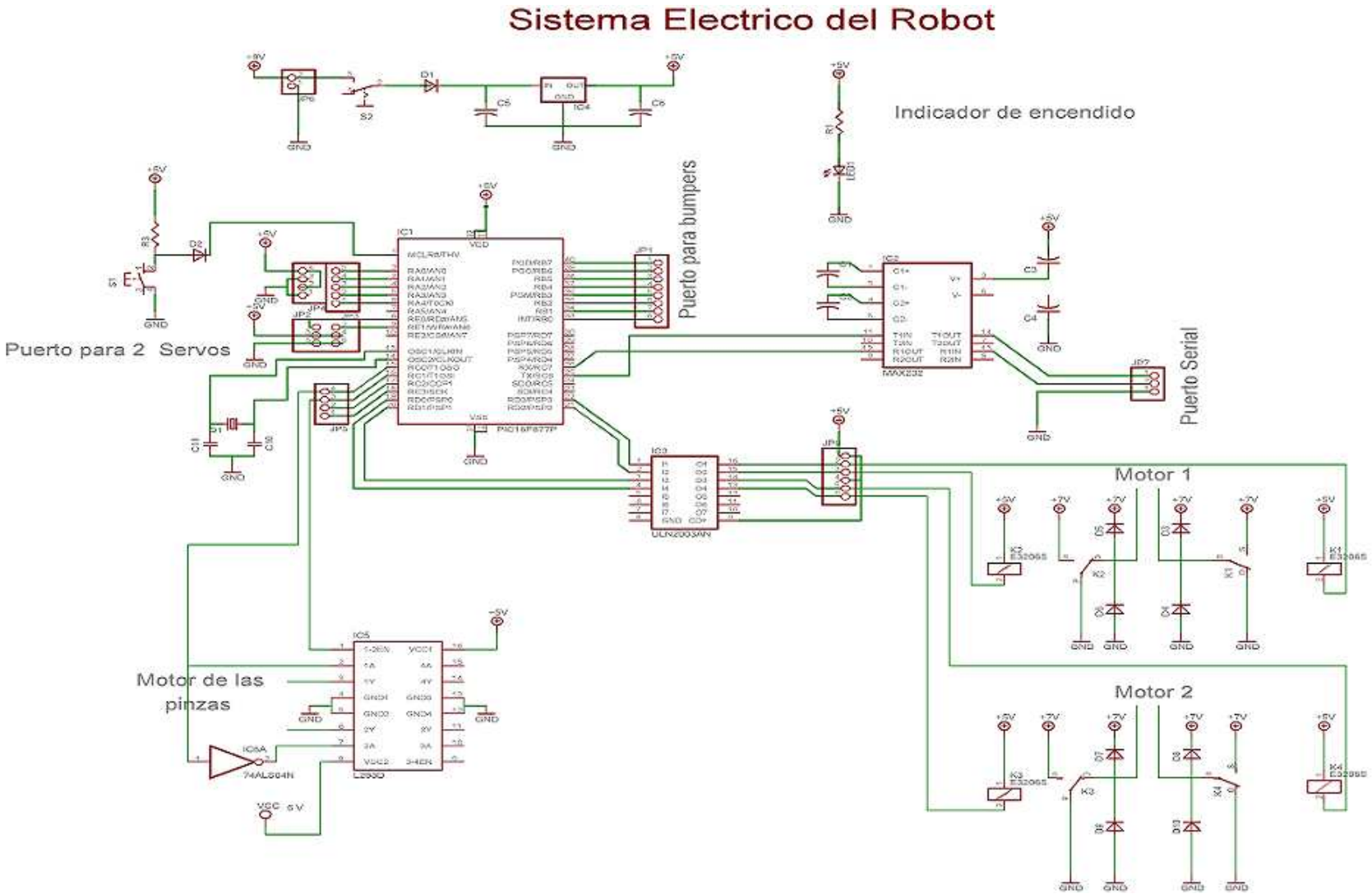


Figura 4.5a Conexión de la etapa de potencia a la tarjeta

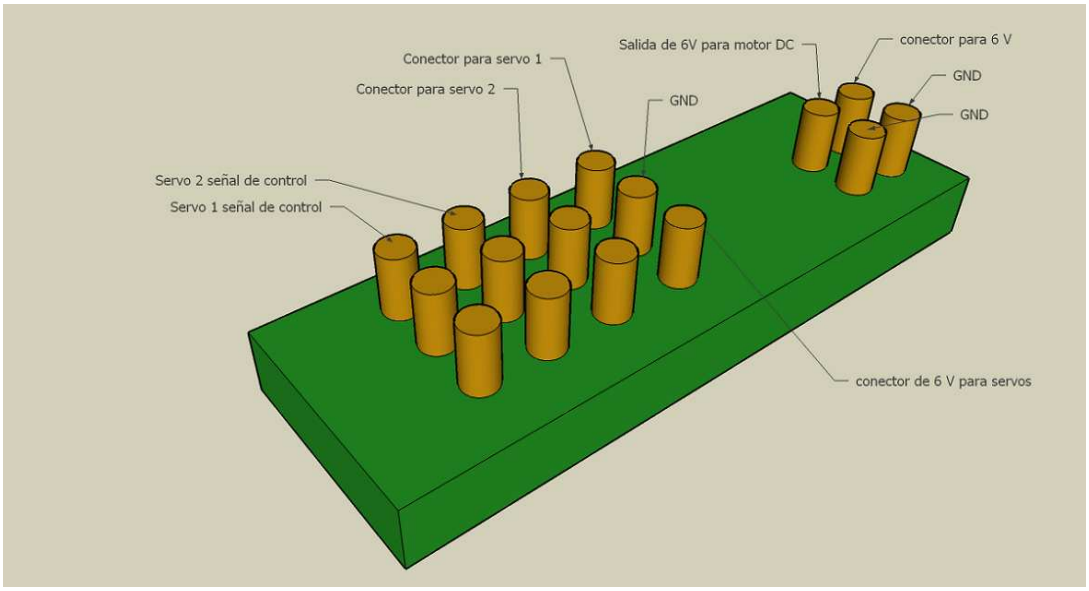


Figura 4.5b Interfaz para alimentar a los servos y el motor de la pinza

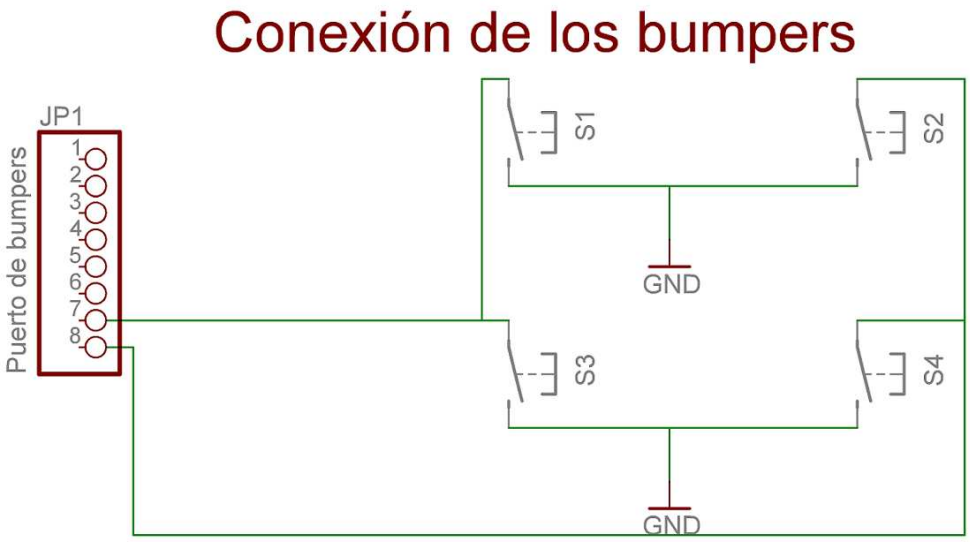


Figura 4.5c Conexión del arreglo de Bumpers a la tarjeta.

4.3.5 Control de los actuadores

Una vez que se monto el sistema, se prosiguió a mover al robot mediante comandos que se envían por la hyperterminal. El programa que controla los movimientos del robot se presentan en el apéndice A con el nombre de programa 3. El programa necesita de los siguientes encabezados:

```
#include <16f877A.h>
#device ADC=8
#include <stdlib.h>
```

```
#include <stdio.h>
#fuses HS,NOPROTECT
#use delay(clock=20000000)
#use rs232(baud=19200, xmit=PIN_C6, rcv=PIN_C7)
#org 0x1FFF, 0x1FFF void loader16f877A(void){}
```

Y las siguientes definiciones de macros:

```
#define ADELANTE   0x09
#define ATRAS      0x06
#define IZQUIERDA  0x05
#define DERECHA    0x0A
#define APAGADO    0x00
```

El programa espera un caracter dado por el usuario y de acuerdo al ese caracter el robot se moverá en la posición especificada por la siguiente tabla:

Carácter	Movimiento
‘1’	Adelante
‘2’	Atrás
‘3’	Izquierda
‘4’	Derecha
‘5’	Parado
‘6’	Finalizar programa

La salida se manda por el puerto D del microcontrolador, el cual solo utiliza los pines D3-D0.

Para mover el brazo mecánico se necesitaron de cuatro señales de control, dos de ellas para mover los servos y las otras dos para mover el motor de DC. Los pines que se utilizaron se muestran a continuación:

Pin	
15-C0	Indica el sentido de giro del motor 1=abrir, 0=cerrar
16-C1	Habilita el motor
8-E0	Servo motor 1
9-E1	Servo motor 2

El programa que se encuentra en el apéndice A con el nombre de programa 4, muestra el control del brazo mecánico. Para una mejor portabilidad, el programa esta dividido en dos archivos fuente, uno con el nombre de brazomecanico.c que contiene las funciones y variables que se requieren para controlar el brazo mecánico y el otro tiene el nombre de main.c que contiene la ejecución del programa. Las librerías que utiliza este programa son:

```
#include <16f877A.h>
#device ADC=8
#include <stdlib.h>
#include <stdio.h>
```

```
#fuses HS,NOPROTECT
#use delay(clock=20000000)
#use rs232(baud=19200, xmit=PIN_C6, rcv=PIN_C7)
#org 0x1FFF, 0x1FFF void loader16f877A(void){}
```

Y el archivo que se incluye es:

```
#include <brazomecanico.c>
```

El programa hace una demostración del movimiento del brazo mecánico, rutina que se anexa en el movimiento que tiene que hacer el robot cuando levanta un objeto.

4.4 El sistema embebido de visión CMUcam3

La CMUcam3 se colocó dentro de una caja negra para protegerla de golpes (ver figura 4.6a) y fue colocada en la parte frontal del robot. La figura 4.6b muestra al robot con la CMUcam3 montada.

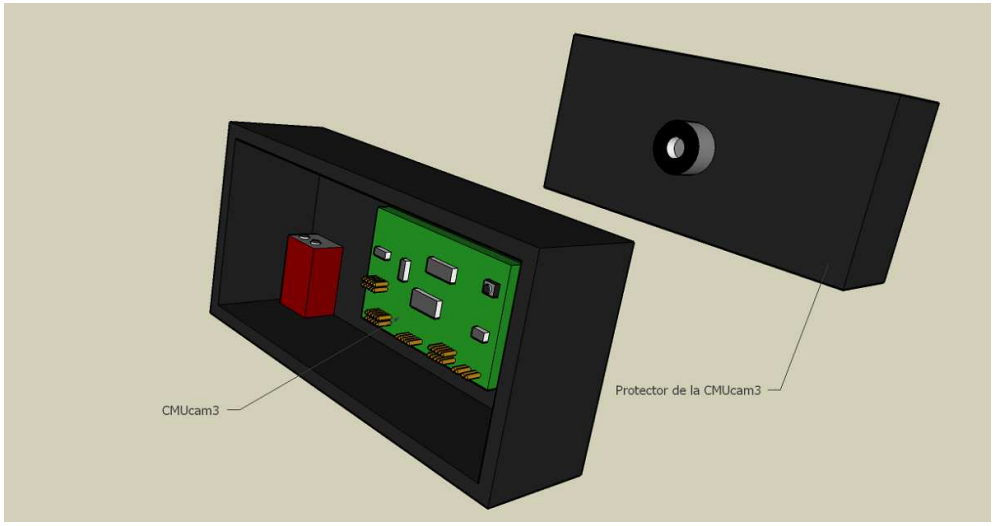


Figura 4.6a La CMUcam3 con su protección

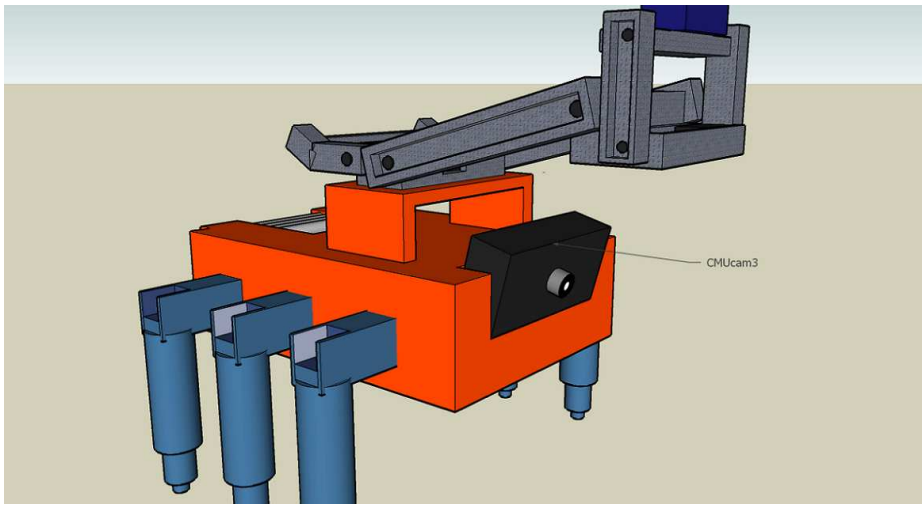


Figura 4.5b Ubicación de la CMUcam3 en el robot

4.4.1 Programación de la CMUcam3

Para programar la CMUcam3 se requiere lo siguiente:

1. Cable convertidor USB-Serial
2. Cable serial para la CMUcam3
3. Una batería de 9V
4. El programa LPC 2000 Flash Utility

La figura 4.5c muestra la conexión típica para programar la CMUcam3.

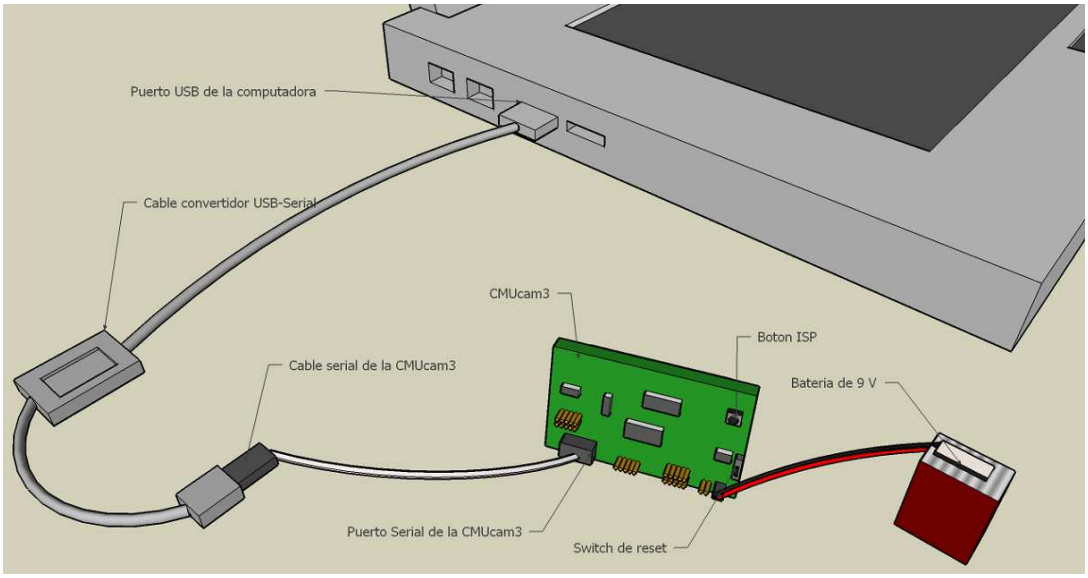


Figura 4.5c conexión de la CMUcam3 mientras es programada

4.4.2 Detección de los colores rojo, verde y azul

Para que el robot detecte los residuos de color rojo, verde y azul, se programa a la CMUcam3 para encontrar el centroide de la figura que tenga estos colores para después ubicar las coordenadas del centroide dentro de un mapa de nueve cuadrantes como se muestra a continuación:

1	2	3
4	5	6
7	8	9

Cuando no detecta nada, manda el centroide en la coordenada (0,0) que corresponde al lado superior izquierdo. Los límites se detallan en el apéndice B programa3, donde si el

centroide cae dentro de algún cuadrante, se manda un carácter por el puerto serial; en la tabla siguiente se pone el carácter que se envía de acuerdo al cuadrante en que se encuentre el centroide.

Cuadrante	Caracter que se envía
1	‘1’
2	‘2’
3	‘3’
4	‘1’
5	‘4’
6	‘3’
7	‘4’
8	‘4’
9	‘4’

El significado de cada carácter que se envía es el siguiente:

Caracter	Significado
‘1’	Girar a la derecha
‘2’	Avanzar
‘3’	Girar a la izquierda
‘4’	Detenerse

Las librerías que se utilizaron para el programa3 del apéndice B son las siguientes:

```
#include <stdio.h>
#include <stdlib.h>
#include <cc3.h>
#include <cc3_ilp.h>
#include <cc3_color_track.h>
```

Y los macros que se utilizaron son los siguientes:

```
#define  SuperiorIzquierdo    putchar('1');
#define  Superior             putchar('2');
#define  SuperiorDerecho      putchar('3');
 #define  CentroIzquierdo     putchar('1');
#define  Centro               putchar('4');
#define  CentroDerecho        putchar('3');
#define  InferiorIzquierdo    putchar('4');
#define  Inferior             putchar('4');
#define  InferiorDerecho      putchar('4');
#define  Negro                putchar('5');
```

4.4.3 Calibración de los colores mediante la utilidad CMUcam Frame Grabber

Para el robot vea el tono de rojo, verde y azul que nosotros queremos, se necesitan hacer pruebas mediante el uso de la utilidad CMUcam frame grabber. En el capítulo 5 se explica esta técnica.

4.4.4 Montaje y conexión de la CMUcam3 al sistema

La CMUcam3 se conecta a la tarjeta controladora mediante sus puertos seriales, la figura 4.6a muestra la conexión correcta de la CMUcam3, la figura 4.6b muestra como queda esta conexión y la figura 4.6c muestra como queda el sistema montado. La comunicación entre la CMUcam3 y la tarjeta controladora se realiza mediante el puerto serial, a la cual le mandara el número de instrucción que deberá ejecutar la tarjeta controladora.

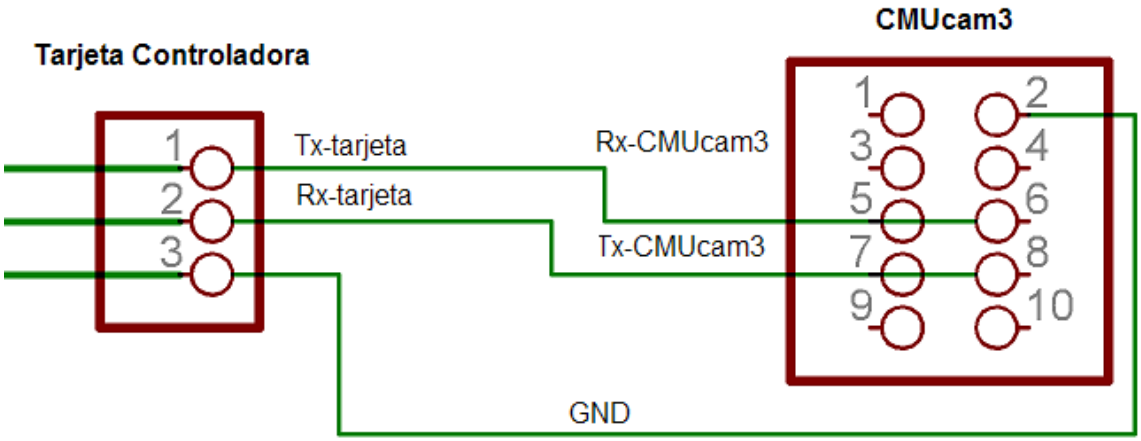


Figura 4.6a Conexión de la CMUcam3 a la tarjeta controladora

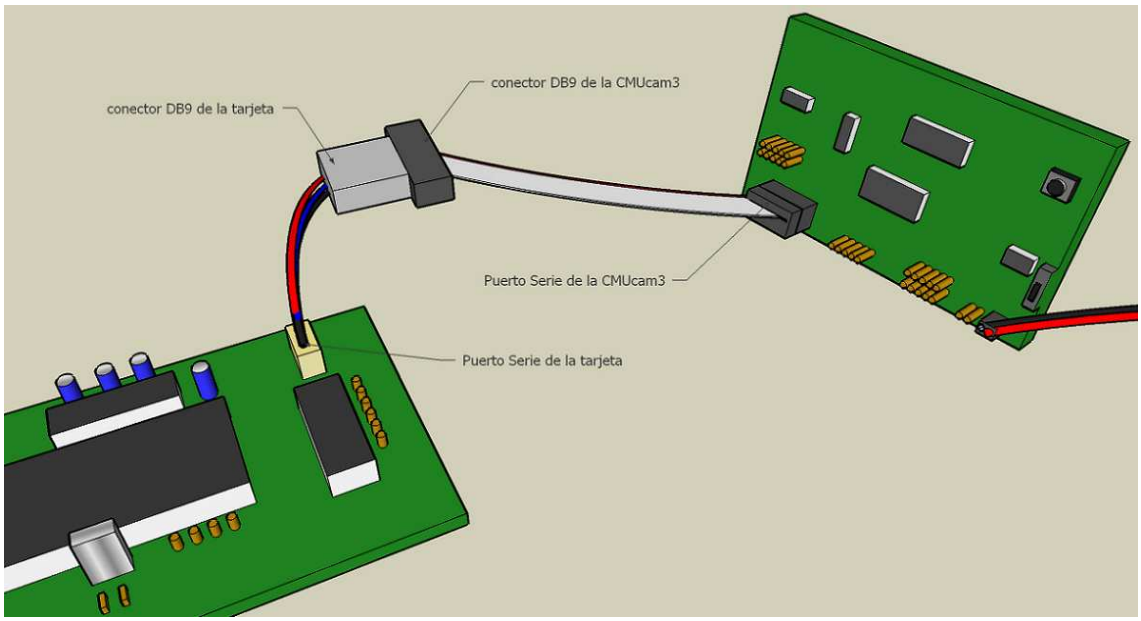


Figura 4.6b conexión física de la CMUcam3 y la tarjeta controladora

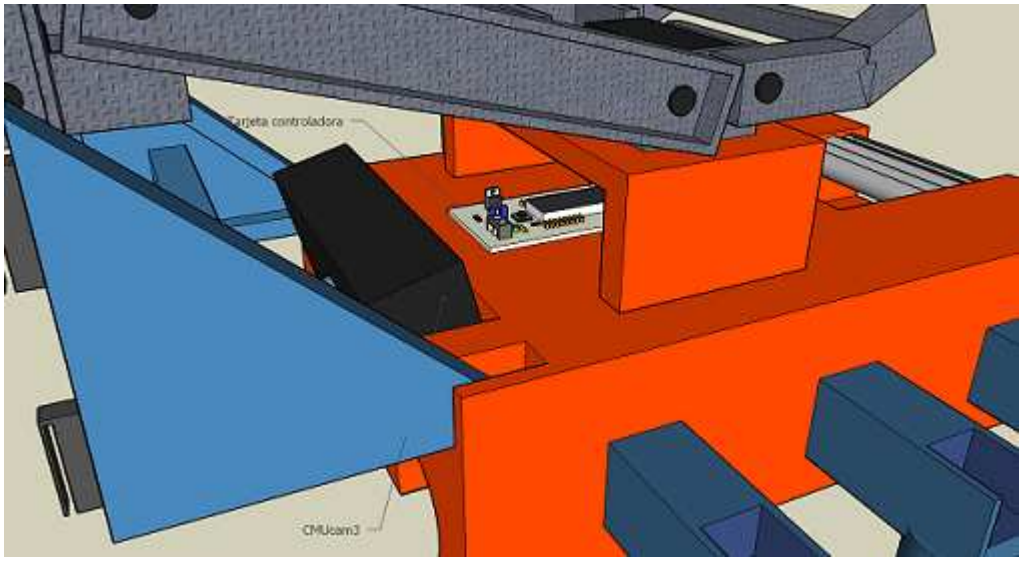


Figura 4.6c Conexión del sistema

4.5 Algoritmos de programación del robot limpiador de playa

En esta sección se explicará el algoritmo de programación del robot limpiador de playa. Los programas referentes a esta sección se encuentran en el apéndice A y sus diagramas de flujo son mostrados aquí para tener una mejor comprensión de lo que se programó.

El primer diagrama de flujo que se desarrolló es el de la navegación del robot, donde el robot tratara de recorrer un entorno desconocido, buscando basura que recolectar, la figura 4.7a muestra el diagrama de flujo de la función main del programa. La función main requiere de dos funciones llamadas "inicializa_puertos()" "inicializa_interrupciones()" que se muestran en las figura 4.7b y 4.7c respectivamente.

El robot se mantiene navegando hasta que la CMUcam3 detecte algún objeto, en ese momento la CMUcam3 le manda a la tarjeta controladora por el puerto serial un carácter indicándole hacia donde se deberá de mover para poder acercarse al objeto. La recepción del carácter produce una interrupción en el programa de navegación del robot e inmediatamente se prosigue a ejecutar el código de interrupción serial que se muestra en la figura 4.7d, 4.7e y 4.7f, donde el robot se dirigirá hacia donde se encuentra el objeto que la CMUcam3 detectó. Una vez localizado el objeto, el robot ejecutará la rutina del control del brazo mecánico para poder recolectar el objeto, las funciones "PWM_GENERATION_SERVO1()" , "PWM_GENERATION_SERVO2()", MOTOR_DC_CONTROL_CERRAR() y MOTOR_DC_CONTROL_CERRAR() son necesarias para el control del brazo mecánico. Las figuras 4.7g, 4.7h, 4.7i y 4.7j muestran el diagrama de flujo de estas funciones.

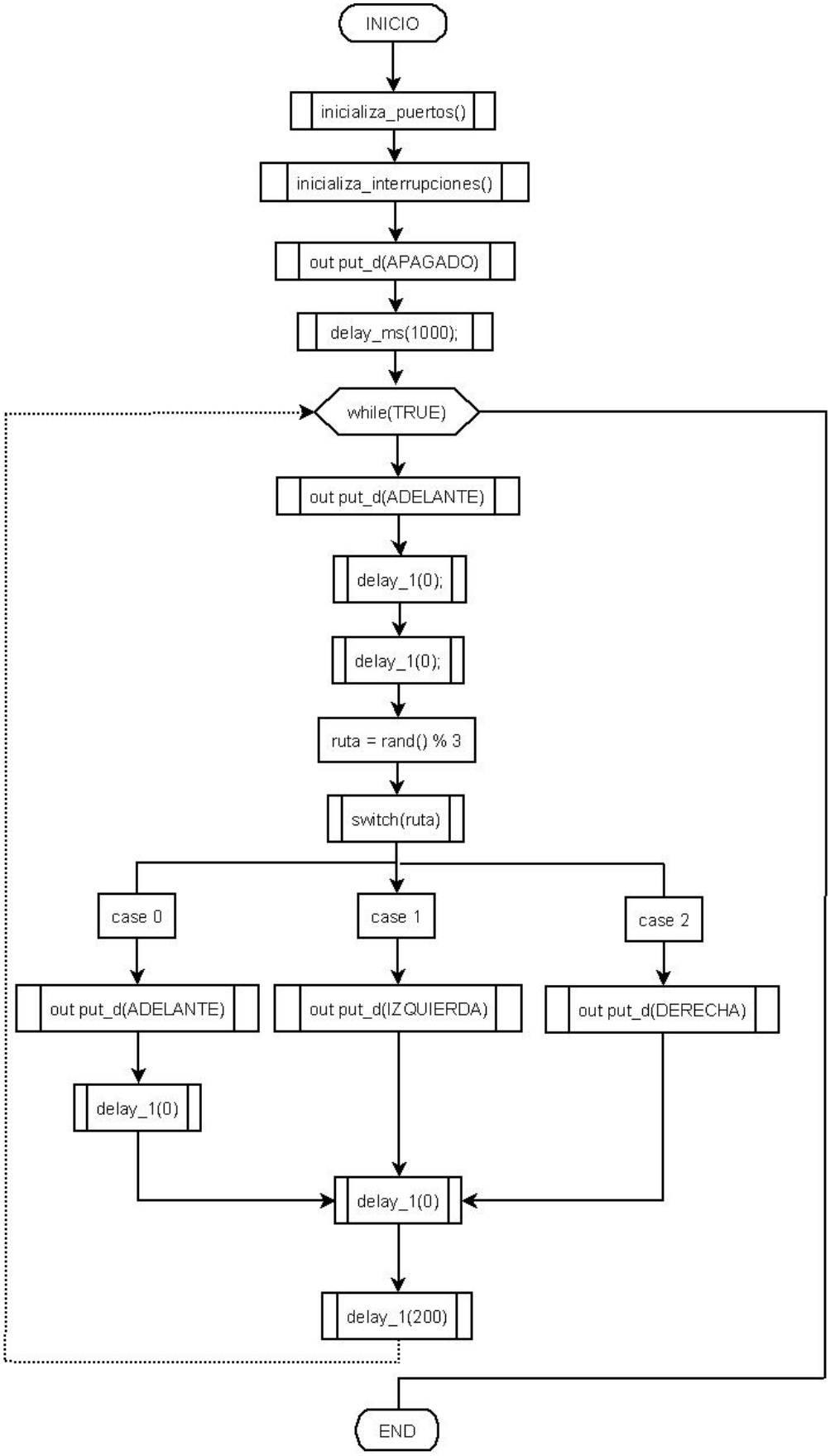


Figura 4.7a diagrama de flujo de la función Main

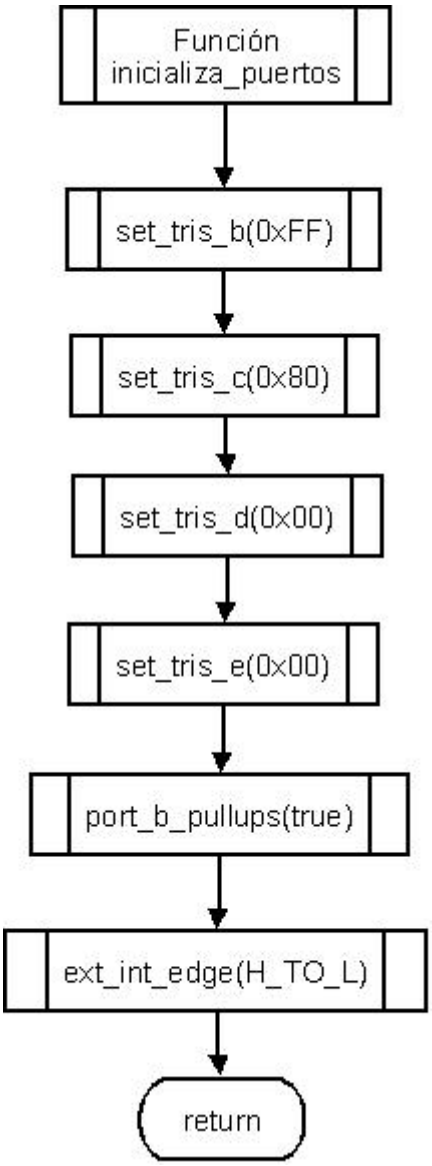


Figura 4.7b Inicialización de puertos

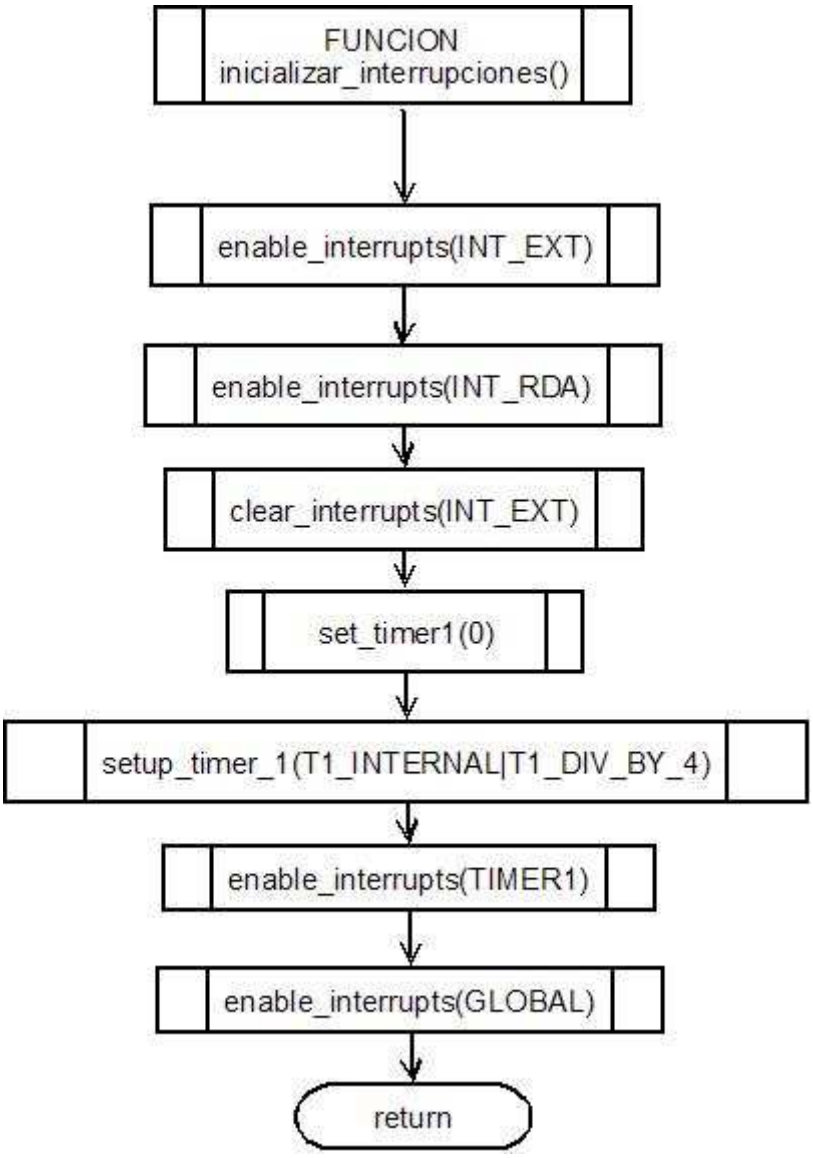


Figura 4.7c Inicialización de interrupciones

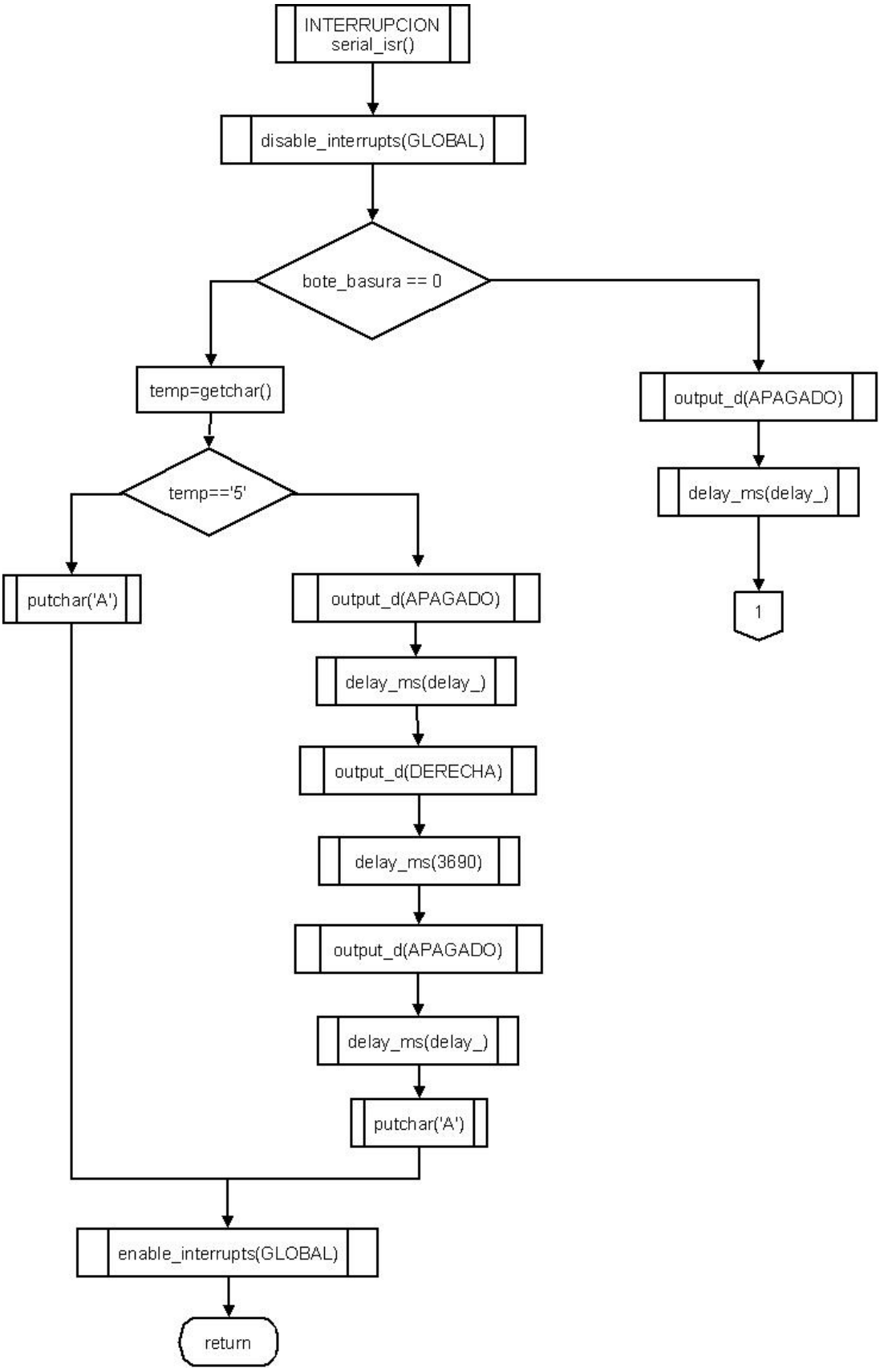


Figura 4.7d Interrupción serial

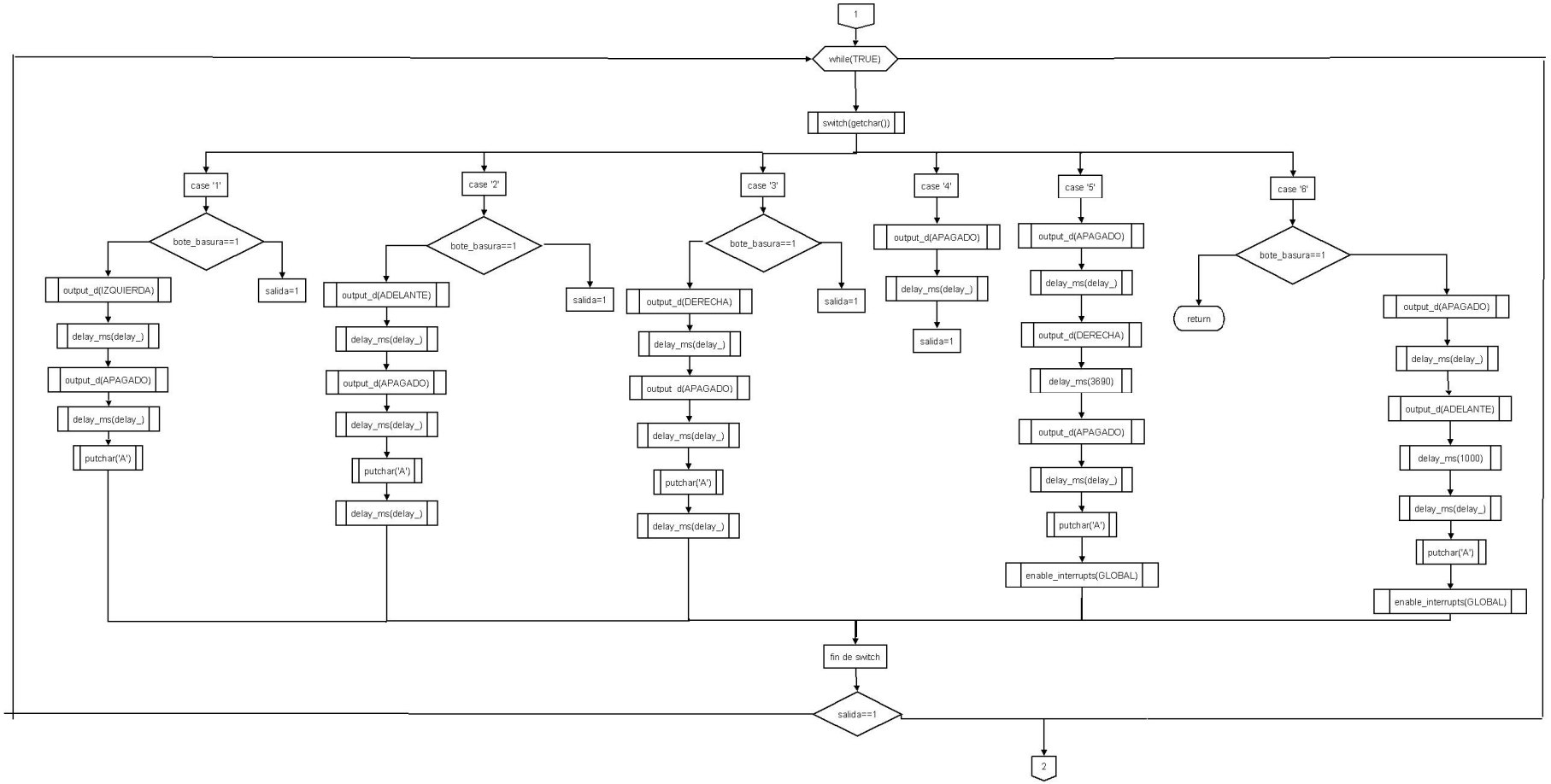


Figura 4.7e Interrupción serial

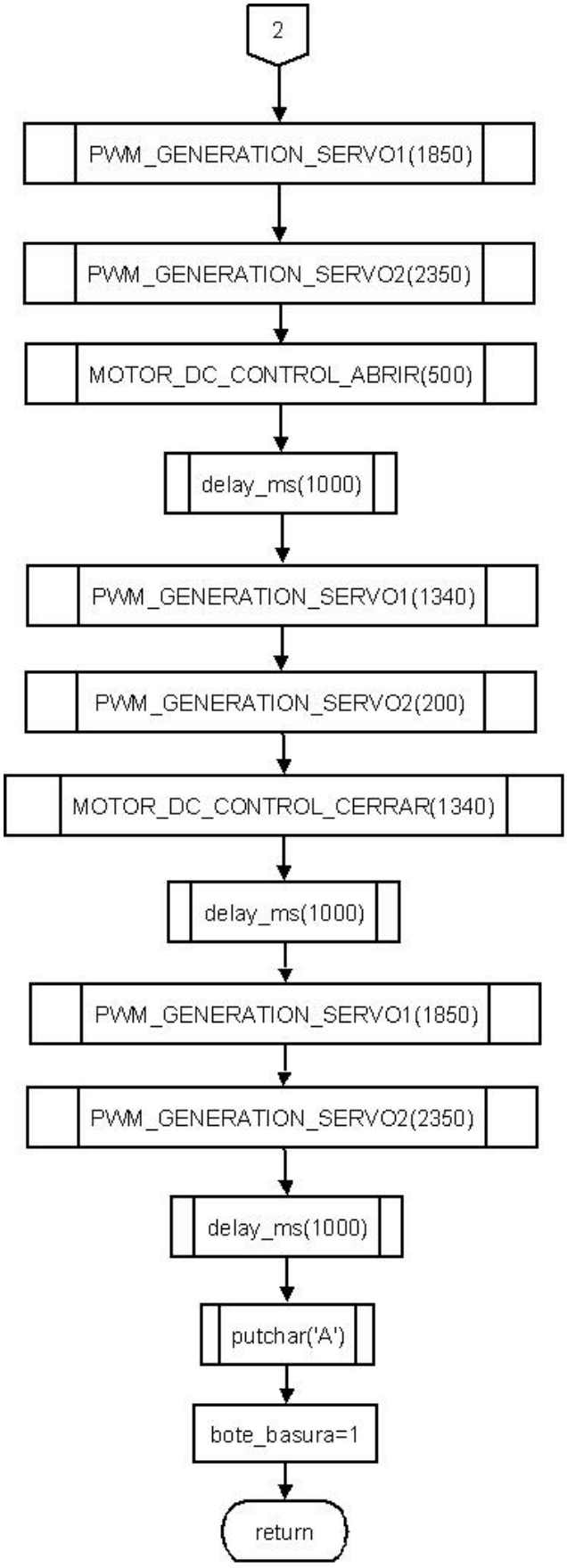


Figura 4.7f Interrupción serial

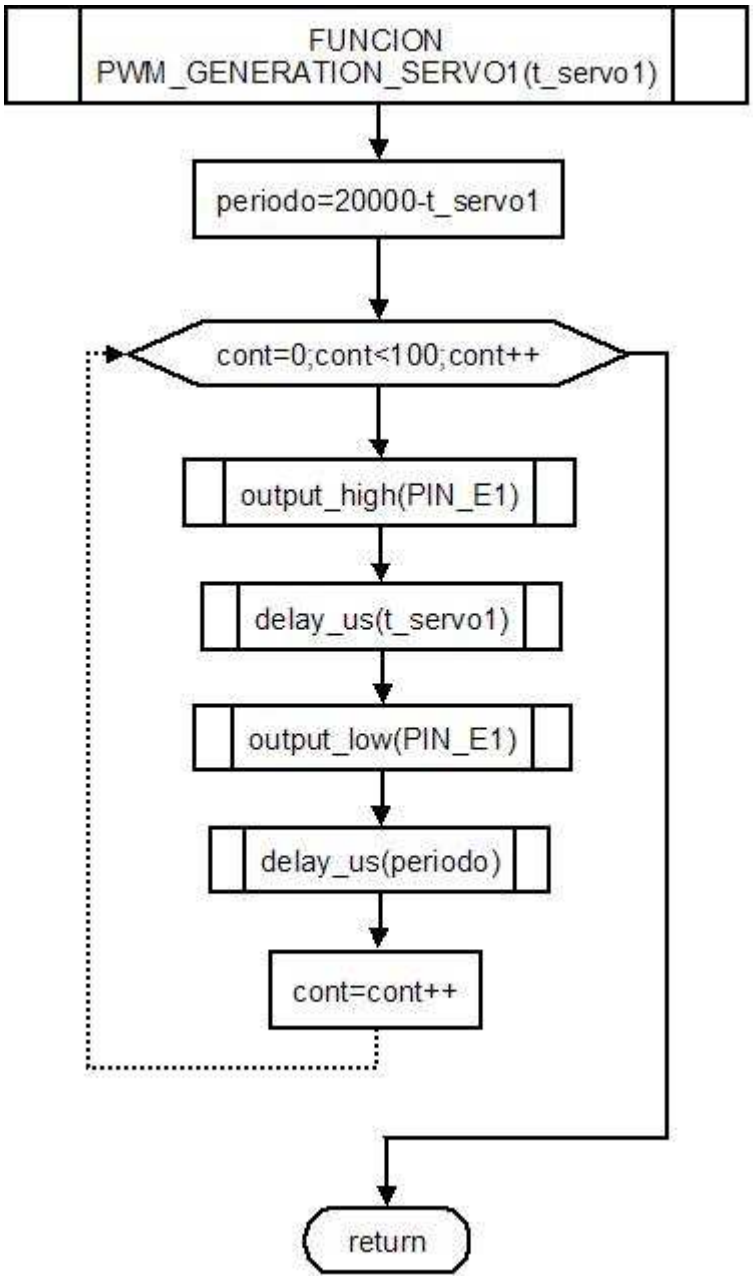


Figura 4.7g Control del servo 1

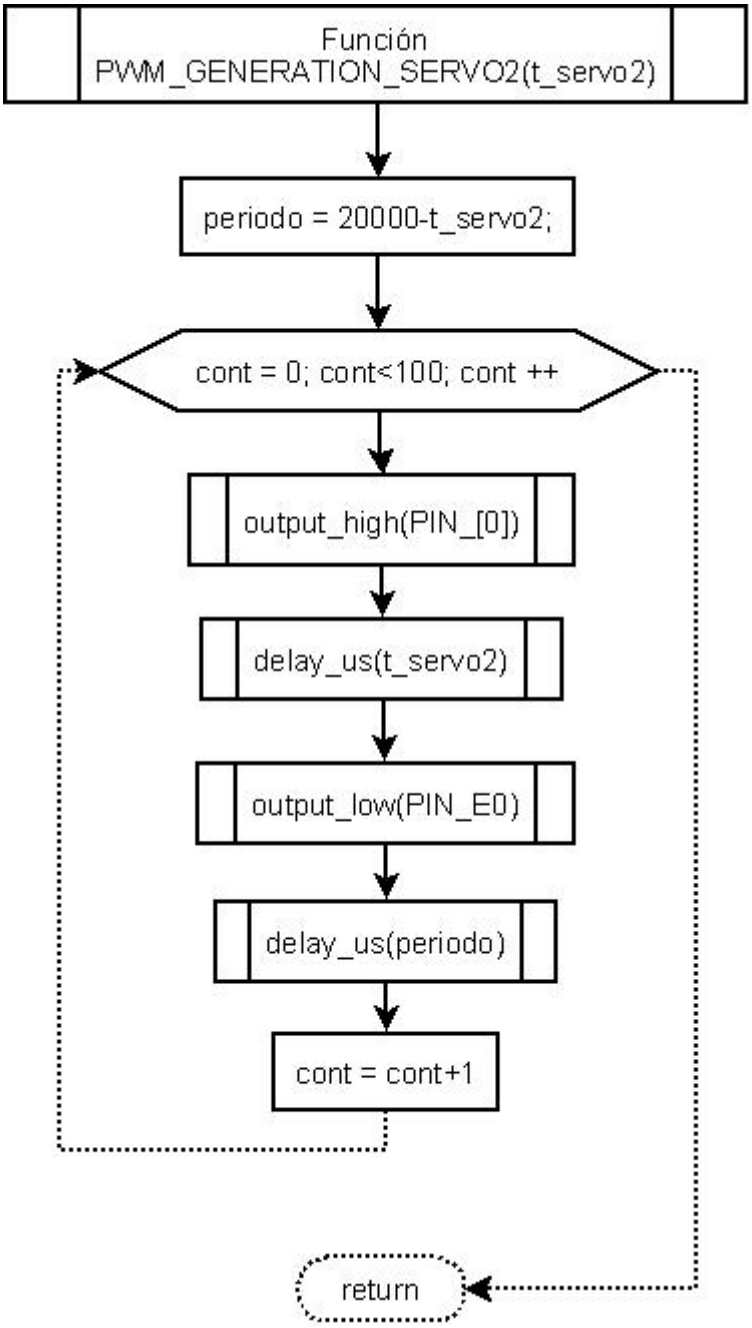


Figura 4.7h Control del servo 2

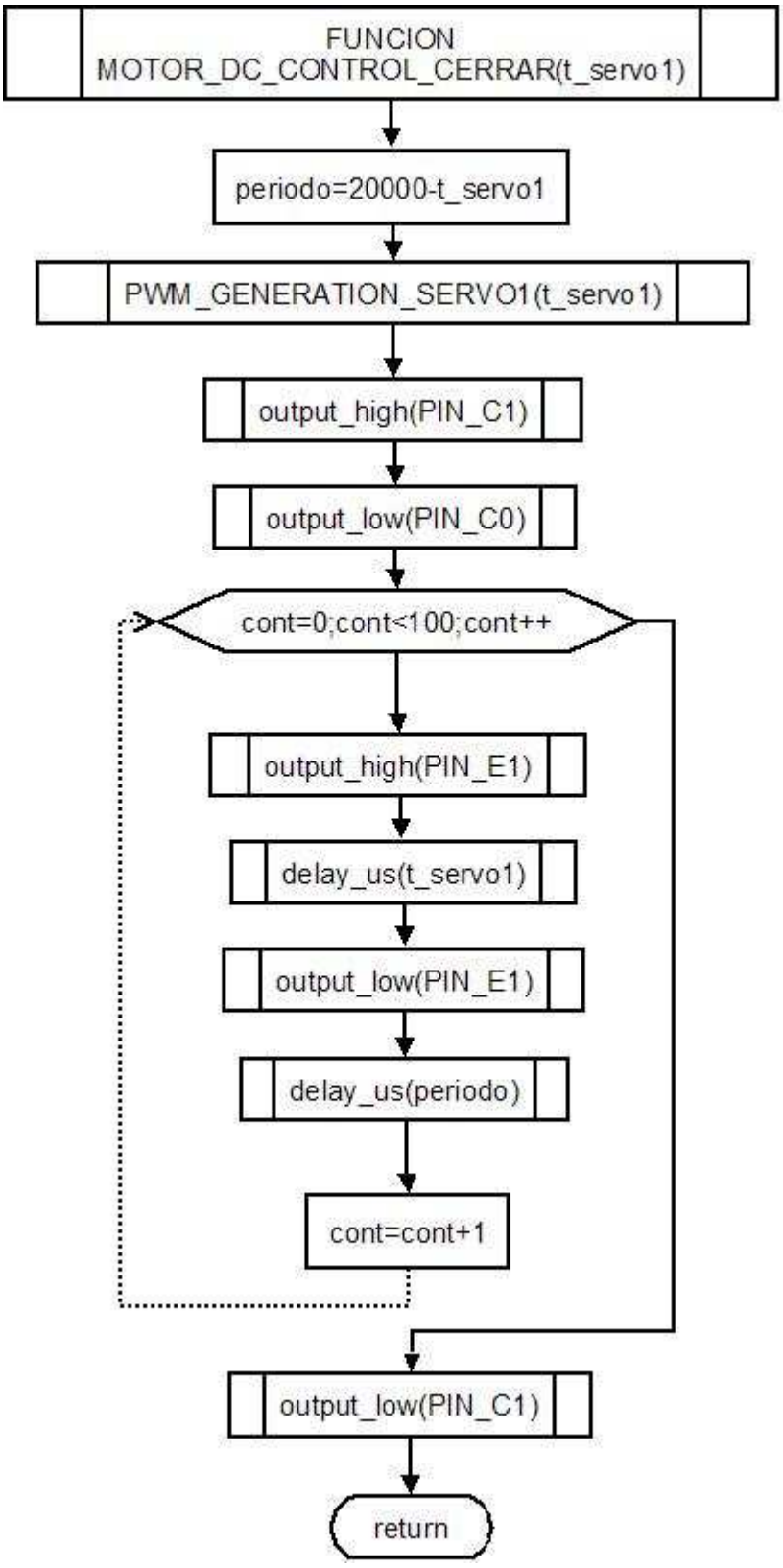


Figura 4.7i Control de las pinzas

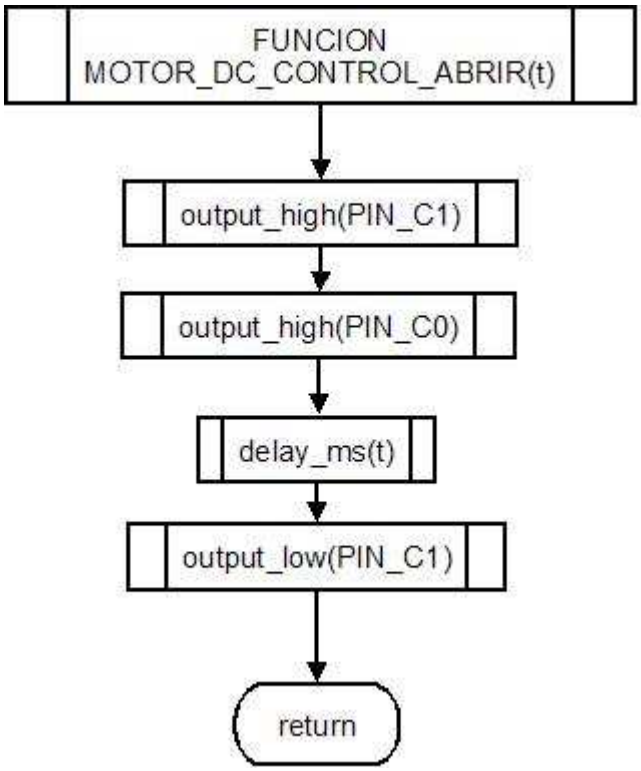


Figura 4.7j Control de las pinzas

El diagrama de flujo de la figura 4.7k muestra el algoritmo que se siguió para evadir un obstáculo, el cual se vale de la interrupción externa que tiene el microcontrolador PIC16F877A. En este mismo diagrama de flujo se muestra el algoritmo que sigue el robot cuando entra en el modo de depositar basura. El algoritmo que se utilizó para sincronizar el movimiento del robot se muestra en la figura 4.7, donde se empleó el timer 1 del PIC. Debido a que no se podía utilizar la función delay_ms() por que desactiva las interrupciones, se emplearon dos funciones para generar retardos en el programa, estas funciones son “delay_l()” y “delay_c” que se muestran en las figuras 4.7m y 4.7n respectivamente

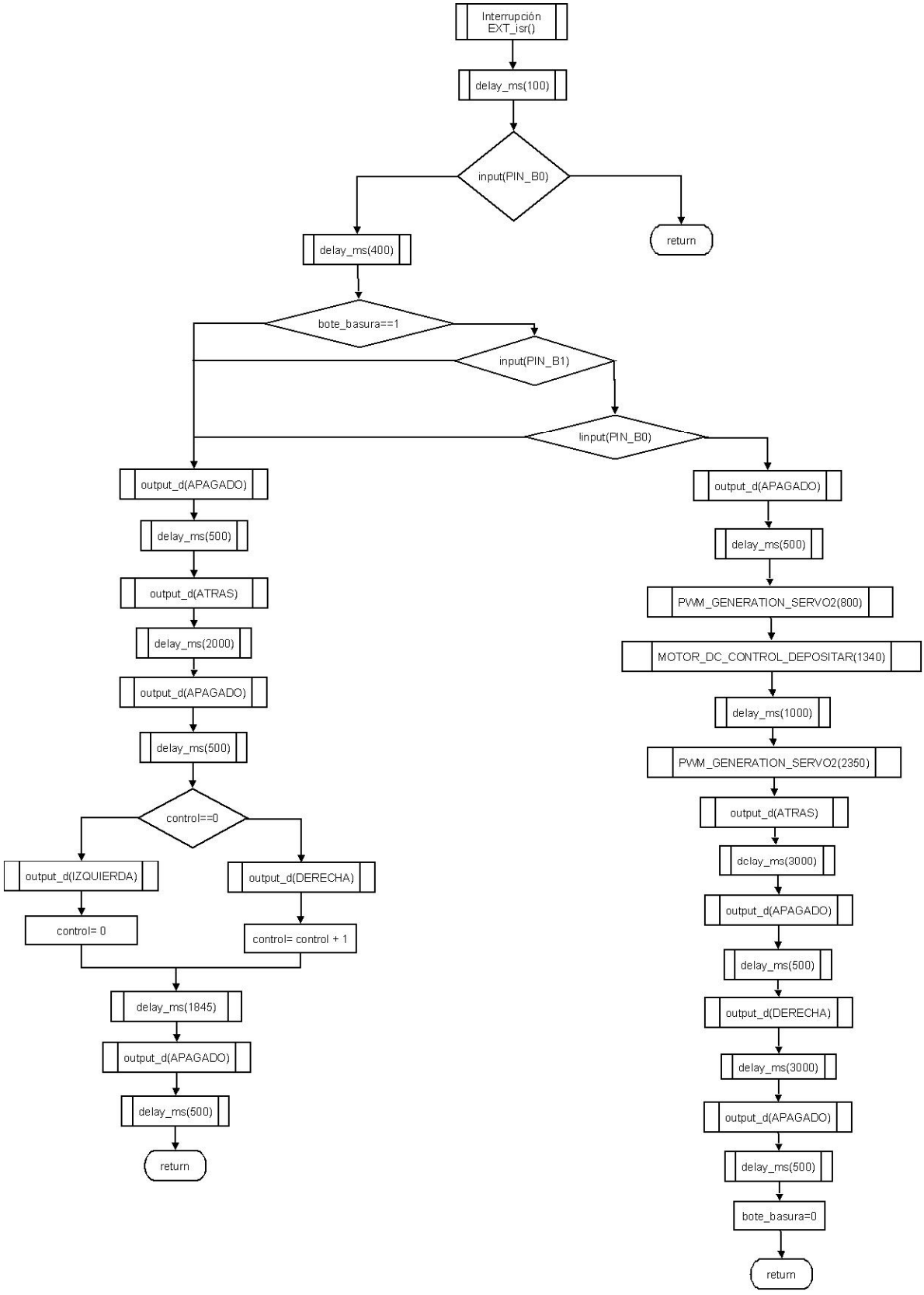


Figura 4.7k Diagrama de flujo de la interrupción externa

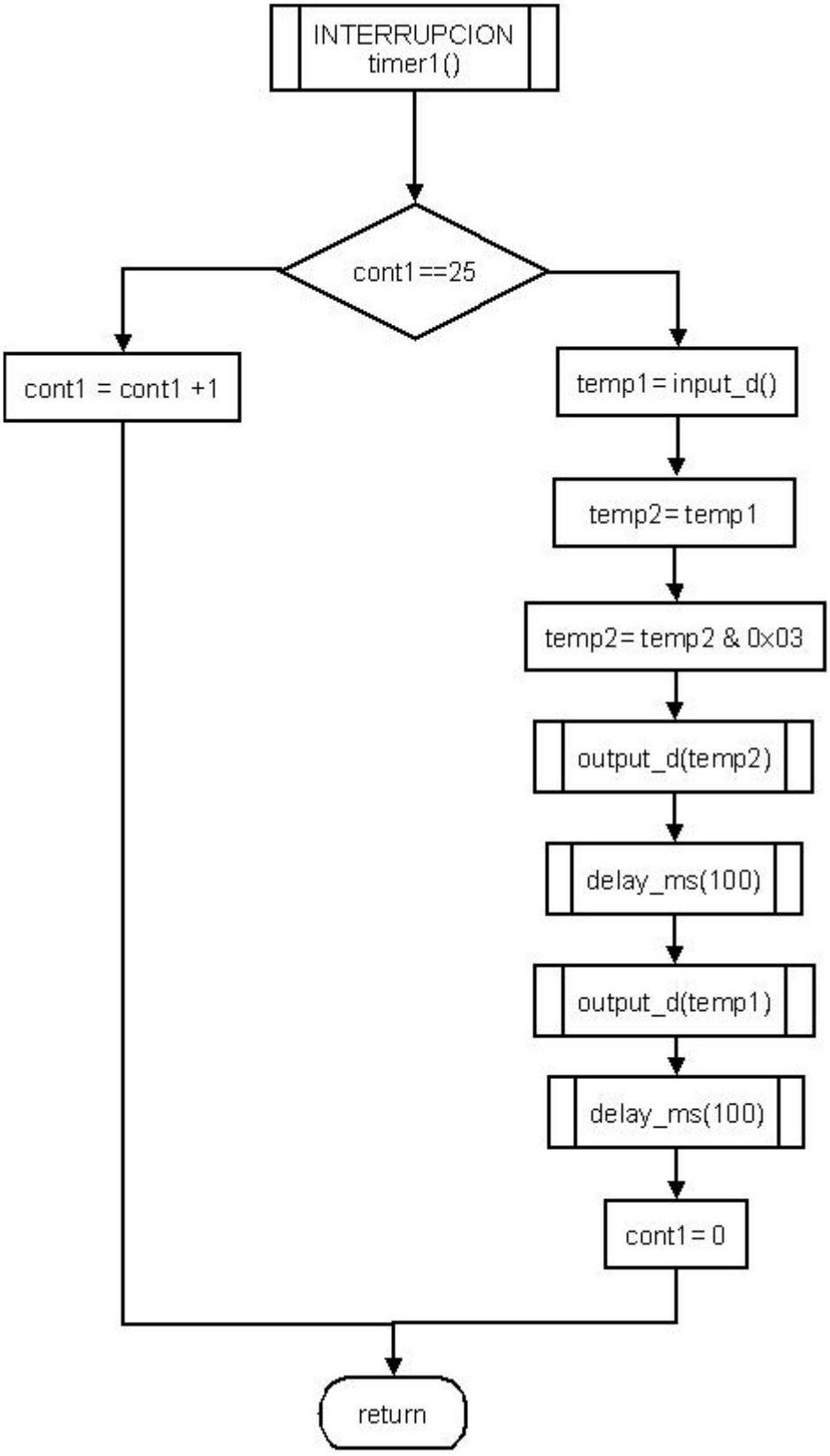


Figura 4.7l Interrupción del TIMER 1

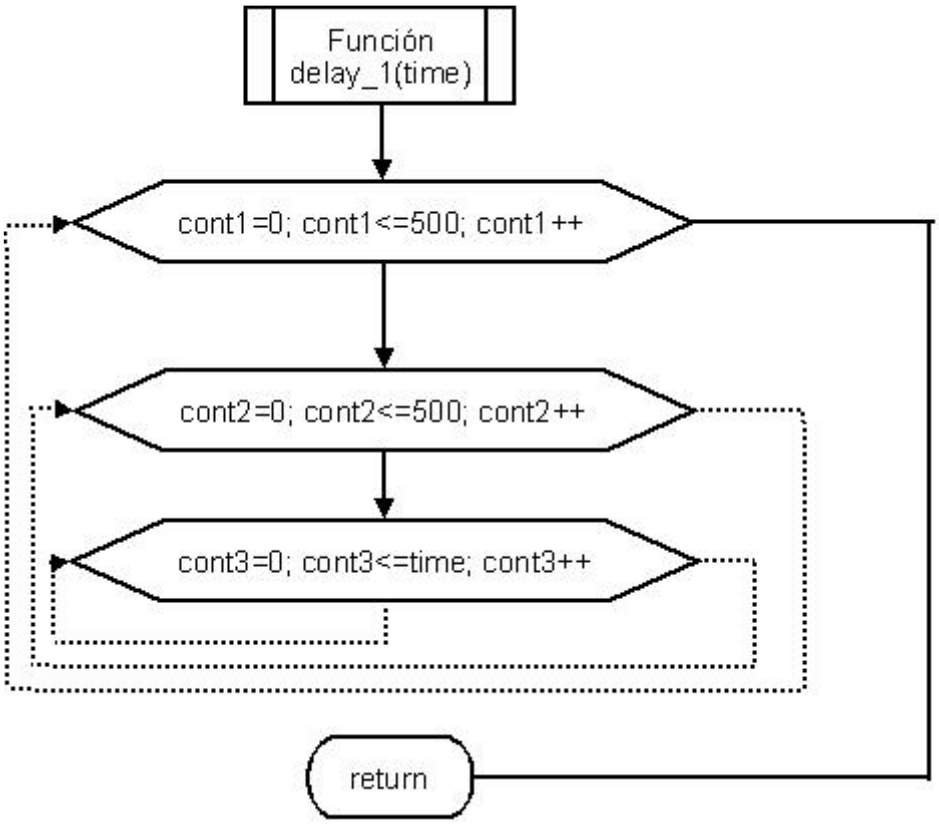


Figura 4.7 m Función delay_l

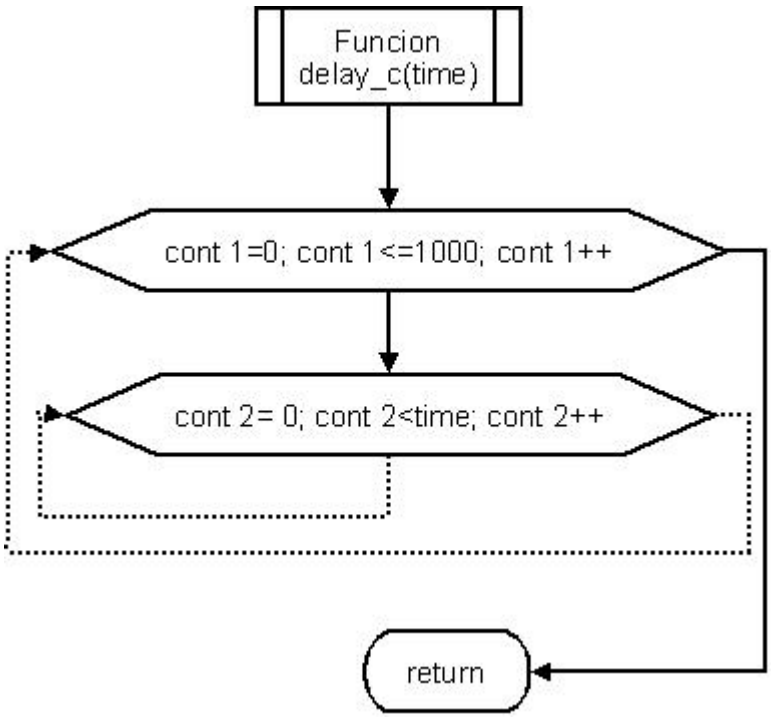


Figura 4.7n Función delay_c

El programa que se utilizó para este trabajo se encuentra en el apéndice A con el nombre de programa 5, que para mejorar su portabilidad se dividió en varios archivos fuente. La figura 4.8 muestra la distribución del proyecto

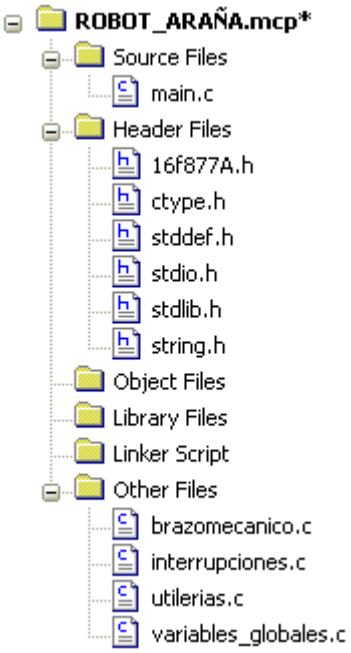


Figura 4.8 Proyecto en MPLAB

La figura 4.8 muestra que los archivos fuente de los que depende el proyecto son:

- ***main.c***: contiene el programa principal de control del robot.
- ***brazomecanico.c****:* contiene todas las funciones y variables necesarias para mover el brazo mecánico.
- ***interrupcones.c***: contiene todas las funciones que inicializan las interrupciones que requirió el proyecto.
- ***utilerias.c***: contiene diversas funciones que se necesitaron para realizar retardos.
- ***variables_globales.c***: contiene las variables globales que se emplearon en este proyecto.

El programa que se empleó en la CMUcam3 se encuentra en el apéndice B como programa 3.

CAPÍTULO V PRUEBAS Y RESULTADOS

5.1 Introducción

En este capítulo se explica las pruebas realizadas al sistema implementado, con el fin de ver su comportamiento y comprobar su funcionamiento. El objetivo del proyecto es implementar un robot limpiador de playa por lo que las pruebas que se hicieron fueron para comprobar su funcionamiento y que cumpliera con el objetivo principal.

5.2 Configuración de las pruebas

Las pruebas se dividen en cuatro partes:

1. Pruebas con la tarjeta controladora

    - Comunicación con la tarjeta controladora.
    - Control de movimiento del robot.
    - Control de movimiento del brazo mecánico.

2. Pruebas con la CMUcam3.

    - Comunicación con la CMUcam3.
    - Seguimiento de colores.

3. Pruebas de comunicación entre la CMUcam3 y la tarjeta controladora.

    - Envío de caracteres a la tarjeta controladora desde la cmucam3.
    - Sincronización de la comunicación entre la CMUcam3 y la tarjeta controladora.

4. Pruebas del sistema completo.

    - Búsqueda de residuos.
    - Evasión de obstáculos.
    - Recolección de residuos.
    - Deposito de residuos.

Las pruebas se hicieron con las siguientes características.

- ✓ Computadora laptop, procesador Intel Pentium a 1.73Ghz, 512 MB en RAM, sistema operativo Windows XP y el puerto USB.
- ✓ Temperatura ambiente 22 °C.
- ✓ Para ajustar los rangos del color a buscar se utilizó CMUcam3 Frame Grabber.
- ✓ Un depósito de cartón con las medidas de 20x20x20 cm.

Los siguientes puntos muestran las pruebas que se realizaron para que el sistema cumpla con los objetivos planteados.

5.2.1 Configuración de las pruebas para la tarjeta controladora.

Para realizar las pruebas con la tarjeta controladora, se empleó la hyperterminal de Windows con las siguientes especificaciones:

Bits pos segundo	19200
Bits de datos	8
Paridad	ninguna
Bits de parada	2
Control de flujo	ninguno

Utilizando la hyperterminal y la comunicación serial con la tarjeta controladora, se depuraron los programas escritos en C para la tarjeta. En los siguientes apartados se muestran las pruebas realizadas para el control del sistema.

5.2.1.1 Comunicación con la tarjeta controladora

Para verificar el buen funcionamiento de la tarjeta controladora, se hizo un programa que enviara y recibiera datos provenientes desde una computadora, la figura 5.1 muestra la conexión correcta para esta prueba.

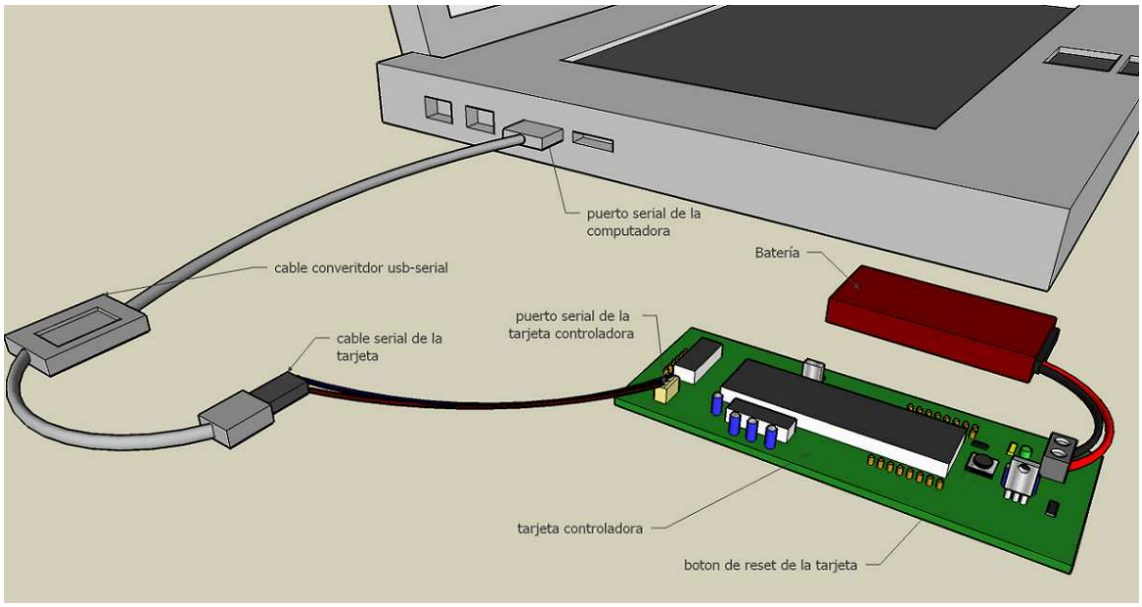


Figura 5.1 Comunicación serial entre la tarjeta y la computadora

El siguiente código (ver programa 1 del apéndice A) muestra el programa principal descargado en la tarjeta para verificar el funcionamiento correcto de la misma.

5.2.1.2 Control de movimiento del robot

Las posibles combinaciones que se pueden obtener por el puerto D (que es el encargado de manejar los dos puentes H) se muestran en la tabla 5.1a. A partir de esta tabla se realizó un programa que nos permitiera conocer el sentido de avance del robot, el programa 6 del apéndice A muestra el código empleado para realizar estas acciones y la

tabla 5.1b muestra las conclusiones a las que se llegó. La figura 5.2 muestra los elementos necesarios para esta prueba.

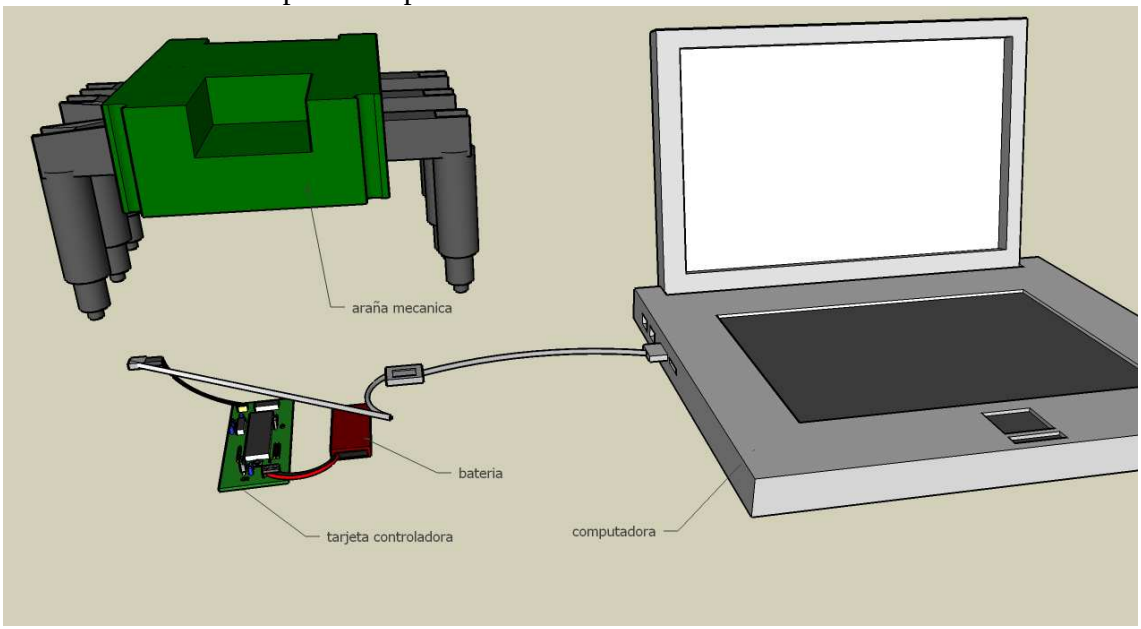


Figura 5.2 Elementos necesario para la prueba

BINARIO	HEXADECIMAL
00001010	0x0A
00001001	0x09
00000110	0x06
00000101	0x05
00000000	0x00

Tabla 5.1a Posibles salidas por el puerto D

BINARIO	HEXADECIMAL	MOVIMIENTO
00001010	0x0A	DERECHA
00001001	0x09	ADELANTE
00000110	0x06	ATRÁS
00000101	0x05	IZQUIERDA
00000000	0x00	APAGADO

Tabla 5.1b Conclusiones de la prueba

Esta prueba fue necesaria para encontrar los comandos necesarios para controlar el sentido de movimiento del robot.

5.2.1.3 Control de movimiento del brazo mecánico

La figura 5.3a muestra los elementos necesarios para esta prueba y el programa 4 del apéndice A muestra el código del programa que se utilizó para hacer pruebas con el control del brazo mecánico, donde se busco principalmente encontrar la señal PWM necesaria para mover el brazo a las posiciones requeridas. El programa, por medio de la

hyperterminal nos permitirá ingresar diferentes tiempos en alto, generando la señal PWM para mover cada servo motor a la posición que se quiera, tomando en cuenta que el rango de tiempos que podemos dar están en el intervalo de .1 ms a 2.3 ms, fuera de este rango el motor vibrara. De acuerdo a las pruebas realizadas, los tiempos en alto de la señal PWM se muestran en la tabla 5.2 y en la figura 5.3b, 5.3c, 5.3d y 5.3e respectivamente.

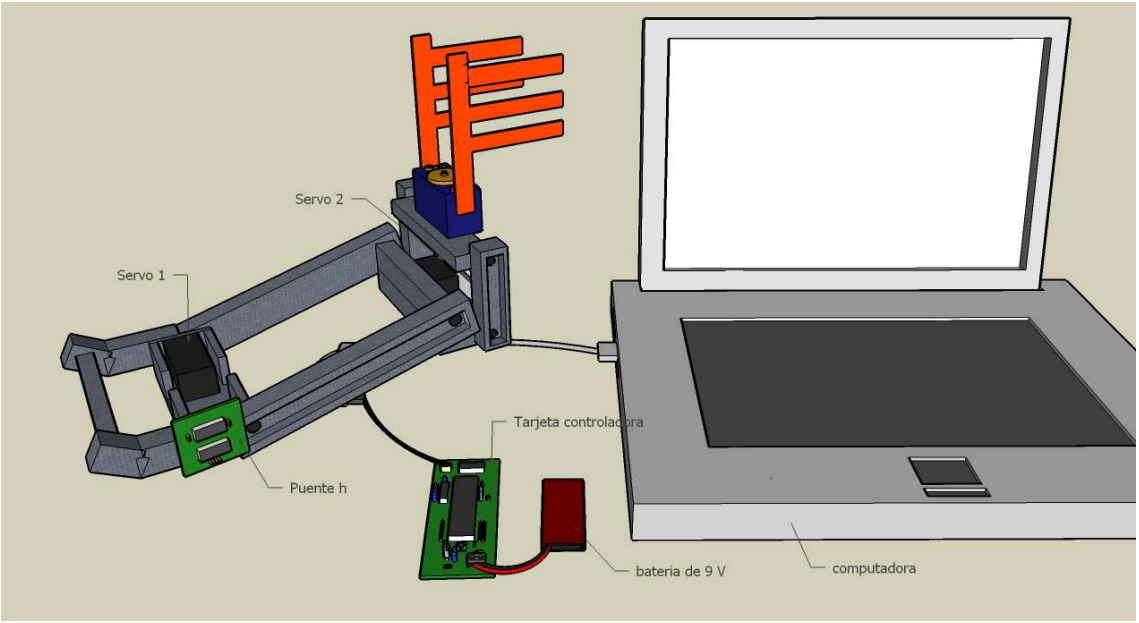


Figura 5.3a elementos a utilizar para el control del brazo mecánico

Subir brazo	Tiempo en [ms]
Servo 1	1.85
Servo 2	2.35
Bajar brazo	
Servo 1	1.34
Servo 2	.2

Tabla 5.2 resultados de la prueba de control del brazo mecánico

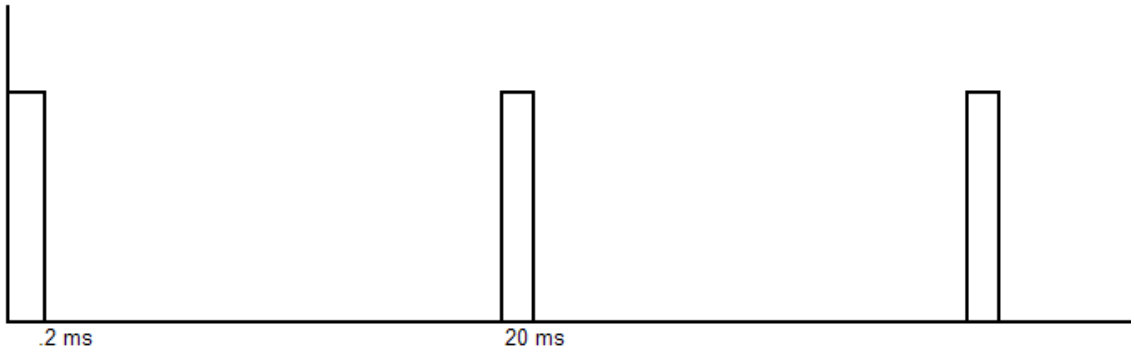


Figura 5.3b señal PWM para mover el servo 2 se baja el brazo

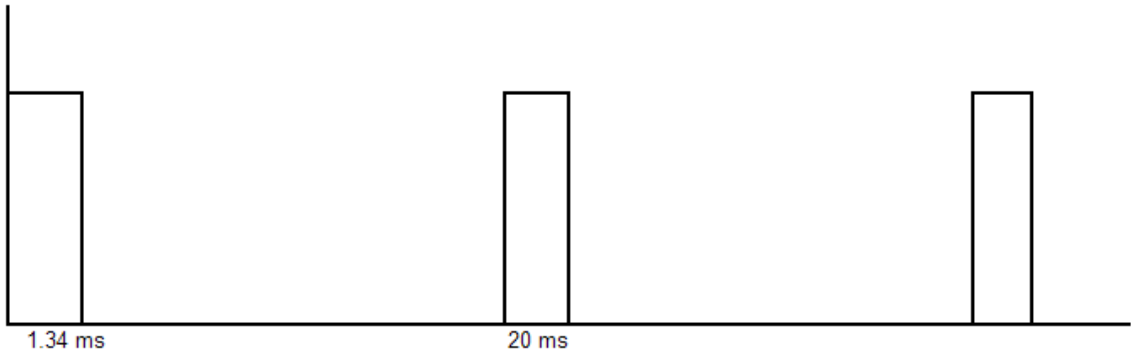


Figura 5.3c señal PWM para mover el servo 1 cuando se baja el brazo

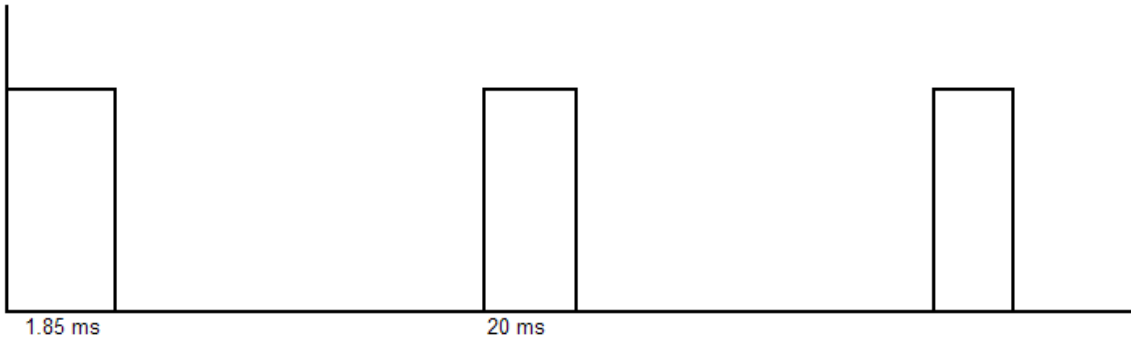


Figura 5.3d señal PWM para mover el servo 1 cuando se sube el brazo

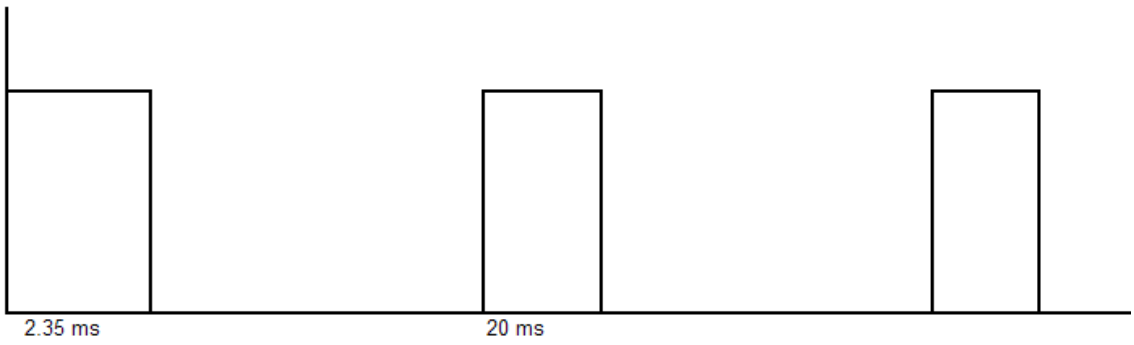


Figura 5.3e señal PWM para mover el servo 2 cuando se sube el brazo

Para controlar la pinza del robot en el mismo programa 4 del apéndice A se modificaron los tiempos en que opera la pinza cuando abre y cierra, las pruebas nos dieron los siguientes resultados que se muestran en la tabla 5.2, donde se muestra el valor que se debe pasar a las funciones "MOTOR_DC_CONTROL_CERRAR()" y "MOTOR_DC_CONTROL_ABRIR()" respectivamente.

Estado de las pinzas	Valor del argumento
Cerrado	500
Abierto	1340

Tabla 5.2

5.2.2 Configuración y pruebas con la CMUcam3

Para realizar las pruebas con la CMUcam3, se empleó la hyperterminal de Windows con las siguientes configuraciones:

Bits pos segundo	115200
Bits de datos	8
Paridad	ninguna
Bits de parada	2
Control de flujo	ninguno

Empleando la hyperterminal y la comunicación serial con la CMUcam3, se depuraron los programas escritos en C para la cámara. A continuación se muestran las pruebas realizadas para el control del sistema.

5.2.2.1 Comunicación con la CMUcam3

La figura 5.4 muestra la conexión correcta para comunicarnos con la CMUcam3 mediante una computadora.

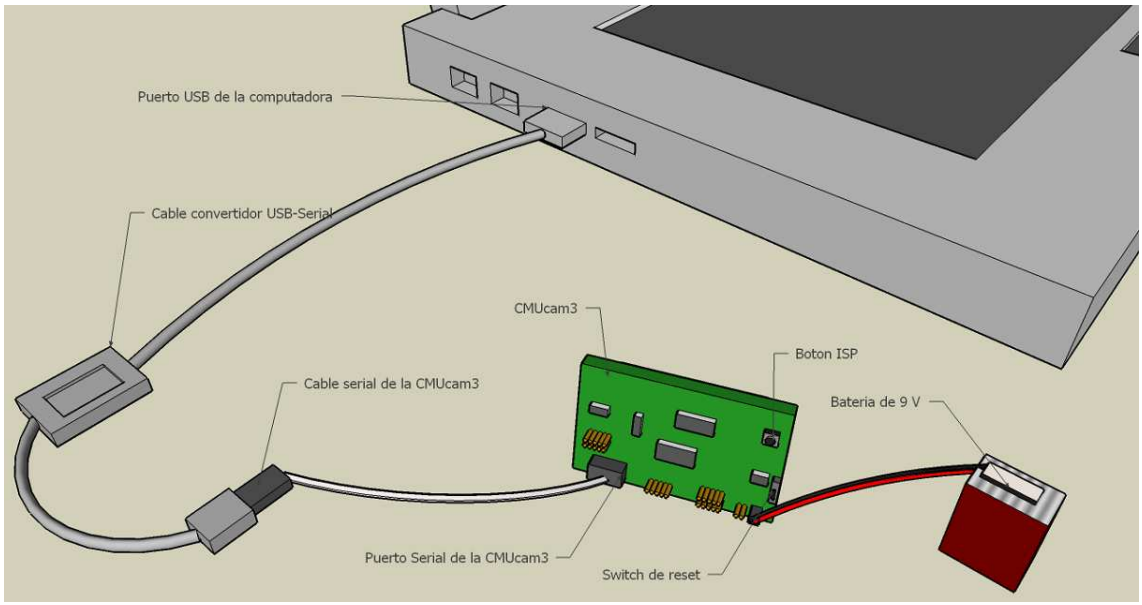


Figura 5.4 Comunicación serial con la CMUcam3

El programa 1 del apéndice B muestra un programa de prueba para comprobar el funcionamiento de la tarjeta. En la página de la CMUcam3 (ver referencia número 2) nos recomiendan descargar en la tarjeta el programa hello-worl.c para comprobar el correcto funcionamiento de la tarjeta.

En las siguientes secciones se muestran las pruebas que se realizaron en la CMUcam3 para poder cumplir con el objetivo de detectar los colores rojo, verde y azul.

5.2.2.2 Seguimiento de colores

Para hacer esta prueba se descargó en la CMUcam3 el programa “simple_track_color.c” que viene en la página principal de la CMUcam (ver referencia número 2). Un programa modificado a partir de éste se encuentra en el apéndice B como programa 2 en este se modificaron los rangos de búsqueda. Para poder ajustar los valores correctos en que se encuentra el color que se desea detectar se empleó la utilidad CMUcam3 Frame Grabber y se le descargó a la CMUcam3 el programa cmucam2.c que se encuentra en el directorio de proyectos del cc3. El código de la cmucam2.c es extenso por lo que se decidió no incluirlo en este trabajo. Este programa permite ingresarle comandos como los vistos en el capítulo 3 sección 3.4.3. Un comando muy útil fue el de GM (get mean) que nos da el valor promedio del tono de color que se ve en la pantalla de la CMUcam3 Frame Grabber, de esta manera cuando queremos detectar un color se coloca en frente de la pantalla de la CMUcam3 el color que se quería detectar, seguido del comando GM, a continuación nos regresaba el color promedio de la imagen que se esta viendo, de esta manera nos damos una idea de los intervalos en que se encuentran los valores del color que deseamos ver, para después colocarlos dentro del programa simple_track_color y verificar que efectivamente ve el color.

Debido a que estos valores cambian mucho de acuerdo a la iluminación del medio se tienen que estar haciendo varias pruebas para obtener un rango del color que se desea seguir. A continuación se muestra el segmento de código que pertenece al programa simple_track_color.c que se modifica para colocar los nuevos rangos del color que se desea detectar.

```
// parámetros para elección del color a buscar

  t_pkt.lower_bound.channel[CC3_CHANNEL_RED]   = 255;
  t_pkt.upper_bound.channel[CC3_CHANNEL_RED]   = 150;
  t_pkt.lower_bound.channel[CC3_CHANNEL_GREEN] = 0;
  t_pkt.upper_bound.channel[CC3_CHANNEL_GREEN] = 50;
  t_pkt.lower_bound.channel[CC3_CHANNEL_BLUE]  = 0;
  t_pkt.upper_bound.channel[CC3_CHANNEL_BLUE]  = 50;
```

Al realizar varias pruebas con la utilidad CMUcam3 Frame Grabber, nos dimos cuenta que en realidad estos valores se tiene que calibrar siempre que se desee utilizar el sistema en un nuevo ambiente; debido a las variaciones de luz que existen en el medio. Por lo que los programas que se muestran en el apéndice B necesitan ser calibrados si se desean emplear en futuros proyectos.

5.2.3 Pruebas entre la comunicación de la CMUcam3 y la tarjeta controladora

Debido a que la CMUcam3 se comunica a 115200 bits por segundo mientras que la tarjeta controladora lo hace a 19200, se tiene que configurar la velocidad de la CMUcam3 a 19200 bits por segundo para sincronizar la comunicación. Una vez hecha esta observación se comenta que en las siguientes secciones se describen las pruebas realizadas para sincronizar la comunicación entre la CMUcam3 y la tarjeta controladora.

5.2.3.1 Envío de caracteres a la tarjeta controladora desde la CMUcam3

Para verificar la correcta comunicación entre la tarjeta controladora y la CMUcam3 se descargó en la tarjeta controladora el programa 7 del apéndice A y en la CMUcam3 el programa 4 del apéndice B. El programa 7 del apéndice A recibe comandos

provenientes de la CMUcam3 que hacen que el robot se mueva de acuerdo al siguiente diagrama de flujo de la figura 5.5a. EL diagrama de flujo del programa emisor se muestra en la figura 5.5b. Esta prueba tiene como objetivo, verificar que los comandos que manda la CMUcam3 a la tarjeta controladora, los ejecute permitiendo de esta manera la comunicación entre el sistema de visión y la tarjeta controladora.

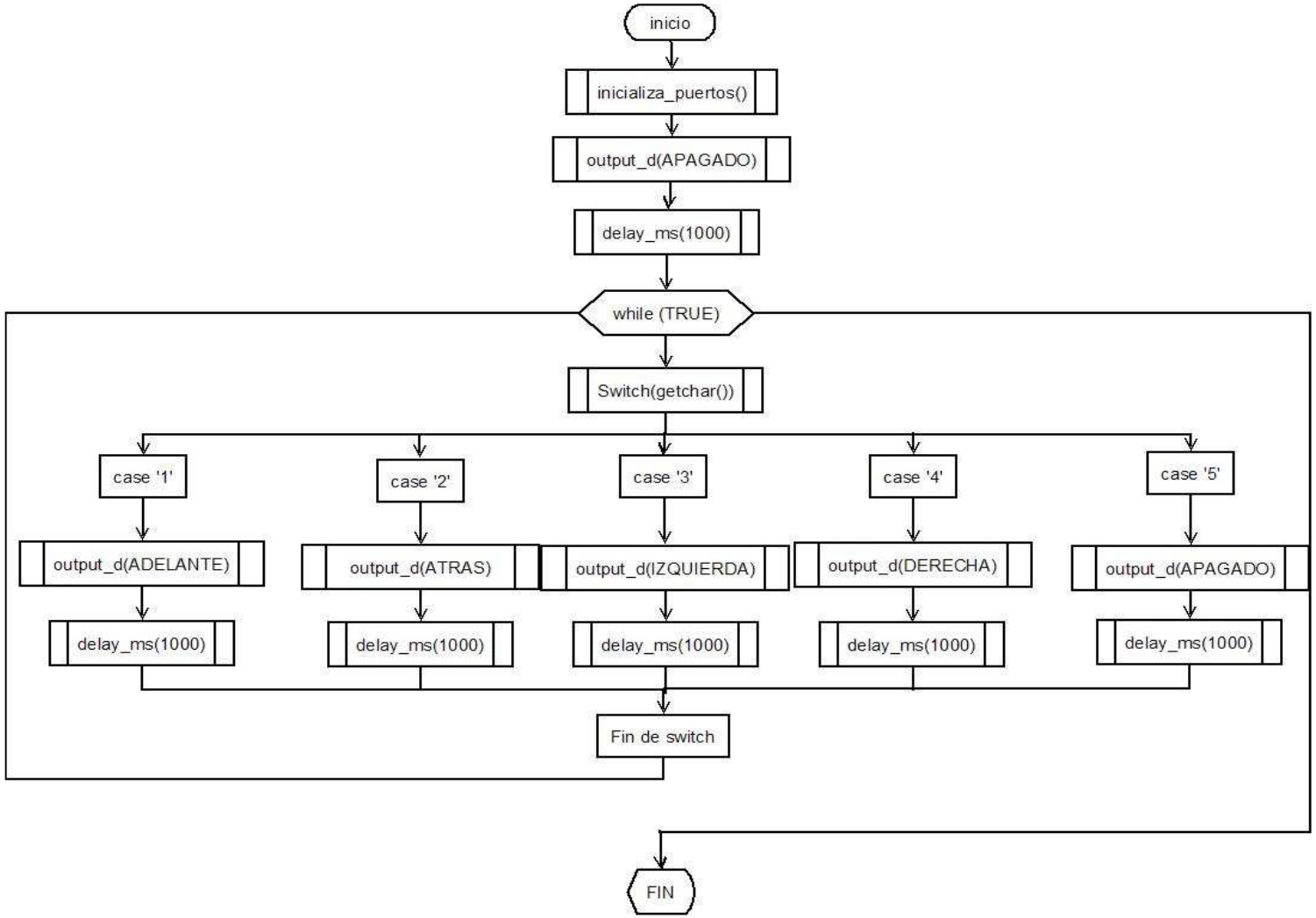


Figura 5.5a diagrama de flujo del receptor

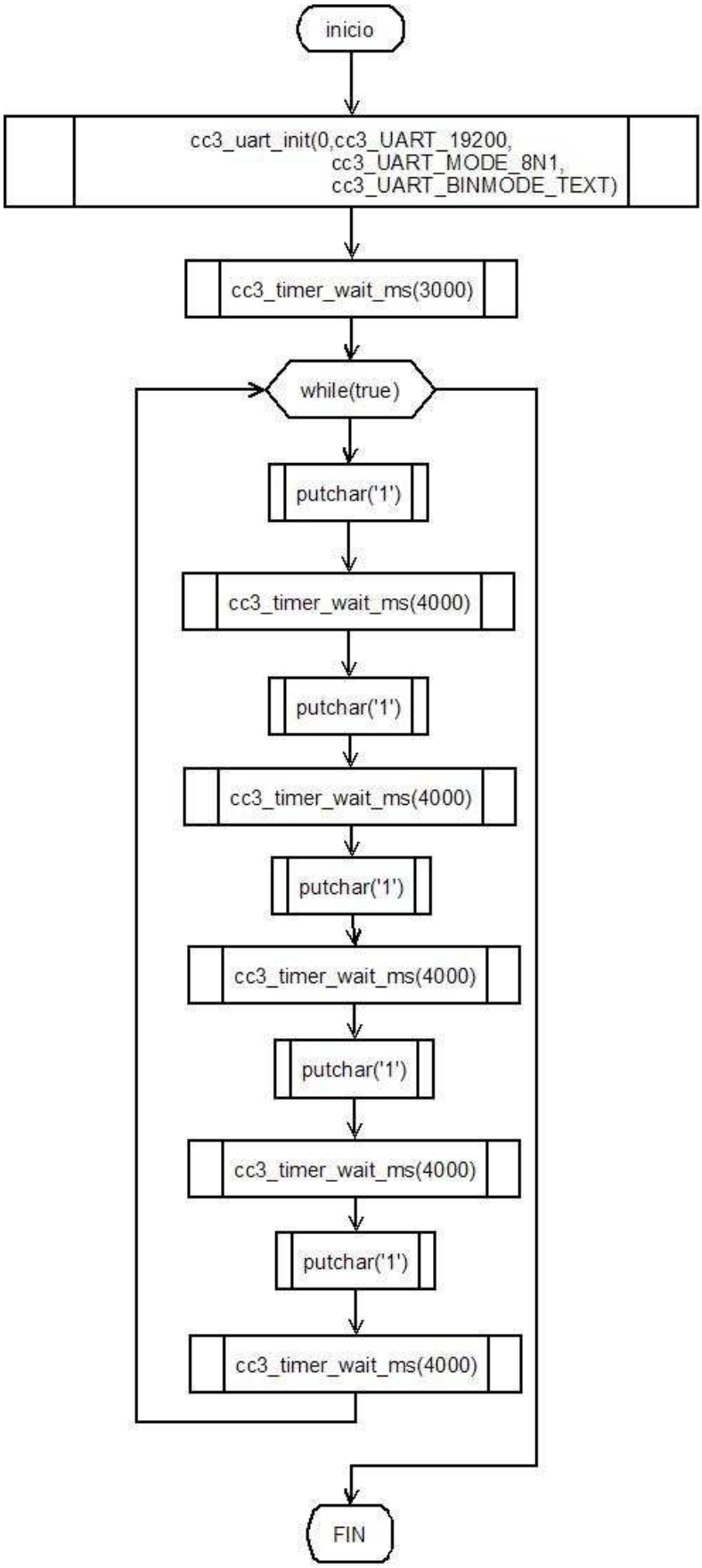


Figura 5.5b diagrama de flujo del emisor

5.2.3.2 Sincronización de la comunicación entre la CMUcam3 y la tarjeta controladora

Debido a que la CMUcam3 es más rápida que la tarjeta controladora, nos percatamos que en muchas ocasiones el buffer de la tarjeta controladora se llenaba y dejaba de recibir datos por lo que se necesito de una técnica para sincronizar la comunicación de los dos dispositivos. Esta técnica consistió en hacer que la CMUcam3 esperara a que la tarjeta controladora terminara de realizar las acciones que había comenzado a ejecutar debidas a un dato previo recibido, esperando por el carácter 'A'. A continuación se muestra el segmento de código de la función que implementa esta espera, el nombre de la función se llama "semaforo()". Cabe mencionar que no es necesario hacer una función que sólo tenga unas cuantas líneas de código pero para mejorar la legibilidad del programa se creó esta función.

```
void semaforo(){

  //printf("\n\resperando a que termine el PIC");
   while(true){
            if (getchar()=='A')
            break;
            }

   }
```

De acuerdo a esto, toda interrupción en el programa de la tarjeta controladora deberá regresar un carácter 'A' una vez terminada su tarea.

5.2.4 Pruebas con el sistema completo

A continuación se describen las pruebas que se realizaron para cumplir con los objetivos del presente trabajo.

5.2.4.1 Búsqueda de residuos

Para la prueba de búsqueda de residuos se hizo que el robot se moviera de manera aleatoria buscando los residuos en una superficie con radio de 4 metros, donde se colocaron 3 cajetillas de cigarros, 2 envolturas de papas fritas y 3 latas de refresco, las cuales fueron pintadas de color verde, rojo y azul. El objetivo de esta prueba fue que el robot recorriera la superficie buscando residuos y una vez que encontrara uno se colocara frente de él. Decidimos que la mejor manera para buscar los residuos era una búsqueda aleatoria utilizando la función rand () para generar un número aleatorio entre 0 y 2, ya que cuando se le programaron varias rutas específicas, el robot no recorría todo el terreno. La tarjeta controladora ejecutará la tarea de búsqueda mientras espera que la CMUcam3 le mande instrucciones a donde moverse. El programa 4 que se encuentra en el apéndice A con el nombre de main.c, realiza la búsqueda de residuos y la figura 4.7a muestra el diagrama de flujo de la búsqueda. Los resultados que nos manda este algoritmo no son los más óptimos ya que tarda mucho tiempo en recorrer un área de radio de 4 metros, aproximadamente el área la recorre en 15 minutos sin tocar todos sus puntos. Cabe mencionar que el algoritmo de navegación es un punto a mejorar en trabajos futuros.

5.2.4.2 Evasión de obstáculos

Para la prueba de evasión de obstáculos se colocó una maceta que simulara una palmera y una silla que simulara un sillón de playa. El programa 5 del apéndice A tiene una función llamada "`EXT_isr()`" que atiende a la interrupción debida al choque de los bumpers, cuando el robot se encuentra navegando y choca con algún objeto ocurre una interrupción que genera que el robot ejecute la subrutina de interrupción para evadir obstáculos, entonces el robot retrocede y gira 45° a su derecha o a su izquierda dependiendo de la variable control. Esta técnica resultó perfecta para los fines de este proyecto.

5.2.4.3 Recolección de residuos

En la Prueba de recolección de residuos se emplearon objetos como envolturas de papas fritas pintadas de rojo, cajetillas de cigarros pintadas de color verde y latas de refresco pintadas de color azul. Se colocó al robot de tal manera que le quedaran cerca los residuos para que no tardara mucho tiempo buscándolos. A continuación se probó el programa 5 del apéndice A en el modo de búsqueda de residuos y cuando el robot ve un residuo de alguno de los colores rojo, verde y azul se detenga en frente de él y se acomode de tal forma que le quede al alcance del brazo mecánico. Una vez que el robot entra en el modo de recolección de basura este tiene que levantar la basura mediante el uso de su brazo mecánico. Las pruebas hechas fueron satisfactorias por que no se necesito modificarle nada al programa para que recolectara la basura. El único inconveniente del brazo es que no puede recolectar basura que esté muy pegada al piso como ocurre en ocasiones con las envolturas de papas fritas, pero para recoger cajetillas de cigarros y latas lo hace bien.

5.2.4.4 Depósito de residuos

Los elementos necesarios para hacer esta prueba fueron el recipiente con medidas de 20x20x20 cm y una lata de refresco. Se le colocó al robot frente al recipiente para que fuera directo a él y en el momento en que chocaran los dos bumpers inferiores colocados a menos de 20 cm de la estructura del robot y los otros dos colocados aproximadamente a 30 cm no estuvieran en contacto, entonces el robot depositará la basura en el recipiente. En caso de que tanto los bumpers superiores e inferiores chocaran entonces el robot no reconocerá que es el recipiente si no que se trata de un obstáculo. Las pruebas hechas resultaron bien pero con un inconveniente, que el robot tarda mucho tiempo en encontrar el depósito de basura en el caso de una prueba general. En trabajos a futuro se puede mejorar esto dándole movimiento a la cámara CMUcam3 para la búsqueda del depósito de residuos.

5.3 Conclusiones

En general el robot respondió satisfactoriamente a las pruebas hechas con él, en especial en la búsqueda de residuos, recolección de residuos y evasión de obstáculos, ya que fueron las pruebas a las que no se le modificó mucho al código de programación que se tenía desde un principio.

Un problema que se encontró es que el peso del brazo mecánico era mucho para el robot, ya que le costaba trabajo moverse con él, además de hacer sus movimientos mas lentos. En ocasiones, cuando el robot se detenía, el peso del brazo originaba que se cayera el robot, por lo que se colocó una estructura que lo detuviera.

Al realizar las pruebas de calibrado de color, se encontró que los cambios de luz afectan al sistema de visión en gran medida, por lo que se optó en verificar cada vez que se trabajara con el robot, por ajustar los rangos de color que se desean detectar.

Las funciones hechas para el control del robot tienen alta portabilidad, ya que cuando se deseaba modificar alguna función o parámetro no se tenía que cambiar la interfaz de las funciones que se emplearon en el programa principal. De esta manera se pueden exportar los archivos con las definiciones de funciones en otros proyectos que requieran utilizar algún mecanismo similar empleado en este proyecto.

CAPÍTULO VI CONCLUSIONES Y TRABAJO A FUTURO

6.1 Conclusiones

Durante el desarrollo de este trabajo se han generado múltiples conclusiones, que son parte fundamental de los objetivos de este proyecto. A continuación se muestra una recopilación de las conclusiones más importantes.

- Fue necesario modificar las tarjetas de potencia con las que contaba la araña de tal manera que nosotros pudiéramos controlarla, por lo que se retomó su diseño original para hacer nuestras propias tarjetas y poder controlarlas mediante nuestra tarjeta controladora, basada en un microcontrolador PIC16F877A.

- Se pudo comprobar que en general las tarjetas controladoras basadas en el microcontrolador PIC16F877A son excelentes para desarrollar sistemas digitales, por que en ningún momento presentaron problemas debidos al microcontrolador y además cuenta con numerosos puertos con que trabajar.

- La CMUcam3 se adaptó perfectamente a la tarjeta controladora, por lo que observamos que este sistema embebido de visión es excelente para desarrollar sistemas que requieren de visión, además de que no presentó ningún problema al implementarse en el robot.

- Las API's que se utilizan para programar la CMUcam3 son de fácil manejo, gracias a que están escritas en un lenguaje bastante robusto como lo es C, dándonos facilidad para modificar y corregir errores lógicos, permitiendo adecuar los programas según nuestras necesidades.

- Las librerías que se proporcionan para programar PIC's en C son de fácil manejo y aumentan la portabilidad del los programas codificados en C.

- Fue necesario codificar en lenguaje C para PIC's librerías como "utilidades.c", "variables_globales.c", "brazomecanico.c", e "interrupciones.c", para mejorar la portabilidad del programa y también mejorar la legibilidad del código.

- El manejo de servomotores empleando PIC's se facilita en gran medida ya que incluyen funciones específicas para controlar los servomotores.

- Se encontraron problemas con la sincronización de las patas de la araña, al momento de moverla por lo que se requirió modificar varias veces su

sincronía, esto debido a que el sistema ya tenía problemas mecánicos que nos heredaron.

- La araña presentó problemas al moverse en la arena por lo que requirió numerosas composturas de la tarjeta de potencia.

6.2 Trabajo a futuro y recomendaciones

El sistema implementado pretende ser un punto de partida para nuevas investigaciones y desarrollos en el campo de la robótica, como por ejemplo, la programación de robots móviles con visión. La siguiente lista presenta algunas posibles mejoras del proyecto que permitirán el desarrollo de nuevas investigaciones a partir de él.

- El robot puede ser empleado para probar en él algoritmos de navegación.
- Se puede emplear para probar nuevos algoritmos de visión.
- Se promueve el desarrollo de sistemas para UNIX o LINUX.
- El robot puede ser utilizado en lugares donde el ser humano no puede llegar, como por ejemplo en la exploración de ductos.
- Se recomienda emplear un brazo mecánico de menos peso.
- Adaptarle nuevas tarjetas de potencia que puedan mover al robot.
- Emplear un sistema mecánico más estable que el de la araña.
- Mejorar los algoritmos de búsqueda de tal forma que se obtenga algunos más eficientes o que permita reducir el tiempo de ejecución de la tarea encomendada.

APÉNDICE A

Programa 1

Este programa muestra el manejo del puerto serial de la tarjeta, el programa recibe una cadena dada por el usuario mediante una hyperterminal y a continuación el microcontrolador devuelve la misma cadena a la computadora para ser visualizada en la hyperterminal.

```
// Se definen librerias
#include <16f877A.h>
#device ADC=8
#include <stdlib.h>
#include <stdio.h>
// Se define la configuración del PIC16F877A
#fuses HS,NOPROTECT
#use delay(clock=20000000)
#use rs232(baud=19200, xmit=PIN_C6, rcv=PIN_C7)
#org 0x1FFF, 0x1FFF void loader16f877A(void){}


void main(){//Inicio del programa

     char cadena[20];   //Arreglo que almacena la cadena dada por el
usuario

     while(1){//Inicio while

                printf("\n\rCapitulo 2 Programa 1 ");
                printf("\n\rComunicacion Serial   ");
                printf("\n\ringrese una cadena o FIN para finalizar el
programa");
                printf("\n\r--> ");
                gets(cadena);

               if(cadena[0]=='F'&&cadena[1]=='I'&&cadena[2]=='N')
break;
               else   printf("\n\rLa cadena que escribio es \"%s\"
",cadena);


     }//Fin de while

     printf("\n\rFin del programa");

}//Fin del programa
```

Programa 2

Este programa lee el puerto A del microcontrolador al cual se le han colocado cinco bumpers, en el caso de que un contacto se presione se registrará un cero de lo contrario existirá un uno y será enviado a la hyperterminal del usuario.

```
// Se definen librerias
```

```
#include <16f877A.h>
#device ADC=8
#include <stdlib.h>
#include <stdio.h>
// Se define la configuración del PIC16F877
#fuses HS,NOPROTECT
#use delay(clock=20000000)
#use rs232(baud=19200, xmit=PIN_C6, rcv=PIN_C7)
#org 0x1FFF, 0x1FFF void loader16f877A(void){}

int main(){//Inicio del programa

     char cadena[20];//Arreglo que almacena la cadena dada por el
usuario
     int puertoA;    //variable que almacena la lectura del puerto A
     int eleccion=1; //variable de control

     printf("\n\rCapitulo 2 Programa 2 ");
     printf("\n\rLectura de los bumpers por el puerto A\n\r");


     while(eleccion==1){//Inicio while

               printf("\n\r1= Lectura del puerto A\n\r2= Salir");
               printf("\n\r--> ");
                 gets(cadena);
               eleccion=atoi(cadena);

               switch(eleccion){//Inicio de switch
                                case 1: puertoA=input_A();
                                        printf("\n\rEl puerto A
=%x",puertoA);
                                        break;
                                case 2: printf("\n\rFin del programa");
                                         delay_ms(1000);
                                        break;
                                default: printf("\n\rOpcion no
valida");
                                         eleccion=1;
                                         break;
                                }//Fin de switch
     }//Fin de while


}//Fin del programa
```

Programa 3

Este programa muestra un menú para el control de los movimientos del robot, mediante una hyperterminal, de acuerdo a lo descrito en la sección 4.3.4 de este trabajo.

```
#include <16f877A.h>
#device ADC=8
#include <stdlib.h>
#include <stdio.h>
#fuses HS,NOPROTECT
#use delay(clock=20000000)
#use rs232(baud=19200, xmit=PIN_C6, rcv=PIN_C7)
```

```
#org 0x1FFF, 0x1FFF void loader16f877A(void){}
#use fixed_io(d_outputs=PIN_D0 ,PIN_D1,PIN_D2 ,PIN_D3,)

#define ADELANTE   0x09
#define ATRAS      0x06
#define IZQUIERDA  0x05
#define DERECHA    0x0A
#define APAGADO    0x00

void main()
{

 char buffer[10];
 int opcion=0;
 int salida=0;

 set_tris_d(0x00);


 output_d(APAGADO);
 delay_ms(1000);



   while (TRUE){

     printf("\n\n\rDar direccion del robot\n");
           printf("\n\r%14s\n\r%14s\n\r%14s\n\r%14s\n\r%14s\n\r%14s",

                                                          "Adelante  =
      1",
                                                          "Atras     =
2",
                                                          "Izquierda =
3",
                                                          "Derecha   =
4",
                                                          "Stop      =
5",
                                                          "finalizar =
6");
     printf("\n\n\r? ");
     gets(buffer);
     opcion=atoi(buffer);

                          switch(opcion){


                               case 1:
                                        printf("\tadelante");
                                        output_d(APAGADO);
                                        delay_c(n);
                                        output_d(ADELANTE);
                                        break;

                              case 2:
                                        printf("\tatras");
                                        output_d(APAGADO);
                                        delay_c(n);
                                        output_d(ATRAS);
                                        break;
```

```
                                case 3:
                                          printf("\tizquierda");
                                          output_d(APAGADO);
                                          delay_c(n);
                                          output_d(IZQUIERDA);
                                           break;
                                case 4:
                                           printf("\tderecha");
                                           output_d(APAGADO);
                                           delay_c(n);
                                           output_d(DERECHA);
                                           break;
                                           case 5:
                                           printf("\tstop");
                                           output_d(APAGADO);
                                           delay_c(n);
                                           output_d(APAGADO);
                                           break;

                                case 6:
                                           salida=1;
                                           break;


                                default:

                                            printf("\n\rWARNING: opcion
                                            no               valida");
                                            delay_ms(3000);
                                            break;

                                       }
   if (salida==1)break;

   }

   printf("\n\n\rDeteniendo el sistema");
   output_d(APAGADO);

   printf("\n\rFin de la ejecucion");
   printf("\n\r");
   delay_ms(1000);

}
```

Programa 4

Este programa hace una demostración del movimiento del brazo mecánico, rutina que se anexa en el movimiento que tiene que hacer el robot cuando levanta un objeto. Para una mayor portabilidad se dividió el código en dos archivos, el primero con el nombre de “brazomecanico.c” que contiene todas las funciones necesarias para mover el brazo mecánico y el segundo archivo con el nombre de “main.c” que contiene la rutina de movimiento del brazo mecánico.

/*brazomecanico.c*

```
void PWM_GENERATION_SERVO1(long t_servo1){
     int cont;
```

```
      long periodo;

      periodo=20000-t_servo1;

                  for(cont=0;cont<100;cont++){
                  output_high(PIN_E1);
                  delay_us(t_servo1);
                  output_low(PIN_E1);
                  delay_us(periodo);
                  cont++;
                  }

}

void PWM_GENERATION_SERVO2(long t_servo2){
      int cont;
      long periodo;

      periodo=20000-t_servo2;

                  for(cont=0;cont<100;cont++){
                  output_high(PIN_E0);
                  delay_us(t_servo2);
                  output_low(PIN_E0);
                  delay_us(periodo);
                  cont++;
                  }
}


void MOTOR_DC_CONTROL_ABRIR(long t){

       output_high(PIN_C1);
       output_high(PIN_C0);
       delay_ms(t);
       output_low(PIN_C1);


}


void MOTOR_DC_CONTROL_CERRAR(long t_servo1){
       int cont;
       long periodo;
       periodo=20000-t_servo1;
       PWM_GENERATION_SERVO1(t_servo1);

       output_high(PIN_C1);
       output_low(PIN_C0);

                  for(cont=0;cont<100;cont++){//MODIFICACION 1
                  output_high(PIN_E1);
                  delay_us(t_servo1);
                  output_low(PIN_E1);
                  delay_us(periodo);
                  cont++;
                  }
       output_low(PIN_C1);
```

/*main.c*

```
#include <16f877A.h>
#device ADC=8
#include <stdlib.h>
#include <stdio.h>


// Se define la configuración del PIC16F877A
#fuses HS,NOPROTECT
#use delay(clock=20000000)
#use rs232(baud=19200, xmit=PIN_C6, rcv=PIN_C7)
#org 0x1FFF, 0x1FFF void loader16f877A(void){}

#include <brazomecanico.c>

void main(){
     long lh1,lh2,t;
     char buffer[10];
     long cont;
     set_tris_e (0x00);
     set_tris_c (0x80);

     while(TRUE){

                    printf("\n\rIngresar tiempo_H servo 1");
                    printf("\n\r-> ");
                    gets(buffer);
                    lh1=atol(buffer);

                    printf("\n\rIngresar tiempo_H servo 2");
                    printf("\n\r-> ");
                    gets(buffer);
                    lh2=atol(buffer);


                    PWM_GENERATION_SERVO1(lh1);
                    PWM_GENERATION_SERVO2(lh2);
                    MOTOR_DC_CONTROL_ABRIR(500);

                    delay_ms(1000);

                    PWM_GENERATION_SERVO1(lh1);
                    PWM_GENERATION_SERVO2(lh2);
                    MOTOR_DC_CONTROL_CERRAR(1340);
                    delay_ms(1000);
                    }

}
```

Programa 5

Este programa es el programa principal de la tarjeta controladora empleado para este proyecto, las especificaciones de su funcionamiento y diagramas de flujo se encuentran en la sección 4.5 de este trabajo.

/*variables_globales.c*

```
int cont=0;
int tarea=0;
int bote_basura=0; //bote_basura=0 tarea de recolectar bote_basura=1
tarea de depositar
```

/*brazomecanico.c*

```
void PWM_GENERATION_SERVO1(long t_servo1){
     int cont;
     long periodo;

     periodo=20000-t_servo1;

                 for(cont=0;cont<100;cont++){
                 output_high(PIN_E1);
                 delay_us(t_servo1);
                 output_low(PIN_E1);
                 delay_us(periodo);
                 cont++;
                 }

}

void PWM_GENERATION_SERVO2(long t_servo2){
     int cont;
     long periodo;

     periodo=20000-t_servo2;

                 for(cont=0;cont<100;cont++){
                 output_high(PIN_E0);
                 delay_us(t_servo2);
                 output_low(PIN_E0);
                 delay_us(periodo);
                 cont++;
                 }
}


void MOTOR_DC_CONTROL_ABRIR(long t){

      output_high(PIN_C1);
      output_high(PIN_C0);
      delay_ms(t);
      output_low(PIN_C1);


}


void MOTOR_DC_CONTROL_CERRAR(long t_servo1){
      int cont;
      long periodo;
      periodo=20000-t_servo1;
      PWM_GENERATION_SERVO1(t_servo1);

      output_high(PIN_C1);
      output_low(PIN_C0);
```

```
                    for(cont=0;cont<100;cont++){//MODIFICACION 1
                    output_high(PIN_E1);
                    delay_us(t_servo1);
                    output_low(PIN_E1);
                    delay_us(periodo);
                    cont++;
                    }
      output_low(PIN_C1);

}
```

/*utilerias.c*

```
void inicializa_puertos(void){
set_tris_b(0xFF);                   //Entradas de los sensores
set_tris_c(0x80);
set_tris_d(0x00);                   //salidas a los relevadores D3-D0
set_tris_e(0x00);                   //Salidas PWM E0-E1

port_b_pullups(true);               // activa las resistencias de pull-
upset_tris_d(0x00);
ext_int_edge( H_TO_L );             // Sets up EXT de H a bajo ocurre
interrupción


}

void inicializa_interrupciones(void){
enable_interrupts(INT_EXT);  //Habilita interrupción externa PB0
enable_interrupts(INT_RDA); // Habilita interrup. dato recibido en
USART
clear_interrupt(INT_EXT);
set_timer1(0);
setup_timer_1(T1_INTERNAL|T1_DIV_BY_4);
enable_interrupts(INT_TIMER1);
enable_interrupts(GLOBAL);   // Habilita interrup. GLOBAL
}

//delay_largo
void delay_l(long int time){
long int cont1,cont2,cont3;

for(cont1=0;cont1<=500;cont1++)
      for(cont2=0;cont2<=500;cont2++)
            for(cont3=0;cont3<=time;cont3++);

}


//delay_corto
void delay_c(long int time){
long int cont1,cont2;

for(cont1=0;cont1<=1000;cont1++)
      for(cont2=0;cont2<=time;cont2++);
```

```
}
```

/*interrupciones.c*

```
#define delay_ 200

//////////////////////
//INTERRUPCION SERIAL//
//////////////////////


#INT_RDA
void serial_isr()
{
int salida=0;
char temp;
disable_interrupts(GLOBAL);

if (bote_basura==0){
            output_d(APAGADO);
            delay_ms(delay_);

            while(TRUE){
                  switch(getchar()){

                        case '1':
                                                if(bote_basura==1){

      salida=1;

      break;

      }
                                                else{
                                                output_d(IZQUIERDA);
                                                delay_ms(delay_);
                                                output_d(APAGADO);
                                                delay_ms(delay_);
                                                putchar('A');
                                                break;
                                                }

                        case '2':
                                                if(bote_basura==1){

      salida=1;

      break;
                                                                        }
                                                else{
                                                output_d(ADELANTE);
                                                delay_ms(delay_);
                                                output_d(APAGADO);
                                                delay_ms(delay_);
                                                putchar('A');
                                                break;
                                                }
                        case '3':
```

```
                                        if(bote_basura==1){

      salida=1;

      break;
                                                                }
                                        else{
                                        output_d(DERECHA);
                                        delay_ms(delay_);
                                        output_d(APAGADO);
                                        delay_ms(delay_);
                                        putchar('A');
                                        break;
                                        }
                        case '4':

                                        output_d(APAGADO);
                                        delay_ms(delay_);
                                        salida=1;
                                        break;

                        case '5':

                                        output_d(APAGADO);
                                        delay_ms(delay_);
                                        output_d(DERECHA);
                                        delay_ms(3690);
                                        output_d(APAGADO);
                                        delay_ms(delay_);
                                        putchar('A');
                                        enable_interrupts(GLOBAL);
                                        return;
                        case '6':

                                        if (bote_basura==1){
                                        output_d(APAGADO);
                                        delay_ms(delay_);
                                        output_d(ADELANTE);
                                        delay_ms(1000);
                                        output_d(APAGADO);
                                        delay_ms(delay_);
                                        putchar('A');
                                        enable_interrupts(GLOBAL);
                                        return;}
                                        else return;

                        }//FIN DE SWITCH
              if (salida==1) break;
              }//FIN DE WHILE TRUE

//////////////////7
//    BRAZO MECANICO/7
//////////////////7


      PWM_GENERATION_SERVO1(1850);  //subir brazo
      PWM_GENERATION_SERVO2(2350);  //subir muñeca
      MOTOR_DC_CONTROL_ABRIR(500);  //abrir pinza

      delay_ms(1000);
```

```
      PWM_GENERATION_SERVO1(1340);  //bajar brazo
      PWM_GENERATION_SERVO2(200);         //bajar muñeca
      MOTOR_DC_CONTROL_CERRAR(1340);      //tomar objeto
      delay_ms(1000);

      PWM_GENERATION_SERVO1(1850);  //subir brazo
      PWM_GENERATION_SERVO2(2350);  //subir muñeca
      //MOTOR_DC_CONTROL_ABRIR(1000);     //abrir pinza

      delay_ms(1000);
      putchar('A');
      bote_basura=1;

else {
      temp=getchar();

      if (temp=='5'){

                output_d(APAGADO);
                        delay_ms(delay_);
                        output_d(DERECHA);
                        delay_ms(3690);
                        output_d(APAGADO);
                        delay_ms(delay_);
                        putchar('A');
                          }
      else {putchar('A');}
      }//código para no salir del área negra
enable_interrupts(GLOBAL);

}

//////////////////////
//INTERRUPCION EXT   //
//////////////////////

#int_EXT
void EXT_isr()
{
static int control=0;


delay_ms(100);
if (input(PIN_B0)) return;

else{//ELSE PRINCIPAL
delay_ms(400);
if ((bote_basura==1)&& !input(PIN_B0) && input(PIN_B1)){//DEPOSITAR EN
EL BOTE

            //printf("\n\rDEPOSITANDO EN EL BOTE");

            output_d(APAGADO);

            delay_ms(500);

            //PWM_GENERATION_SERVO1(1340);      //BAJAR BRAZO

            PWM_GENERATION_SERVO2(800);   //BAJAR MUÑECA
```

```
            MOTOR_DC_CONTROL_DEPOSITAR(1340);

            delay_ms(1000);

            //PWM_GENERATION_SERVO1(1850);      //subir brazo

            PWM_GENERATION_SERVO2(2350);  //subir muñeca

            //MOTOR_DC_CONTROL_ABRIR(1000);     //abrir pinza


            output_d(ATRAS);

            delay_ms(3000);

            output_d(APAGADO);

            delay_ms(500);

            output_d(DERECHA);

            delay_ms(3000);

            output_d(APAGADO);

            delay_ms(500);

            bote_basura=0;

            putchar('A'); //SEÑAL QUE INDICA EL INICIO DE UN NUEVO
CICLO

            }
else{//ESQUIVAR OBSTÁCULO

      output_d(APAGADO);

      delay_ms(500);
      output_d(ATRAS);

      delay_ms(2000);
      output_d(APAGADO);

      delay_ms(500);

                        if (control==0){
                        output_d(DERECHA);
                        control++;
                        }

                        else {
                        output_d(IZQUIERDA);
                        control=0;
                        }

      delay_ms(1845);
      output_d(APAGADO);
      delay_ms(500);
      }
```

```
      }//FIN DE ELSE PRINCIPAL
return;
}

//////////////////////
//INTERRUPCION TIMER1//
//////////////////////
#INT_TIMER1
void timer1(){
static long cont1=0;
int temp1;
int temp2;

if (cont1==25){


   temp1=input_d();
   temp2=temp1;
   temp2&=0x03;
   output_d(temp2);
   delay_ms(100);
   output_d(temp1);
   delay_ms(100);
   cont1=0;
   }

else cont1++;

}
```

/*main.c*

```
#include <16f877A.h>
#device ADC=8
#include <stdlib.h>
#include <stdio.h>
// Se define la configuración del PIC16F877
#fuses HS,NOPROTECT
#use delay(clock=20000000)
#use rs232(baud=19200, xmit=PIN_C6, rcv=PIN_C7)
#org 0x1FFF, 0x1FFF void loader16f877A(void){}
#use fixed_io(d_outputs=PIN_D0 ,PIN_D1,PIN_D2 ,PIN_D3,)

#define ADELANTE  0x09
#define ATRAS          0x06
#define IZQUIERDA      0x05
#define DERECHA   0x0A
#define     APAGADO     0x00

#include <variables_globales.c>
#include <brazomecanico.c>
#include <utilerias.c>
#include <interrupciones.c>

void main(){

unsigned int ruta;
inicializa_puertos();
```

```
inicializa_interrupciones();


output_d(APAGADO);
delay_ms(1000);

//navegación

while(tarea==0){

output_d(ADELANTE);
delay_l(0);

output_d(ADELANTE);
delay_l(0);

      ruta=rand()%3;

switch(ruta){
                  case 0:

                              output_d(ADELANTE);
                              delay_l(0);
                              break;
                  case 1:


                              output_d(IZQUIERDA);
                              break;
                  case 2:

                              output_d(DERECHA);
                              break;

                  }

                  delay_l(0);

                  delay_c(200);



                        }
}
```

Programa 6

Este programa permitió encontrar los comandos que mueven al robot en el sentido correcto. Sus detalles se encuentran en la sección 5.2.1.2 de este trabajo.

```
// Se definen librerias
#include <16f877A.h>
#device ADC=8
#include <stdlib.h>
#include <stdio.h>
// Se define la configuración del PIC16F877A
```

```
#fuses HS,NOPROTECT
#use delay(clock=20000000)
#use rs232(baud=19200, xmit=PIN_C6, rcv=PIN_C7)
#org 0x1FFF, 0x1FFF void loader16f877A(void){}

void main(){
     printf("\n\rPrograma para encontrar las secuencias de");
     printf("\n\rmovimiento del robot");

     while(TRUE){

                 output_d(0x0A);
                 delay_ms(2000);

                 output_d(0x09);
                 delay_ms(2000);

                 output_d(0x06);
                 delay_ms(2000);

                 output_d(0x05);
                 delay_ms(2000);

                 output_d(0x00);
                 delay_ms(2000);


              }


}
```

Programa 7

Este programa fue empleado en la prueba de comunicación con la CMUcam3 en la sección 5.2.3.1.

```
#include <16f877A.h>
#device ADC=8
#include <stdlib.h>
#include <stdio.h>
// Se define la configuración del PIC16F877
#fuses HS,NOPROTECT
#use delay(clock=20000000)
#use rs232(baud=19200, xmit=PIN_C6, rcv=PIN_C7)
#org 0x1FFF, 0x1FFF void loader16f877A(void){}


#define ADELANTE   0x09
#define ATRAS      0x06
#define IZQUIERDA  0x05
#define DERECHA    0x0A
#define APAGADO    0x00


void main(){

inicializa_puertos();
```

```
output_d(APAGADO);
delay_ms(1000);

while(TRUE){
                  switch(getchar()){

                        case '1':         output_d(ADELANTE);
                                          delay_ms(2000);
                                          break;

                        case '2':
                                          output_d(ATRAS);
                                          delay_ms(2000);
                                          break;


                        case '3':
                                          output_d(IZQUIERDA);
                                          delay_ms(2000);
                                          break;

                        case '4':         output_d(DERECHA);
                                          delay_ms(2000);
                                          break;

                                          break;

                        case '5':
                                          output_d(APAGADO);
                                          delay_ms(2000);
                                          break;


                        }//FIN DE SWITCH
}
```

APÉNDICE B

Programa 1

Este programa imprime un mensaje de bienvenida y pide al usuario que se comunica mediante una hyperterminal, ingresar una cadena, enseguida la CMUcam3 regresa la cadena como la escribió el usuario

```
#include <stdio.h>

//#include <cc3.h>

int main(void) { //inicio del programa

    char buffer[10];

    //configuración de la comunicación serial de la CMUcam3
    cc3_uart_init (0,
                CC3_UART_RATE_115200,
                CC3_UART_MODE_8N1,
                CC3_UART_BINMODE_TEXT);

    while(1) {
                printf("\n\rBienvenido al programa de comunicacion
serial ");
                printf("\n\rIngrese una cadena  o FIN para salir
");
                printf("\n\r-> ");

                gets(buffer);

                if (buffer[0]=='F'&&buffer[1]=='I'&&buffer[2]=='N')
break;
                else printf("\n\rLa cadena que ingreso fue: \n\r
%s",buffer);
    }

    printf("\n\rFin de la ejecucion");

}//Fin del programa
```

Programa 2

Este programa permite que la CMUcam3 detecte el color rojo intenso, el programa obtiene el centroide de una región de color rojo intenso, su envolvente y el número de pixeles que encontró con ese tono de color, para después enviarlos por el puerto serial y mostrarlo por la hyperterminal.

```
#include <stdio.h>
#include <stdlib.h>
#include <cc3.h>
#include <cc3_ilp.h>
#include <cc3_color_track.h>
```

```
void simple_track_color(cc3_track_pkt_t* t_pkt);

int main(void) {
  cc3_track_pkt_t t_pkt;


  cc3_uart_init (0,
                 CC3_UART_RATE_115200,
                 CC3_UART_MODE_8N1,
                 CC3_UART_BINMODE_TEXT);

  cc3_camera_init ();
  cc3_camera_set_colorspace(CC3_COLORSPACE_RGB);
  cc3_camera_set_resolution(CC3_CAMERA_RESOLUTION_LOW);


  // init pixbuf with width and height
  cc3_pixbuf_load();

  // parámetros para elección del color a buscar
  t_pkt.lower_bound.channel[CC3_CHANNEL_RED]   = 255;
  t_pkt.upper_bound.channel[CC3_CHANNEL_RED]   = 150;
  t_pkt.lower_bound.channel[CC3_CHANNEL_GREEN] = 0;
  t_pkt.upper_bound.channel[CC3_CHANNEL_GREEN] = 50;
  t_pkt.lower_bound.channel[CC3_CHANNEL_BLUE]  = 0;
  t_pkt.upper_bound.channel[CC3_CHANNEL_BLUE]  = 50;
  t_pkt.noise_filter = 2;


  while(true) {

    simple_track_color(&t_pkt);
    printf( "centroid = %d,%d bounding box = %d,%d,%d,%d num pix= %d
             density = %d\n",

                    t_pkt.centroid_x, t_pkt.centroid_y,
                    t_pkt.x0,t_pkt.y0,t_pkt.x1,t_pkt.y1);

   }

}


void simple_track_color(cc3_track_pkt_t * t_pkt)
{
  cc3_image_t img;

  img.channels = 3;
  img.width = cc3_g_pixbuf_frame.width;
  img.height = 1;
  img.pix = cc3_malloc_rows (1);
  if (img.pix == NULL) {
    return;
  }

    cc3_pixbuf_load ();
    if (cc3_track_color_scanline_start (t_pkt) != 0) {
      while (cc3_pixbuf_read_rows (img.pix, 1)) {

           cc3_track_color_scanline (&img, t_pkt);
```

```
          }
        }
    cc3_track_color_scanline_finish (t_pkt);

  free (img.pix);
  return;
}
```

Programa 3

Este programa es el programa principal que se descargo en la CMUcam3 para la detección de los colores rojo, verde y azul, mandando la codificación del centroide de la figura por el puerto serial a la tarjeta controladora.

```
#include <stdio.h>
#include <stdlib.h>
#include <cc3.h>
#include <cc3_ilp.h>
#include <cc3_color_track.h>

#define  SuperiorIzquierdo putchar('1'); //GIRO DERECHA
#define  Superior          putchar('2'); //ADELANTE
#define  SuperiorDerecho   putchar('3'); //GIRO IZQUIERDA
#define  CentroIzquierdo   putchar('1');
#define  Centro            putchar('4'); //STOP
#define  CentroDerecho     putchar('3');
#define  InferiorIzquierdo putchar('4');
#define  Inferior          putchar('4');
#define  InferiorDerecho   putchar('4');
#define  Negro             putchar('5');

void simple_track_color_2(cc3_track_pkt_t* t_pkt1,cc3_track_pkt_t*
t_pkt2,cc3_track_pkt_t* t_pkt3,cc3_track_pkt_t* t_pkt4);
void semaforo(void);

int main(void) {
  cc3_track_pkt_t t_pkt1;
  cc3_track_pkt_t t_pkt2;
  cc3_track_pkt_t t_pkt3;
  cc3_track_pkt_t t_pkt4;

  cc3_uart_init (0,
                 CC3_UART_RATE_19200,
                 CC3_UART_MODE_8N1,
                 CC3_UART_BINMODE_TEXT);

  cc3_camera_init ();

  cc3_camera_set_resolution(CC3_CAMERA_RESOLUTION_LOW);

  cc3_pixbuf_load();


  // Track Red
```

```
  t_pkt1.lower_bound.channel[CC3_CHANNEL_RED] =120;
  t_pkt1.upper_bound.channel[CC3_CHANNEL_RED] = 150;
  t_pkt1.lower_bound.channel[CC3_CHANNEL_GREEN] =105;
  t_pkt1.upper_bound.channel[CC3_CHANNEL_GREEN] =135;
  t_pkt1.lower_bound.channel[CC3_CHANNEL_BLUE] =70;
  t_pkt1.upper_bound.channel[CC3_CHANNEL_BLUE] =99;
  t_pkt1.noise_filter = 2;

  // Track Green
  t_pkt2.lower_bound.channel[CC3_CHANNEL_RED] = 90;
  t_pkt2.upper_bound.channel[CC3_CHANNEL_RED] = 120;
  t_pkt2.lower_bound.channel[CC3_CHANNEL_GREEN] = 100;
  t_pkt2.upper_bound.channel[CC3_CHANNEL_GREEN] = 130;
  t_pkt2.lower_bound.channel[CC3_CHANNEL_BLUE] = 54;
  t_pkt2.upper_bound.channel[CC3_CHANNEL_BLUE] = 75;
  t_pkt2.noise_filter = 2;

  // Track blue
  t_pkt3.lower_bound.channel[CC3_CHANNEL_RED]   = 40;
  t_pkt3.upper_bound.channel[CC3_CHANNEL_RED]   = 66;
  t_pkt3.lower_bound.channel[CC3_CHANNEL_GREEN] = 30;
  t_pkt3.upper_bound.channel[CC3_CHANNEL_GREEN] = 68;
  t_pkt3.lower_bound.channel[CC3_CHANNEL_BLUE]  = 30;
  t_pkt3.upper_bound.channel[CC3_CHANNEL_BLUE]  = 58;
  t_pkt3.noise_filter = 2;

  // Track NEGRO
  t_pkt4.lower_bound.channel[CC3_CHANNEL_RED] = 0;
  t_pkt4.upper_bound.channel[CC3_CHANNEL_RED] = 0;
  t_pkt4.lower_bound.channel[CC3_CHANNEL_GREEN] =0;
  t_pkt4.upper_bound.channel[CC3_CHANNEL_GREEN] = 0;
  t_pkt4.lower_bound.channel[CC3_CHANNEL_BLUE] = 0;
  t_pkt4.upper_bound.channel[CC3_CHANNEL_BLUE] = 0;
  t_pkt4.noise_filter = 2;

  // Track bote


  cc3_timer_wait_ms(1000);
  while(true) {
              simple_track_color_2(&t_pkt1,&t_pkt2,&t_pkt3,&t_pkt4);

//control del LÍMITE
          if(t_pkt4.centroid_y>0 && t_pkt4.centroid_x>0)
                                    {
                                    Negro
                                    semaforo();
                                    continue;
                                    }

//control del rojo
                    //1ER RENGLON
          if(t_pkt1.centroid_y>0 && t_pkt1.centroid_y<=47 )
                                    {
                                     if(t_pkt1.centroid_x>0 &&
t_pkt1.centroid_x<=58)          {SuperiorIzquierdo semaforo();
continue;}
                                      else if(t_pkt1.centroid_x>58  &&
t_pkt1.centroid_x<=116 ) {Superior          semaforo(); continue;}
                                    else if(t_pkt1.centroid_x>116)
{SuperiorDerecho   semaforo(); continue;}
```

```
                                    }
                    //2DO RENGLON
          if(t_pkt1.centroid_y>47 && t_pkt1.centroid_y<=94)
                                   {
                                    if(t_pkt1.centroid_x<=58)
{CentroIzquierdo   semaforo(); continue;}
                                    else if(t_pkt1.centroid_x>58  &&
t_pkt1.centroid_x<=116 ) {Centro            semaforo(); continue;}
                                    else if(t_pkt1.centroid_x>116)
{CentroDerecho     semaforo(); continue;}
                                    }
                    //3ER RENGLON
          if(t_pkt1.centroid_y>94)
                                   {
                                    if(t_pkt1.centroid_x<=58)
{InferiorIzquierdo semaforo(); continue;}
                                    else if(t_pkt1.centroid_x>58  &&
t_pkt1.centroid_x<=116 ) {Inferior           semaforo(); continue;}
                                    else if(t_pkt1.centroid_x>116 &&
t_pkt1.centroid_x<=174)  {InferiorDerecho    semaforo(); continue;}
                                    }


//control del verde
                    //1ER RENGLON
          if(t_pkt2.centroid_y>0 && t_pkt2.centroid_y<=47)
                                   {
                                    if(t_pkt2.centroid_x>0 &&
t_pkt2.centroid_x<=58)          {SuperiorIzquierdo  semaforo();
continue;}
                                    else if(t_pkt2.centroid_x>58  &&
t_pkt2.centroid_x<=116 ) {Superior           semaforo(); continue;}
                                    else if(t_pkt2.centroid_x>116)
{SuperiorDerecho     semaforo(); continue;}
                                    }
                    //2DO RENGLON
          if(t_pkt2.centroid_y>47 && t_pkt2.centroid_y<=94)
                                   {
                                    if(t_pkt2.centroid_x<=58)
{CentroIzquierdo     semaforo(); continue;}
                                    else if(t_pkt2.centroid_x>58  &&
t_pkt2.centroid_x<=116 ) {Centro             semaforo(); continue;}
                                    else if(t_pkt2.centroid_x>116)
{CentroDerecho        semaforo(); continue;}
                                    }
                    //3ER RENGLON
          if(t_pkt2.centroid_y>94)
                                   {
                                    if(t_pkt2.centroid_x<=58)
{InferiorIzquierdo     semaforo(); continue;}
                                    else if(t_pkt2.centroid_x>58  &&
t_pkt2.centroid_x<=116 ) {Inferior              semaforo(); continue;}
                                    else if(t_pkt2.centroid_x>116 &&
t_pkt2.centroid_x<=174)  {InferiorDerecho       semaforo(); continue;}
                                    }

//control del azul
                    //1ER RENGLON
          if(t_pkt3.centroid_y>0 && t_pkt3.centroid_y<=47)
                                   {
                                    if(t_pkt3.centroid_x>0 &&
```

```
t_pkt3.centroid_x<=58)          {SuperiorIzquierdo     semaforo();
continue;}
                                    else if(t_pkt3.centroid_x>58  &&
t_pkt3.centroid_x<=116 ) {Superior              semaforo(); continue;}
                                    else if(t_pkt3.centroid_x>116)
{SuperiorDerecho       semaforo(); continue;}
                                    }
                  //2DO RENGLON
          if(t_pkt3.centroid_y>47 && t_pkt3.centroid_y<=94)
                                   {
                                    if(t_pkt3.centroid_x<=58)
{CentroIzquierdo       semaforo(); continue;}
                                    else if(t_pkt3.centroid_x>58  &&
t_pkt3.centroid_x<=116 ) {Centro                semaforo(); continue;}
                                    else if(t_pkt3.centroid_x>116)
{CentroDerecho         semaforo(); continue;}
                                    }
                  //3ER RENGLON
          if(t_pkt3.centroid_y>94)
                                   {
                                    if(t_pkt3.centroid_x<=58)
{InferiorIzquierdo     semaforo(); continue;}
                                    else if(t_pkt3.centroid_x>58  &&
t_pkt3.centroid_x<=116 ) {Inferior              semaforo(); continue;}
                                    else if(t_pkt3.centroid_x>116 &&
t_pkt3.centroid_x<=174)  {InferiorDerecho       semaforo(); continue;}
                                    }

   }//FIN DE WHILE(TRUE)

}//FIN DE MAIN


void simple_track_color_2(cc3_track_pkt_t * t_pkt,cc3_track_pkt_t *
t_pkt2,cc3_track_pkt_t * t_pkt3,cc3_track_pkt_t * t_pkt4)
{
  cc3_image_t img;

  img.channels = 3;
  img.width = cc3_g_pixbuf_frame.width;
  img.height = 1;
  // image will hold just 1 row for scanline processing
  img.pix = cc3_malloc_rows (1);
  if (img.pix == NULL) {
    return;
  }

    cc3_pixbuf_load ();
    if (cc3_track_color_scanline_start (t_pkt) != 0 &&
cc3_track_color_scanline_start (t_pkt2) != 0
                                                    &&
cc3_track_color_scanline_start (t_pkt3) != 0
                                                    &&
cc3_track_color_scanline_start (t_pkt4) != 0) {
      while (cc3_pixbuf_read_rows (img.pix, 1)) {
           // This does the HSV conversion
           // cc3_rgb2hsv_row(img.pix,img.width);
           // Use the same scanline twice since it is already in
memory and hence quick to operate upon
           cc3_track_color_scanline (&img, t_pkt);
           cc3_track_color_scanline (&img, t_pkt2);
```

```
            cc3_track_color_scanline (&img, t_pkt3);
            cc3_track_color_scanline (&img, t_pkt4);
           }
         }
    cc3_track_color_scanline_finish (t_pkt);
    cc3_track_color_scanline_finish (t_pkt2);
    cc3_track_color_scanline_finish (t_pkt3);
    cc3_track_color_scanline_finish (t_pkt4);

  free (img.pix);
  return;

}


void semaforo(){
    //printf("\n\resperando a que termine el PIC");
    while(true){
                if (getchar()=='A')
                break;
                }

    }
```

Programa 4

Este programa fue empleado en la prueba de comunicación con la tarjeta controladora en la sección 5.2.3.1.

```
#include <stdio.h>
#include <stdlib.h>
#include <cc3.h>
#include <cc3_ilp.h>
#include <cc3_color_track.h>


int main(void) {

  cc3_uart_init (0,
                 CC3_UART_RATE_19200,
                 CC3_UART_MODE_8N1,
                 CC3_UART_BINMODE_TEXT);


  cc3_timer_wait_ms(3000);

  while(true) {
              putchar('1');
              cc3_timer_wait_ms(4000);

              putchar('2');
              cc3_timer_wait_ms(4000);

              putchar('3');
              cc3_timer_wait_ms(4000);
```

```
                putchar('4');
                cc3_timer_wait_ms(4000);

                putchar('5');
                cc3_timer_wait_ms(4000);


    }//FIN DE WHILE(TRUE)

}//FIN DE MAIN
```

Bibliografía y Referencias

- **[1] http://www.microchip.com/**
- **[2] http://www.cmucam.org/**
- **[3] Embedded Robotics Thomas Bräunl. Ed Springer.**
- **[4] CCS C Compiler help file.**
- **[5] PIC16F877A Data Sheet.**
- **[6] http://www.arm.com/products/CPUs/ARM7TDMI.html**
- **[7] Datasheet LPC2106.**
- **[8] http://www.ovt.com/**
- **[9] http://www.codesourcery.com/gnu_toolchains/arm/**
- **[10] http://www.ccsinfo.com/**
- **[11] CMUcam3 datasheet.**
- **[12] CMUcam3 sdk guide installation.**
- **[13] Programación en Linux con ejemplos Kurt Wall.**
- **[14] http://www.cygwin.com/**
- **[15] http://www.cadsoftusa.com/ eagle**
- **[16] http://www.microchipc.com/PIC16bootload/index.php**
- **[17] Datasheet ULN2003.**
- **[18] www.tycorc.com**
- **[19] Diseño de Microprocesadores. Jesús Savage. Facultad de Ingeniería. UNAM. Página 11.**
-
- **[20] Microcontrolador PIC 16F84. Desarrollo de proyectos. 2da Edición. Enrique Palacios, Fernando Remiro, Lucas J. López. Editorial Alfaomega.**