Escola Universitària d'Enginyeria
Tècnica Industrial de Barcelona
Consorci Escola Industrial de Barcelona
UNIVERSITAT POLITÈCNICA DE CATALUNYA

Memoria

"DISEÑO Y CONSTRUCCIÓN DE UN ROBOT HUMANOIDE"

PFC presentado para optar al título de Ingeniería Técnica Industrial especialidad en Electrónica Industrial

por **Marc Bonell Sanchez**

Barcelona, 15 de Junio de 2011

Director: Herminio Martínez García
Departamento de Ingeniería Electrónica (710)
Universitat Politècnica de Catalunya (UPC)

ÍNDICE

ÍNDICE DE LA MEMORIA ECONÓMICA

ÍNDICE ANEXOS

RESUMEN

En este proyecto se diseña y construye tanto el circuito de control como la parte mecánica de un robot humanoide. La peculiaridad de este es que el control de los servomotores no se realiza mediante un módulo controlador comercial sino que se crea uno basándose en un proceso de análisis para obtener los mejores resultados posibles.

Podremos ver también los distintos tipos de componentes que lo integran y conocer su funcionamiento a rasgos generales junto con los planos y características de las piezas mecánicas.

Por otra parte se pretende dar a conocer los distintos tipos de robots que existen en la actualidad y los avances que se han ido creando a lo largo de la historia en cuanto al movimiento de robots humanoides.

El resultado final será un robot humanoide capaz de poder mover todos sus servomotores mediante el controlador creado y capaz de mantener mínimamente el equilibrio.

RESUM

En aquest projecte es dissenya i construeix tant el circuit de control com la part mecànica d'un robot humanoide. La peculiaritat d'aquest és que el control dels servomotors no es realitza mitjançant un mòdul controlador comercial sinó que es crea un basant-se en un procés d'anàlisi per obtenir els millors resultats possibles.

Podrem veure també els diferents tipus de components que l'integren i conèixer el seu funcionament a trets generals juntament amb els plànols i característiques de les peces mecàniques.

D'altra banda es pretén donar a conèixer els diferents tipus de robots que existeixen en l'actualitat i els avenços que s'han anat creant al llarg de la història pel que fa al moviment de robots humanoides.

El resultat final serà un robot humanoide capaç de poder moure tots els seus servomotors mitjançant el controlador creat i capaç de mantenir mínimament l'equilibri.

ABSTRACT

In this project is designed and built both the control circuit and the mechanical part of a humanoid robot. The peculiarity of this is that the control of servos is not performed by a controller module business; it's created a basis of an analysis process to obtain the best possible results.

We also see the different types of components that compose it and understand how it works with the general outlines of plans and features of the mechanical parts.

On the other hand, it tries to present different types of robots that exist today and the advances that have been created throughout history as the motion of humanoid robots.

The end result will be a humanoid robot able to move all servos through the servo controller, and be able to minimally maintain the balance.

AGRADECIMIENTOS

Especial agradecimiento a mi familia y mi pareja por tener tanta paciencia en los momentos de frustración y haberme apoyado en todo momento frente las diversas dificultades que me he ido encontrado a lo largo del proyecto, en especial mi padre que gracias a su gran experiencia en el mundo de la electrónica me ha enseñado muchos trucos que solo se aprenden a base de horas de trabajo.

Por otra parte, mencionar el equipo de IE3P que desinteresadamente me han dado todo su apoyo y todos sus conocimientos en la materia y, a más a más, por todo el material prestado para realizar pruebas.

Un enorme placer haber cursado esta carrera, aportándome una gran cantidad de conocimiento en el aspecto electrónico como en el aspecto personal y en relación al trato con compañeros de trabajo.

Gracias.

CAPÍTULO 1: INTRODUCCIÓN

Hoy en día, el avance de la robótica se ha convertido en un gran aporte al desarrollo de la sociedad en campos diferentes tales como: la medicina, la industria, el entretenimiento, etc.

El hombre constantemente ha buscado desarrollar robots cada vez con mayor capacidad de procesamiento de datos para poder interactuar con el entorno y sobre todo tratando de imitar la anatomía humana, de donde nacen los llamados humanoides. Una labor muy fuerte de aproximadamente más de 30 años de investigación, ha logrado desarrollar a humanoides, obteniendo buenos resultados gracias a la aplicación de algoritmos inteligentes, conocimiento de nuevos materiales, etc. Y es así, que el mayor referente en este desarrollo es el robot ASIMO diseñado por HONDA. Para poder lograr el desarrollo de los humanoides, se toma como idea fundamental el conocer más sobre la locomoción bípeda humana, ya que este tipo de robots al poseer dos extremidades inferiores (de ahí su nombre bípedos), basan su locomoción en el caminar y anatomía de los miembros inferiores de los humanos. Todo esto ha llevado a plantear el presente trabajo, el cual tiene como objetivo principal controlar y ensamblar una plataforma robótica bípeda, permitiendo que dicha plataforma realice varias rutinas de movimientos tales como: caminar hacia adelante o atrás, girar a la izquierda o derecha, patear y levantarse de forma análoga a los movimientos comunes que realizan los seres humanos.

CAPÍTULO 2: ROBÓTICA MÓVIL

El avance de la robótica se ha convertido en un gran aporte al desarrollo de la sociedad, por ello en el presente capítulo se tratarán conceptos básicos referentes a este campo de estudio, el desarrollo de los robots bípedos y otros temas relacionados con esta temática.

La necesidad de que los robots puedan realizar tareas por sí mismos, de forma autónoma, hace necesario que éstos tomen decisiones acordes con relación a su entorno: "en esto, tiene mucho que ver los aspectos principales que constituyen la base de la robótica móvil: la percepción, visión, navegación, planeación, constr ucción de mapas, localización y, por supuesto la interacción humano-robot".

2.1 Tipos de robots móviles

Los robots móviles se pueden clasificar de acuerdo a su modo de desplazamiento en: robots con ruedas, con orugas, configuraciones articuladas y robots con locomoción mediante patas.

2.1.1 Robots móviles con ruedas

Este tipo de robots se caracterizan por ser la solución más simple y eficiente de movilidad en terrenos suficientemente duros, en donde se pueden conseguir velocidades relativamente altas. Como desventaja de este tipo de robots, se tiene que, pueden patinar en cierto tipo de terrenos; además, el desplazamiento mediante ruedas no es eficiente en terrenos blandos o irregulares.

A continuación, se presenta una breve explicación de las características de los sistemas de desplazamiento más comunes en robots móviles con ruedas:

Ackerman

Es el tipo usado por ejemplo en vehículos de 4 ruedas convencionales tales como automóviles. Este sistema de locomoción se ilustra en la Figura 1. La rueda delantera interior gira un ángulo superior al exterior (θ1>θ0) para eliminar el deslizamiento. Las prolongaciones de los ejes de las dos ruedas delanteras intersecan en un punto sobre la prolongación del eje de las ruedas traseras. El lugar de los puntos trazados sobre el suelo por los centros de los neumáticos son circunferencias concéntricas con centro el eje de rotación P1.

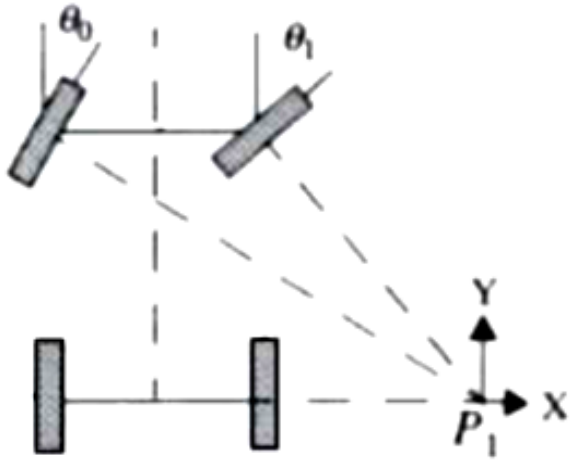


Fig. 1. Locomoción a 4 ruedas.

Triciclo

La rueda delantera se utiliza tanto para el direccionamiento como para la tracción. El eje posterior con las dos ruedas laterales se desplaza libremente. La movilidad resulta más eficiente en este sistema comparado con el anterior, pero puede presentar inestabilidad en terrenos irregulares. EL centro de gravedad tiende a desplazarse cuando el vehículo se desplaza por una pendiente, causando la pérdida de tracción. En la siguiente figura se observa este sistema.

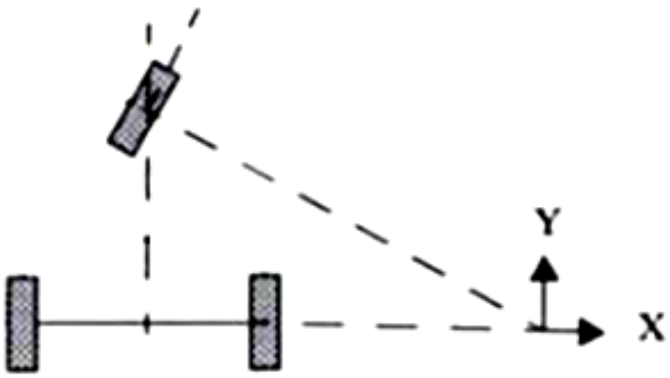


Fig. 2. Locomoción a 3 ruedas.

Direccionamiento Diferencial

Este tipo de sistema se caracteriza por la diferencia de velocidades entre las ruedas laterales, que a su vez son las que generan tracción, además no existe una o más ruedas de soporte como se muestra en la siguiente. Este tipo es más usado en robots para interiores.

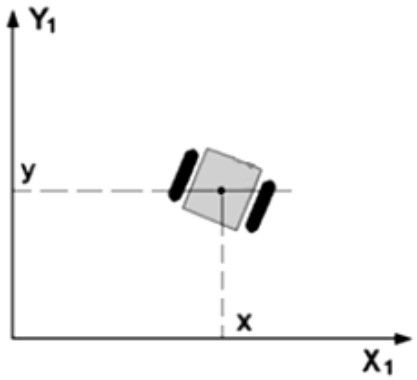


Fig. 3. Locomoción diferencial.

Skid Steer

En este tipo de locomoción aparecen varias ruedas en cada lado del vehículo, las cuales actúan simultáneamente. El desplazamiento es el resultado de combinar la velocidad tanto de la rueda de la izquierda como de la derecha. En la figura siguiente se muestra el "Terregator" un vehículo desarrollado en el Robotic Institute de la Carnegie Mellon University, para aplicaciones en exteriores tales como la minería.

Fig. 4. Locomoción a varias ruedas.

Pistas de Deslizamiento

Se trata de robots tipo oruga en los que tanto la impulsión como el direccionamiento son realizados por sus pistas de deslizamiento, su funcionamiento es análogo al skid steer. De manera más precisa, las pistas actúan análogamente a las ruedas de gran diámetro. Su locomoción es eficiente en terrenos irregulares. En la figura 5 se muestra un robot llamado AURIGA que utiliza este sistema de locomoción, el cual fue desarrollado en el laboratorio del Instituto Andaluz de Automática y Robótica, de la Universidad de Málaga.

Fig. 5. Locomoción tipo oruga.

2.1.2 Locomoción mediante patas

Algunas veces se ha requerido un tipo diferente de movilidad a la que prestan los robots de locomoción por ruedas, los investigadores han desarrollado prototipos imitando distintas formas de desplazamiento, similares a las de los animales y al hombre. Por estas características existen varias aplicaciones, gracias a su adaptabilidad para desplazarse en terrenos irregulares. Para este tipo de locomoción se deben tomar en cuenta algunos aspectos, tales como: posición, velocidad y equilibrio, usando únicamente el movimiento de las articulaciones mediante motores. El diseño de este tipo de robots presenta ciertas dificultades debido a su gran número de grados de libertad, mientras que el algoritmo de control presenta cierta complejidad debido al gran número de movimientos a coordinar. De acuerdo al número de patas, este tipo de robots adquiere su denominación, como por ejemplo, los robots de dos patas se denominan bípedos, de cuatro patas cuadrúpedos, de seis patas hexápodos, etc. En la Figura 6 se muestran dos ejemplos de este tipo de robots.

Fig. 6. Locomoción a patas.

2.1.3 Configuraciones articuladas

Las configuraciones articuladas son de interés para terrenos difíciles a los que debe adaptarse el cuerpo del robot. La solución más simple consiste en articular dos o más módulos de locomoción mediante ruedas; las configuraciones articuladas con gran cantidad de eslabones son apropiadas para caminos estrechos. Este tipo de robots, por su configuración, pueden adoptar la forma de la superficie donde se desplazan. En la figura 7 se presenta un robot con las características antes mencionadas llamado Robot Gusano (Blender).

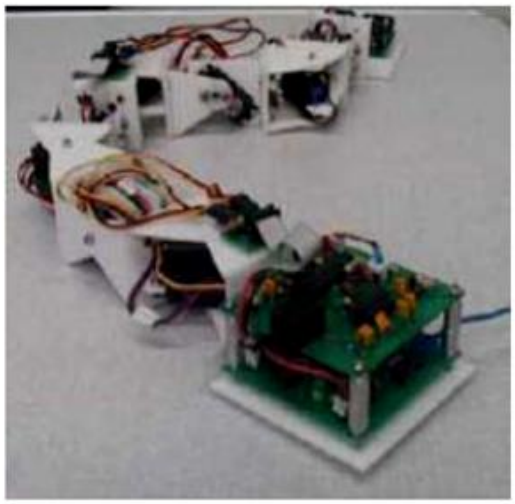


Fig. 7. Configuración articulada.

2.1.4 Robots bípedos

Este tipo de robots forman parte de la robótica móvil, específicamente se encuentran en la clasificación de robots con patas. Como su nombre indica, poseen dos patas para su locomoción. En su mayoría intentan imitar el sistema motriz de los humanos para desplazarse e interactuar con su entorno, entre los más conocidos están los robots humanoides.

Para explicar de una mejor manera el sistema de locomoción de los robots bípedos se presenta a continuación su evolución histórica:

2.1.4.1 Evolución histórica

La evolución de los robots humanoides se analiza desde el año 1986, haciendo un especial énfasis en el robot humanoide ASIMO (Advanced Step in Innovative Movile) y sus predecesores. A continuación se presenta toda la evolución de los robots bípedos hasta los robots que existen en la actualidad, concebidos como robots humanoides, tomando como guía el desarrollo que ha logrado Honda.

2.1.4.1.1 Estudios de la Locomoción Bípeda (1986)

El robot llamado E0, el cual se ilustra en la Figura 8, fue el primer intento de Honda para que un robot bípedo anduviera. Este prototipo tenía la capacidad de andar anteponiendo una pierna después de la otra, pero su desplazamiento era muy lento, ya que entre paso y paso tardaba 5 segundos. Esto se debía a la necesidad de mantener su centro de gravedad en la suela de sus pies, lo cual lo obligaba a detenerse para reajustar su equilibrio después de cada paso.

Fig. 8. E0.

Para lograr aumentar la velocidad de desplazamiento y para que pudiera caminar en superficies desiguales, desarrollaron la marcha rápida, la cual consistía en dar la mayor cantidad de pasos en el menor tiempo posible.

En la figura 9 se ilustra el movimiento del centro de gravedad de un robot con marcha lenta comparado con uno de marcha rápida.

Fig. 9. Marcha lenta y marcha rápida.

2.1.4.1.2 Desarrollo del desplazamiento rápido (1987-1991)

Para el desarrollo de esta nueva serie de prototipos, Honda se basó en la investigación y el análisis del caminar humano.

Experimental Model 1

En la figura 10 se ilustra el robot E1, Experimental Model 1. Este prototipo andaba a un paso estático de 0,25 Km/h con una cierta distinción entre el movimiento de las dos piernas.

Fig. 10. E1.

Experimental Model 2

En la figura 11 se ilustra el prototipo E1. Experimenta Model 2, para éste, Honda desarrolló el primer movimiento dinámico de 1,2 Km/h, imitando la manera de caminar de los humanos.

Fig. 11. E2.

Experimental Model 3

Honda sigue desarrollando el caminar de los humanos y además de ciertos animales para así conocer mejor la naturaleza de la locomoción bípeda. En la figura 12 se ilustra el prototipo E3-Experimental Model 3, con el cual se logró una velocidad de 3Km/h sobre superficies planas.

Fig. 12. E3.

2.1.4.1.3 Funciones básicas de la locomoción bípeda (1991 - 1993)

Luego de los desarrollos de Honda de la fase anterior, aún era necesario lograr, un paso rápido y equilibrado sobre cualquier tipo de superficie, sin que eso implicara la caída del robot.

Para ello se desarrollaron lo siguientes prototipos:

Experimental Model 4

En la figura 13 se ilustra el prototipo E4-Experimental Model 4, en donde se incrementó la longitud de la rodilla a 40 cm para simular la velocidad de la caminata humana a 4,7 Km/h.

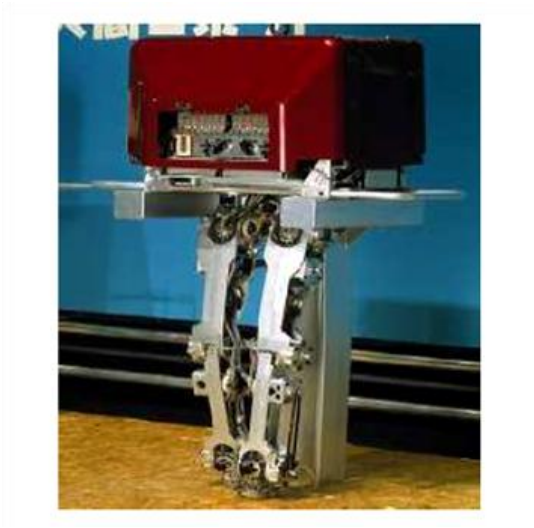


Fig. 13. E4.

Experimental Model 5

En la figura 14 se muestra el prototipo E5-Experimental Model 5, el cual fue el primer robot de Honda de locomoción autónoma.

Fig. 14. E5.

Experimental Model 6

En la figura 15 se ilustra el prototipo E6-Experimental Model 6, el cual ya cuenta con un control autónomo del equilibrio en la situación de subir y bajar gradas, rampas o evitar obstáculos.

Fig. 15. E6.

2.1.4.1.4 Investigación sobre robots humanoides totalmente independientes (1993-1997)

En esta serie se desarrollan robots humanoides, los cuales interactuaban con su entorno, siendo útiles en tareas básicas del ser humano.

A continuación se presentan los desarrollos durante este periodo:

Prototype Model 1

El robot de la Figura 16 ilustra a el robot P1-Prototype Model 1, el cual fue el primer prototipo con forma humana de Honda, su altura era de 1,915 m y su peso aproximado 175 Kg. Ya no solo disponía de piernas, tenía cuerpo y extremidades superiores, tenía la capacidad de activar y desactivar interruptores eléctricos y llevar objetos.

Fig. 16. P1.

Prototype Model 2

En la figura 17 se muestra al robot P2-Prototype Model 2. Honda sorprendió con este robot, pues fue el primer prototipo humanoide bípedo auto regulable del mundo, con una altura de 182 cm y un peso de 210 Kg. Su torso contenía un computador, una batería, un radio inalámbrico lo cual le permitía tener un control inalámbrico. El P2 podía realizar las tareas independientemente, sin cables.

Fig. 17. P2.

Prototype Model 3

En la figura 18 se muestra al robot P3-Prototype Model 3. Fue el primer robot humanoide bípedo, imitador de la forma de andar de los humanos, totalmente independiente. Este prototipo tiene una altura de 160 cm y un peso de 130 Kg el cual comparado con el anterior, se redujo considerablemente gracias al estudio de nuevos materiales y al sistema de control descentralizado que poseía. Su pequeño tamaño le permitía adaptarse mejor al entorno de trabajo de humanos.

Fig. 18. P3.

2.1.4.1.5 ASIMO (Advanced Step in Innovation MOvile)

A continuación se presenta la evolución del robot de Honda ASIMO, desde el 2000 hasta nuestros días:

ASIMO (2000)

En la figura 19 se muestra al robot ASIMO. Es el robot humanoide más avanzado del mundo su debut se realizó en el año 2000, y su nombre proviene de las siglas Advanced Step In MOvile, el cual se pronuncia "ashimo", que en japonés que significa "Piernas". La evolución de este robot le llevó a Honda 14 años de investigación de la locomoción bípeda.

Fig. 19. Asimo 2000.

ASIMO X2 (2002)

ASIMO X2 posee un avanzado sistema de reconocimiento facial añadido a sus capacidades de reconocimiento de voz y gesto.

ASIMO (2004)

Honda introdujo una nueva versión, la cual se muestra en la figura 20. Este prototipo fue mejorado en su diseño exterior y aumentada su autonomía permitiéndole correr a 3 Km/h, otro avance de este modelo fue la incorporación de pulgares opuestos a su mano, los cuales le permitían coger objetos y además sentir la fuerza ejercida cuando una persona cogía su mano.

Fig. 20. Asimo 2004.

ASIMO (2005)

En la figura 21 se muestra al prototipo ASIMO del año 2005, dándole habilidades profesionales como repartir café, entregar mensajes, empujar carritos, etc. Esta versión puede desplazarse a 6 Km/h.

Fig. 21. Asimo 2005.

Esta versión es utilizada como recepcionista en las oficinas. Para permitir esto Honda desarrolló una tarjeta de telecomunicación, esta tarjeta almacenaba información del personal, lo cual posibilitaba que ASIMO pueda reconocer unívocamente a sus compañeros de trabajo, además de aplicar otros reconocimientos de voz y gestos faciales para confirmar su identidad.

ASIMO (2007)

En la figura 22 se muestra al robot ASIMO (2007). En este prototipo entre sus mejoras, Honda desarrolla tecnologías que permiten inteligencia múltiple, lo cual permite trabajar a varios robots ASIMO en coordinación.

Otra de las innovaciones en este prototipo, es la creación de un sistema que le permite calcular la dirección y velocidad de las personas mientras ellas se desplazan, de tal manera que el robot no las pueda bloquear y permita el libre tránsito.

Además, se le agrega un sistema de carga de batería autónoma: cuando su almacenamiento de energía disminuye de cierto nivel, el robot detectará y buscará la estación de carga más cercana.

Fig. 22. Asimo 2007.

Por todas estas razones es obvio decir que ASIMO no es un juguete, está desarrollado para ayudar a los humanos. Es decir trabajar en casa, ayudar a los ancianos, etc.

CAPÍTULO 3: COMPONENTES UTILIZADOS

3.1 Microcontrolador

Fig. 27: Pic 18F2550

Como elemento de control utilizaremos un microcontrolador de la casa Microchip, en concreto el PIC 18f2550. Los PIC son microcontroladores de tipo RISC, de 8 bits, que incorporan periféricos muy diversos: temporizadores, unidades de comunicaciones serie síncronas y asíncronas, bus CAN, USB, conversores A/D, comparadores, etc.… por lo que se adaptan a una gran variedad de aplicaciones. En el cuadro siguiente se han resumido las características de este modelo.

Tabla 1. Características de PIC18F2550.

Parameter Name	Value
Program Memory Type	Flash
Program Memory (KB)	32
CPU Speed (MIPS)	12
RAM Bytes	2,048
Data EEPROM (bytes)	256
Digital Communication Peripherals	1-A/E/USART, 1-MSSP(SPI/I2C)
Capture/Compare/PWM Peripherals	2 CCP
Timers	1 x 8-bit, 3 x 16-bit
ADC	10 ch, 10-bit
Comparators	2
USB (ch, speed, compliance)	1, Full Speed, USB 2.0
Temperature Range (C)	-40 to 85
Operating Voltage Range (V)	2 to 5.5
Pin Count	28

La programación del pic se realiza *in circuit*, por lo que no es necesario extraerlo del zócalo. Eso facilita la programación de estos y evita posibles roturas de pines.

Este se encargará de controlar tanto los servomotores como los sensores del robot.

En la siguiente figura se muestra la distribución de pines del microcontrolador.

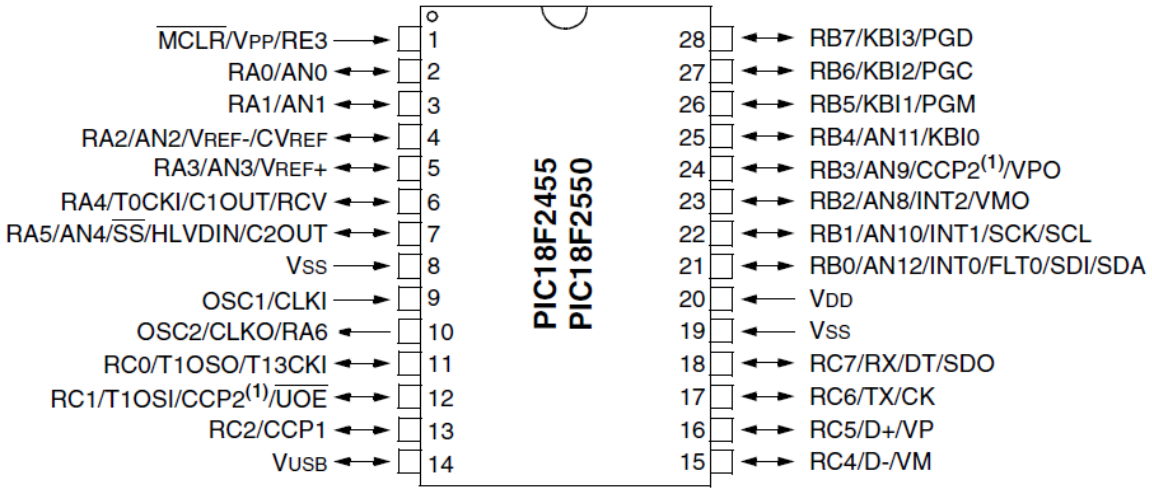


Fig. 28: Distribución de pines de PIC 18F2550.

3.1.1 Configuración del módulo oscilador:

Una de las primeras dificultades que se plantean a la hora de utilizar este microcontrolador es la configuración del oscilador. En caso de no tenerlo correctamente configurado no se conseguirá obtener los resultados esperados.

El módulo oscilador del PIC18F2550 viene dado de la siguiente manera:

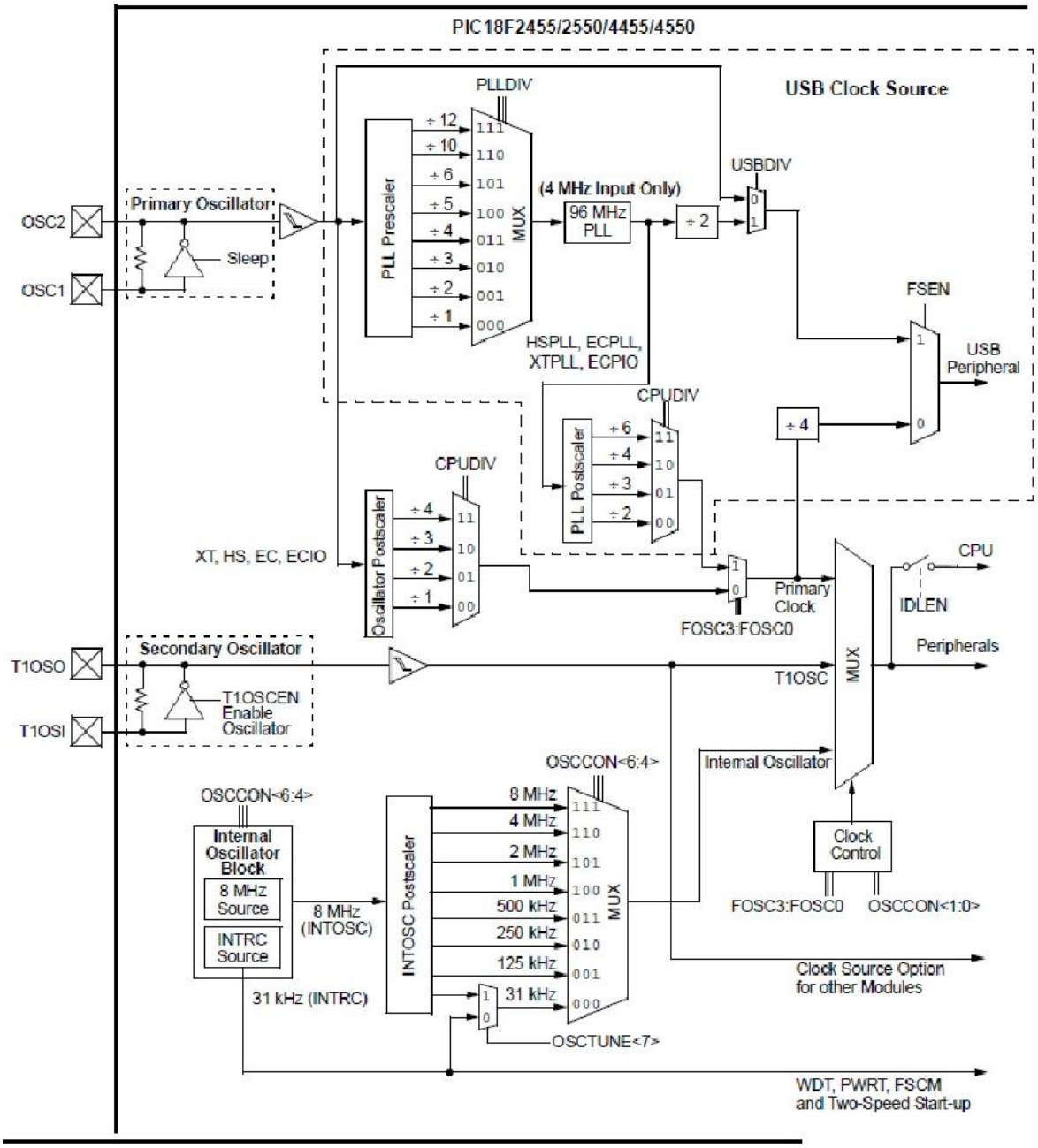


Fig. 29: Diagrama oscilador.

El oscilador tiene varias configuraciones según el cristal usado y como queramos disponer de él. Como podemos observar, se puede usar directamente la frecuencia de oscilación del cristal de cuarzo para enviarlo hacia la CPU o se puede hacer pasar dicha oscilación a un PLL que se encargará de multiplicarla de forma que se obtienen nada menos que 96MHz en su salida. Estos 96MHz luego serán divididos como mínimo por 2 para introducirlos hacia la CPU.

En nuestro caso, necesitamos la configuración que nos ofrezca la máxima velocidad posible a causa del algoritmo de ordenación. Ese algoritmo debe efectuarse en menos de 600uS que es el tiempo mínimo que necesitaría, en el peor de los casos, alguno de los servos.

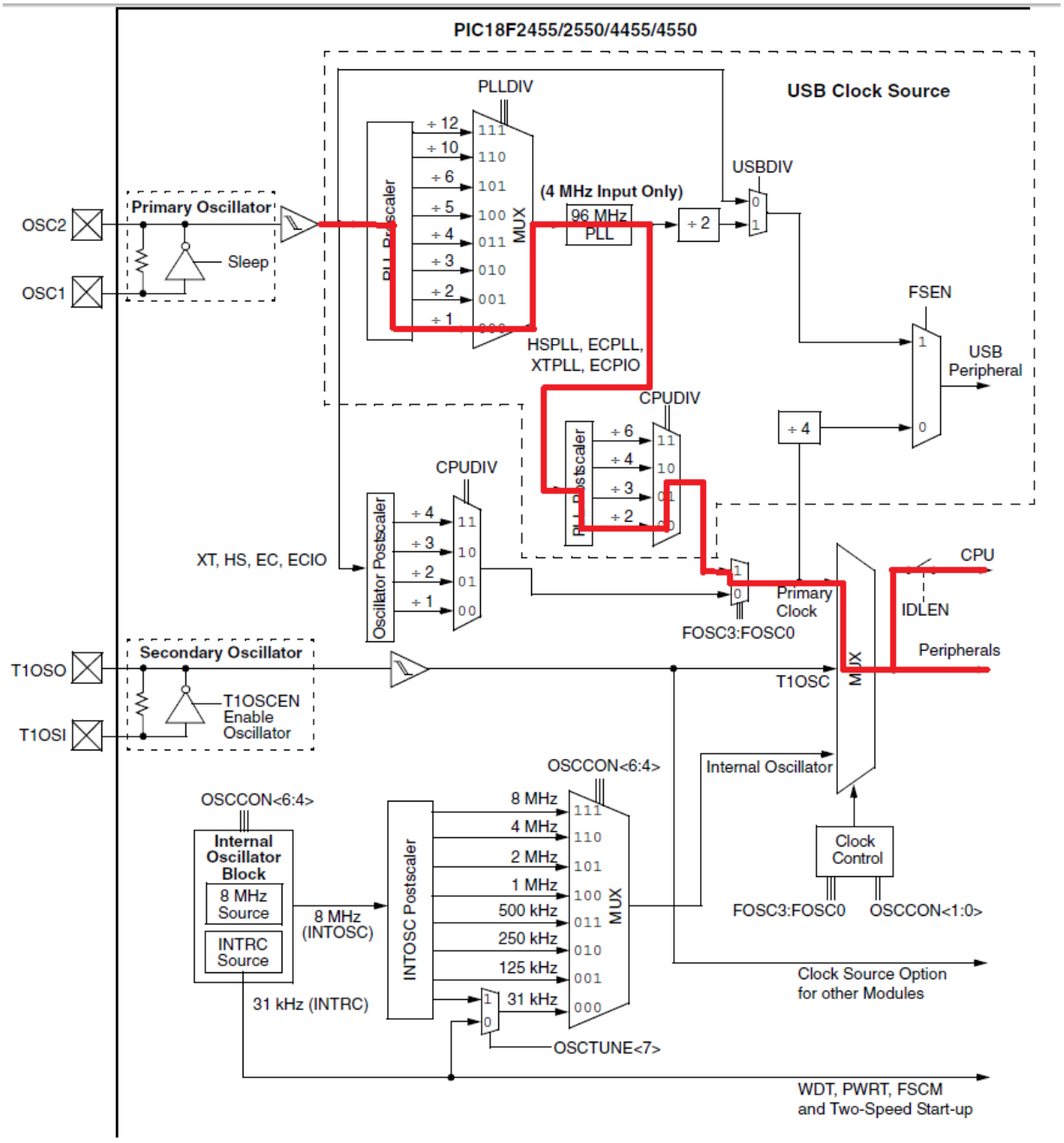


Fig. 30: Camino del oscilador.

Para obtener la configuración anterior se hace mediante los fuses de configuración. Primeramente debemos usar el fuse XTPLL:

Con XTPLL le indicamos al compilador que usaremos un cristal en conjunto con el PLL para generar los 48Mhz. Necesarios para nuestra aplicación. Como usamos un cristal de 4Mhz no es necesaria la utilización de un divisor en el postcaler. En caso de usar un cristal de distinta frecuencia, por ejemplo de 20Mhz se tendría que poner el fuse HSPLL y un divisor de postcaler de 5 para obtener los 4Mhz en la entrada del PLL. Veamos en el cuadro que opciones tenemos.

REGISTER 25-2: CONFIG1H: CONFIGURATION REGISTER 1 HIGH (BYTE ADDRESS 300001h)

R/P-0	R/P-0	U-0	U-0	R/P-0	R/P-1	R/P-0	R/P-1
IESO	FCMEN	—	—	FOSC3(1)	FOSC2(1)	FOSC1(1)	FOSC0(1)
bit 7							bit 0

Legend:		
R = Readable bit	P = Programmable bit	U = Unimplemented bit, read as '0'
-n = Value when device is unprogrammed		u = Unchanged from programmed state

bit 7 **IESO:** Internal/External Oscillator Switchover bit
1 = Oscillator Switchover mode enabled
0 = Oscillator Switchover mode disabled

bit 6 **FCMEN:** Fail-Safe Clock Monitor Enable bit
1 = Fail-Safe Clock Monitor enabled
0 = Fail-Safe Clock Monitor disabled

bit 5-4 **Unimplemented:** Read as '0'

bit 3-0 **FOSC3:FOSC0:** Oscillator Selection bits(1)
111x = HS oscillator, PLL enabled (HSPLL)
110x = HS oscillator (HS)
1011 = Internal oscillator, HS oscillator used by USB (INTHS)
1010 = Internal oscillator, XT used by USB (INTXT)
1001 = Internal oscillator, CLKO function on RA6, EC used by USB (INTCKO)
1000 = Internal oscillator, port function on RA6, EC used by USB (INTIO)
0111 = EC oscillator, PLL enabled, CLKO function on RA6 (ECPLL)
0110 = EC oscillator, PLL enabled, port function on RA6 (ECPIO)
0101 = EC oscillator, CLKO function on RA6 (EC)
0100 = EC oscillator, port function on RA6 (ECIO)
001x = XT oscillator, PLL enabled (XTPLL)
000x = XT oscillator (XT)

Note 1: The microcontroller and USB module both use the selected oscillator as their clock source in XT, HS and EC modes. The USB module uses the indicated XT, HS or EC oscillator as its clock source whenever the microcontroller uses the internal oscillator.

Fig. 31: Registro de configuración.

En la figura anterior vemos como configuramos el registro al poner el fuse ya nombrado.

A parte de los bits 3-0, el compilador configura el bit 7 i el bit 6 tal como se muestra en la figura por defecto.

El siguiente fuse necesario es el PLL1. Aquí se selecciona el factor de división del *postcaler* y se seleccionará teniendo en cuenta el valor del cristal que se ha utilizado. Siempre se tiene que tener en cuenta que se necesitan 4Mhz en la entrada del PLL para que este genere 96Mhz, de lo contrario, el PLL no podría

engancharse a dicha frecuencia. En nuestro caso utilizaremos un cristal de 4Mhz por lo que el factor de división necesario va a ser 1.

Por último, queda por configurar la entrada de la señal de 96MHz hacia la CPU y periféricos del microcontrolador. Para conseguirlo se debe poner el fuse CPUDIV1 que es el que divide la frecuencia de 96MHz por 2, por tanto nos quedan 48MHz hacia la CPU.

Otro de los fuses que hemos considerado es el de MCLR, aquí se le dice al compilador que se usará la función MCLR del Pin 1 del microcontrolador. También existe la posibilidad de poner NOMCLR con lo cual se deja libre el pin RE3 que se puede configurar entonces solo como entrada.

Veamos en el cuadro como se configura este bit.

REGISTER 25-5: CONFIG3H: CONFIGURATION REGISTER 3 HIGH (BYTE ADDRESS 300005h)

R/P-1	U-0	U-0	U-0	U-0	R/P-0	R/P-1	R/P-1
MCLRE	—	—	—	—	LPT1OSC	PBADEN	CCP2MX
bit 7							bit 0

Legend:		
R = Readable bit	P = Programmable bit	U = Unimplemented bit, read as '0'
-n = Value when device is unprogrammed		u = Unchanged from programmed state

bit 7 **MCLRE:** $\overline{\text{MCLR}}$ Pin Enable bit
1 = $\overline{\text{MCLR}}$ pin enabled, RE3 input pin disabled
0 = RE3 input pin enabled, $\overline{\text{MCLR}}$ pin disabled

Fig. 32: Registro de configuración.

El siguiente fuse necesario es el de NOWDT que significa que no se utiliza el *Whachdog* o perro guardián. El *whachdog* es una herramienta útil para evitar que nuestro microcontrolador pueda entrar en un búcle fuera del programa. Ese error en el programa puede ser debido a causas de errores de programación o simplemente por posibles factores externos, de forma que se podría quedar allí en ese punto de programa indefinidamente. Este WHATCHDOG timer se encarga de que eso no ocurra. Para conseguirlo, empieza a contar hasta llegar a desbordarse y, una vez desbordado, aplica un reset al microcontrolador, por tanto, este empieza su programa desde el principio. Pero al programador no le interesa que eso ocurra cada dos por tres porque sino nunca se llegaría a ejecutar el programa entero. En el set de instrucciones existe una instrucción para resetear ese *timer* y de esa forma evitar que se desborde sin que sea necesario. En nuestro caso no utilizaremos esa protección a causa de que está basado en el *timer0*, *timer* que ya está usado para poder controlar correctamente los servos.

En el siguiente cuadro se ven las opciones disponibles respecto al *Whachdog*. En el podemos observar los distintos modos de que se disponen y el bit de encendido de este:

REGISTER 25-4: CONFIG2H: CONFIGURATION REGISTER 2 HIGH (BYTE ADDRESS 300003h)

U-0	U-0	U-0	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1
—	—	—	WDTPS3	WDTPS2	WDTPS1	WDTPS0	WDTEN
bit 7							bit 0

Legend:		
R = Readable bit	P = Programmable bit	U = Unimplemented bit, read as '0'
-n = Value when device is unprogrammed		u = Unchanged from programmed state

bit 7-5 **Unimplemented:** Read as '0'

bit 4-1 **WDTPS3:WDTPS0:** Watchdog Timer Postscale Select bits
1111 = 1:32,768
1110 = 1:16,384
1101 = 1:8,192
1100 = 1:4,096
1011 = 1:2,048
1010 = 1:1,024
1001 = 1:512
1000 = 1:256
0111 = 1:128
0110 = 1:64
0101 = 1:32
0100 = 1:16
0011 = 1:8
0010 = 1:4
0001 = 1:2
0000 = 1:1

bit 0 **WDTEN:** Watchdog Timer Enable bit
1 = WDT enabled
0 = WDT disabled (control is placed on the SWDTEN bit)

3.1.2 Funcionamiento del convertidor A/D del PIC

Los microcontroladores PIC pueden incorporar un módulo de conversión de señal analógica a señal digital. Los módulos AD que utiliza Microchip hacen un muestreo y retención (simple & hold) con un condensador y después utiliza el módulo de conversión de la figura siguiente:

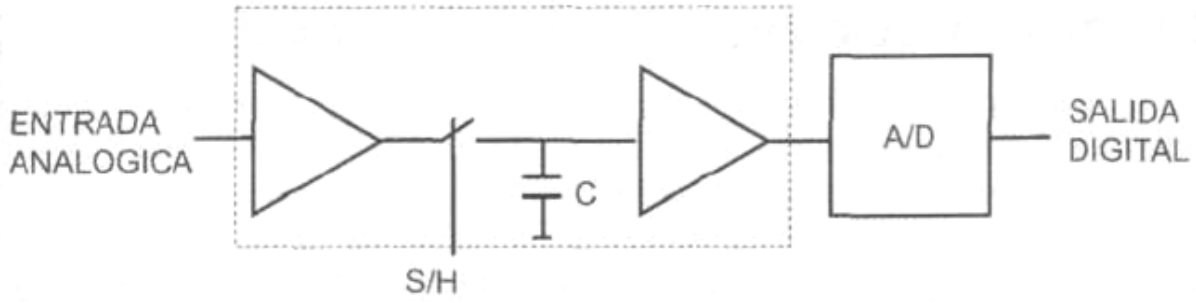


Fig. 33. Esquema general de convertidor A/D.

El módulo de conversión A/D es del tipo de aproximaciones sucesivas.

El convertidor de aproximaciones sucesivas se utiliza en aplicaciones donde se necesitan altas velocidades de conversión. Se basa en realizar sucesivas comparaciones de forma ascendente o descendente hasta encontrar un valor

digital que iguale la tensión entregada por el conversor D/A y la tensión de entrada.

Durante la fase de muestreo el interruptor se cierra y el condensador se carga a la tensión de entrada (el tiempo que el interruptor está cerrado es fundamental para la correcta carga del condensador). Una vez abierto el interruptor, el condensador mantendrá (teóricamente) la tensión de entrada mientras el módulo A/D realiza la conversión.

El módulo de conversión se caracteriza por parámetros como los siguientes:

· Rango de entrada.

· Número de bits.

· Resolución.

· Tensión de fondo de escala.

· Tiempo de conversión.

· Error de conversión.

El módulo de conversión de los PIC de gama media tiene un número de bits de 10, por lo que su resolución es:

$$resolución = \frac{V_{in}}{2^N - 1}$$

Siendo V_{in} la tensión de entrada y N el número de bits del convertidor. Es decir, para la tensión máxima de entrada (5V) la resolución es de 0.0048 (4.8 mV) por LSB.

La resolución sí cambia si se modifica la tensión de fondo de escala, es decir, la tensión de referencia. Los PICs permiten cambiar la tensión de referencia en un valor absoluto (de 0 a +V) o en un margen (de –V a +V).

Las tensiones a convertir siempre son positivas.

Los PIC 16f876 tienen 5 canales (en el puerto A). El convertidor es de 10 bits y, tal y como se ha comentado, es de aproximaciones sucesivas. Permite variar la tensión de referencia a la máxima Vcc o a una tensión positiva menor a través de AN3/Vref+ y a la mínima Vss o a una tensión positiva mayor a través de AN2/Vref-.

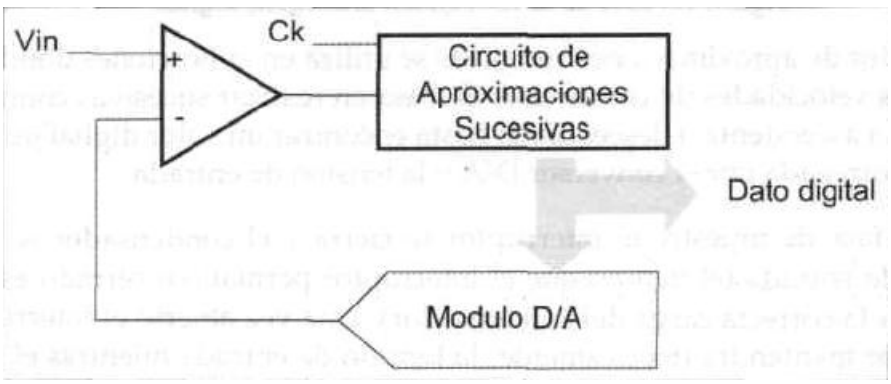


Fig. 34. Esquema de funcionamiento del convertidor A/D.

Puede seguir funcionando cuando el PIC está en modo *sleep* ya que dispone de un oscilador RC interno propio.

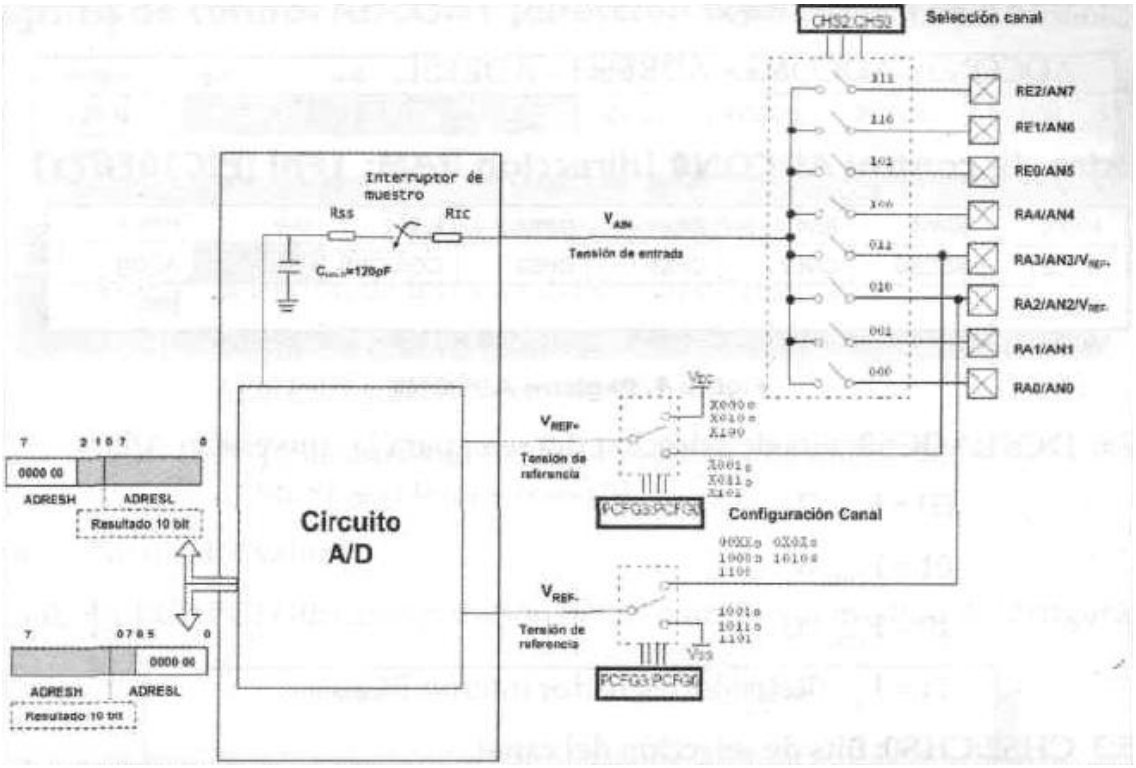


Fig. 35. Esquema interno del PIC.

La función de transferencia del convertidor A/D es el resultado de que la primera transición ocurra cuando la tensión analógica de entrada es igual a Vref/1024.

La resolución vendrá dada por la siguiente ecuación:

$$1LSB = V_{REF-} + \frac{(V_{REF+} - V_{REF-})}{1024}$$

En el caso de que la V_{REF+} = V_{DD} y V_{REF-} = Vss entonces la resolución es:

$$1LSB = \frac{5}{1024} = 4.8mV$$

De esta forma si la lectura es de 512 LSB, la tensión analógica leída es:

$$V_{IN} = 512 \cdot \frac{5}{1024} = 512 \cdot 4.8mV = 2.4576\ V$$

3.1.3 Configuración de los timers 0 i 1

Como ya es sabido, una modulación PWM es una onda cuadrada que siempre se encuentra en la misma frecuencia, y lo que va variando es la relación entre el tiempo en estado alto y, en consecuencia, el tiempo en estado bajo, ya que siempre debe tener la misma frecuencia. En este caso, lo que se hará es utilizar el *timer* del PIC configurado y calculado para que cada 50 Hz entre en la

interrupción por *timer*. De esa forma aseguramos que cada 20 ms se entre en esa interrupción y se actúe en consecuencia.

La necesidad de utilizar el timer0 en vez de usar el PWM propio del microcontrolador es a causa de que solo se dispone de 2 módulos PWM dentro de este y se precisan como mínimo 10 para controlar las piernas

Para conseguirlo lo que haremos es utilizar la fórmula siguiente que describe el funcionamiento del *timer* 0:

$$T = (T_{CM} \cdot Prescaler \cdot (65536 - carga\ TMR0)$$

Donde T_{CM} es el ciclo maquina que se puede calcular mediante la ecuación:

$$T_{CM} = \frac{4}{F_{osc}}$$

Se conoce T_{CM}, conocemos el preescaler y conocemos T, lo único que nos falta es la carga del TMR0.

Usando un preescaler de 4 obtenemos que se debe cargar el TMR0 aproximadamente con 5536. Ahora lo debemos convertir a hexadecimal, que corresponde al número 15A0.

Realizando pruebas se pudo observar que los servomotores no necesitaban explícitamente que fueran 20ms de periodo de señal sino que podía ser un poco mayor sin ningún problema. Lo que se hace al final es cargar el *timer0* con un valor de 1B que corresponde a un período de aproximadamente 22ms.

Parece innecesaria la idea de añadir esos 2 milisegundos pero si se realiza un pequeño cálculo se puede observar lo siguiente.

- F_{CM}=12MHz
- Tiempo añadido 2ms

$$2ms \cdot \frac{12000000cm}{1s} = 24000cm$$

Se observa que en ese tiempo el microcontrolador ha 24000 ciclos máquina.

Por un lado ya se tiene la interrupción que se encargará de crear la señal PWM, pero por otro lado se necesitará contar el tiempo en micro segundos para poder saber en qué momento se debe bajar el pulso de cada servomotor. Para hacerlo se utilizará la interrupción por *timer 1*.

Se sabe que los servomotores utilizados tienen unos anchos de pulso de entre 600us i 2400us, por lo tanto, lo que se hará es, en un principio, configurar el *timer1* para que se encienda junto al *timer0*. En este primer momento se cargará de forma que no entre a la interrupción hasta los primeros 500us después de haber empezado el algoritmo de señales PWM. De lo contrario relentecería mucho el proceso de ordenación de la tabla de servomotores. Una vez pasados esos 500us se cargará el *timer1* para que entre a la interrupción cada 10us. Sabiéndose que entre el pulso máximo y el pulso mínimo hay 1800us de

diferencia y que el servomotor es capaz de moverse alrededor de 180º obtenemos que:

$$\frac{1800us}{180º} = 10us/º$$

Teniendo la interrupción por *timer1* configurada cada 10us obtenemos una resolución de control de los servomotores de 1º.

Para saber la cifra con la que debemos cargar el *timer* al principio nos regiremos por la formula siguiente:

$$T = (T_{CM} \cdot Prescaler \cdot (65536 - carga\ TMR1)$$

Usando un preescaler de 1 obtenemos que se debe cargar el TMR1 aproximadamente con 59536. Pero por causas de retardos por causa del compilador C, por ensayos se sabe que se debe cargar con el valor de 59560.

Ahora ya se conoce cuál es el valor de la primera carga. Nos queda por definir el valor de las siguientes cargas de 10us de tiempo. Basándose en la formula anterior obtenemos que usando un preescaler de 1 se debe cargar el TMR1 aproximadamente con 65413. Pero por causas de retardos por causa del compilador C, por ensayos se sabe que se debe cargar con el valor de 65466.

3.1.4 Lenguaje de programación

Compiladores para programar Microcontroladores PIC hay muchos en el mercado. Los hay que compilan código en lenguaje ensamblador, en C, en Basic, en Pascal etc., pero ¿Cuál utilizar?

Primeramente definiremos y consideraremos las ventajas e inconvenientes que tiene el uso de lenguajes de alto nivel como el C frente al uso de otros compiladores que utilizan el lenguaje ensamblador.

Programación en lenguajes de bajo nivel.

En el principio de los tiempos de los Microprocesadores la única forma que había de programarlos era utilizando el Código Máquina en el cual la unidad central de proceso o CPU procesaba instrucciones que venían definidas por un conjunto de unos y ceros de 8, 16, 32 o más bits. Cada combinación diferente de bits, tiene para el micro un significado distinto, y le indicará a éste que realice una tarea determinada. Sin embargo, este tipo de programación resultaba poco inteligible para el ser humano, porque se tenían que manejar constantemente conjuntos de 'unos' y 'ceros' sin ningún significado aparente. Por este motivo se desarrolló el lenguaje Ensamblador (*Assembler*), que consistía en asignar a cada combinación de bits un conjunto de pocas letras (denominado mnemónico) que representaba mejor el significado de la operación o instrucción que se le indica al microprocesador. Cuando se compila un programa en Ensamblador, el compilador de Ensamblador realiza automáticamente la traducción de los mnemónicos a Código Máquina (conjunto de bits), que es el único lenguaje que entiende la CPU.

Durante mucho tiempo el ensamblador ha sido el lenguaje utilizado de manera casi mayoritaria para programar microcontroladores. Sin embargo, este lenguaje implementa el conjunto de instrucciones que cada microcontrolador en concreto entiende, por lo que es totalmente dependiente del hardware, sin que pueda

hacerse transportable a otros microcontroladores de una manera fácil.!Esto quiere decir que el Ensamblador no constituye realmente un lenguaje de programación estándar, capaz de permitir el transportar un programa diseñado para un microcontrolador a otro que tenga un conjunto de instrucciones diferente.

Programación en lenguajes de alto nivel

La programación en lenguaje de alto nivel permite la creación de programas independientemente de la plataforma utilizada, ya estemos hablando de microprocesadores o microcontroladores. Además hay que tener en cuenta que los microcontroladores evolucionaron a partir de los microprocesadores y no al revés, los microcontroladores aparecieron en la industria por la necesidad de tener sistemas programados embebidos, es decir que tanto CPU, memoria y periféricos estuvieran integrados dentro de un mismo circuito integrado.

Algunos de estos lenguajes de alto nivel son el BASIC, FORTRAN, PASCAL y C. Este último además de permitir la programación desde un nivel cercano al programador (alto nivel), también brinda la posibilidad de controlar aspectos más cercanos al hardware (bajo nivel), como la manipulación directa de bits y bytes, por lo que se considera que es un lenguaje de nivel medio, más que de alto nivel, esta ya sería una razón para decantarse por compiladores de C y no de otros lenguajes de alto nivel como el Basic ó Pascal.

Ventajas del lenguaje C frente al Ensamblador

- Mayor facilidad de programación. El lenguaje C dispone de un conjunto de operadores, datos y comandos que le confieren, al mismo tiempo, potencia y facilidad de programación, lo que permite un tiempo de desarrollo de programas mucho menor que con el lenguaje Ensamblador.

- Portabilidad entre sistemas. Con el lenguaje C se asegura la portabilidad entre diferentes plataformas hardware o software, lo que quiere decir, por ejemplo, que un algoritmo implementado en C en una plataforma con Linux puede ser adaptado, prácticamente sin modificaciones, a un PIC. Esto permite el aprovechamiento de numerosos algoritmos que se encuentran ya disponibles para otras plataformas diferentes a los PIC. El Ensamblador, por el contrario, ya se ha indicado que es fuertemente dependiente del hardware, por lo que no permite su adaptación de una plataforma a otra distinta.

- Desarrollo de programas estructurados. El lenguaje C permite desarrollar programas estructurados en funciones, bloques o procedimientos, lo que proporciona una compartimentación del código. Por el contrario, el Ensamblador no es un lenguaje estructurado, lo que lleva a que los programas desarrollados en ensamblador sean lineales, con el inconveniente que esto implica en lo que se refiere a claridad del código escrito.

- Fácil mantenimiento de los programas. Por ser un lenguaje de compresión relativamente fácil.

A pesar de todas las ventajas indicadas para el lenguaje C, no se puede prescindir totalmente del Ensamblador. Dependiendo del nivel de eficiencia que se quiera adquirir, el Ensamblador, al estar más cercano al nivel del hardware que se programa, permite generar código más compacto (menor número de instrucciones para realizar una misma tarea), lo que lleva a una mayor velocidad de ejecución.

Aunque el desarrollo de programas para microcontroladores es posible realizarlo totalmente en Ensamblador, la utilización del lenguaje C supone una alternativa muy interesante por su rapidez, facilidad, y portabilidad, sin que esto signifique que este nuevo enfoque venga a sustituir definitivamente al Ensamblador, en casos en los que se necesita crear partes de código sujetas a determinadas restricciones (reducido número de instrucciones, alta velocidad de ejecución,…) se puede implementar esa parte de código directamente en Ensamblador dentro de un programa en C, consiguiendo de este modo un código mucho más eficiente. Esta ventaja no la tienen otros compiladores de Basic ó Pascal.

Compiladores de C para PIC.

La oferta de compiladores de C para PIC es muy numerosa, a la hora de elegir entre uno u otro es necesario tener en cuenta los siguientes factores:

- Optimización del código generado.

- Dispositivos para los que el compilador es capaz de generar código.

- Biblioteca de funciones precompiladas y directivas de las que disponen, un compilador que disponga ya de muchas funciones listas para usarse ahorra mucho trabajo al programador.

- Posibilidades adicionales, como inserción de código Ensamblador, depuración, etc.

- Precio.

Los principales son los siguientes:

- MPLAB-C18 (Microchip). Compilador de la casa Microchip el mismo fabricante que fabrica los PIC. Este concretamente es para los PIC de 8 bits.

- MPLAB-C24 (Microchip) como el anterior pero para microcontroladores de 16 bit incluyendo los dsPIC con capacidad de procesamiento de señales digitales.

- MPLAB-C32 (Microchip) para programar Microcontroladores de 32 bits la gama más alta de Microchip.

Los que producen un código más optimizado son estos tres últimos para eso los desarrolladores de los compiladores son los que fabrican los PIC, además en las gamas altas aumenta el Set de instrucciones específicas para C que se le añaden a la CPU y los primeros en aprovechar los recursos Hardware son ellos, sin embargo no son compiladores populares para el que empieza en la programación de estos dispositivos. Además hay que tener un compilador para cada gama de Microcontroladores y el C18 aunque es un compilador para dispositivos de 8 bits no acepta los de gama más baja incluido el famoso 16F84, por lo que ante la posibilidad de variar el microchip utilizado durante la programación no seria práctico.

- PICC (Hi -Tech). Sin duda el mejor compilador (profesionalmente hablando) hecho por terceros, está disponible para las plataformas Windows, Linux y MAC incluye soporte para la gama de 24 y 32 bits y su código es muy portable, ya que cumple prácticamente con el estándar ANSI C, además admite perfectamente la integración de código ensamblador dentro del C. Pero, como todas las cosas buenas hay que pagarlas la versión profesional completa vale más de 1000$

aunque existe una versión Lite gratis para las tres plataformas que puedes conseguir tras registrarte.

Otro inconveniente es que al ser un producto enfocado al desarrollo profesional no existen muchos Kits de aprendizaje, es decir, conjunto de tarjetas preparadas para probar los ejemplos del compilador, hasta para bajarte los manuales de la página del fabricante te tienes que registrar.

- PCW Compiler (Custom Computer Services - CCS). Genera un código máquina muy compacto y eficiente. Además se integra perfectamente con MPLAB y otros simuladores/emuladores como PROTEUS para el proceso de depuración. Incluye una biblioteca muy completa de funciones precompiladas para el acceso al hardware de los dispositivos (entrada/salida, temporizaciones, conversor A/D, transmisión RS- 232, bus I2C….,etc) así como drivers para dispositivos externos (LCD, teclados numéricos, memorias EEPROM, conversores A/D, relojes en tiempo real, etc). Si además queremos cuidar de forma minuciosa una parte del programa permite insertar partes de código directamente en Ensamblador, manteniendo otras partes del programa en C.

- mikroC un buen compilador de C para aprender a programar los PIC, de la página del fabricante se puede bajar abundante documentación así como una versión demo del mismo, el fabricante también ofrece compiladores para otros lenguaje como el Basic y Pascal, así como abundantes ejemplos y kit de desarrollo. Se trata de un compilador muy a tener en cuenta a la hora de empezar a programar PIC en C.

- MPC (Byte Graft). Otro compilador para PIC en C aunque menos conocido.

También permite incluir código ensamblador y se integra con el MPLAP pero ni punto de comparación en cuanto documentación y ejemplos a los dos anteriores.

- SDCC Compi ler es un pequeño compilador de software libre para las plataformas Linux, Windows y MAC en el que podemos desarrollar aplicaciones para dispositivos como Intel 8051, Maxim 80DS390, Zilog Z80 y el Motorola 68HC08 actualmente está en fase de desarrollo el poder soportar los PIC de 16 y 18 bits.

De entre todos éstos PCW Compiler y mikroC son los que tienen una mejor relación prestaciones/precio y además disponen de abundante documentación y ejemplos en la red, lo que los hace ideales para empezar a programar PIC en C.

3.1.5. Programador del microcontrolador

Entre las diversas opciones que se ofrecen en el amplio mercado se ha escogido el Pickit v.2 por su sencillez de uso, precio y facilidad para conseguirlo.

El PICkit 2 es capaz de programar la mayor parte de los microcontroladores flash de Microchip, soporta la línea básica (PIC10F, PIC12F5xx, PIC16F5xx), la línea intermedia (PIC12F6xx, PIC16F), PIC18F, PIC24, dsPIC30 y dsPIC33) y también muchos productos EEPROM de Microchip™. En especial se mencionan algunos de los más populares como el PIC16F84A y el PIC16F628A. Además la lista de circuitos que soporta se puede actualizar directamente de la página de internet de Microchip.

Fig. 36: PICKIT2 Programer.

De cara a programar se ha seguido el esquema del fabricante para conexión directa con el microchip:

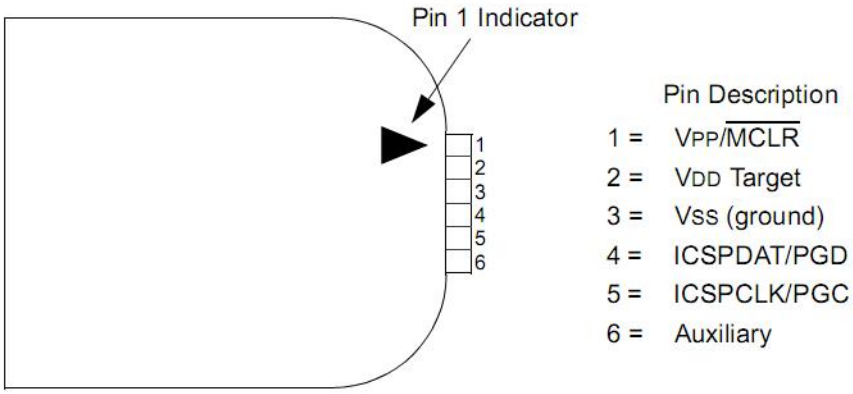


Fig. 37: Esquema de conexión del PICKit al microprocesador.

3.2 Regulador de tensión

El sistema de alimentación vendrá subministrado por una batería de 6V, y, por otra parte, el microcontrolador debe ir alimentado a 5V. Aquí se plantea un nuevo problema a causa que los reguladores comerciales lineales típicos, por ejemplo el LM7805, tienen una caída de tensión entre su entrada y su salida de aproximadamente 2,5V. Rápidamente se observa que 6V-2.5V nos daría un resultado de 3,5V en su salida. Por una parte, el microcontrolador a esa tensión estaría en un punto en que si bajara un poco más, este dejaría de funcionar correctamente. Por otro lado, al tener un regulador de 5V de tensión en la salida y 6V en la entrada, no nos aseguraría tener una tensión estable.

Para solucionar ese problema se plantea la posibilidad de usar un regulador LDO (low drop-out). Su característica principal es que entre su tensión de entrada y su tensión de salida, existe una caída de tensión muy pequeña comparada con los reguladores convencionales.

El regulador escogido es el LM2940 i a continuación podemos ver sus características:

Features

- Dropout voltage typically 0.5V @I_O = 1A
- Output current in excess of 1A
- Output voltage trimmed before assembly
- Reverse battery protection
- Internal short circuit current limit
- Mirror image insertion protection
- P+ Product Enhancement tested

Podemos observar a partir de sus características que es capaz de subministrar 1ª sin problemas pero en esta aplicación no será necesario disponer de tanta corriente ya que solo se alimentaran a 5V los componentes que no sean servomotores. Los servomotores irán conectados directamente a la batería. También podemos observar que su caída de tensión es de aproximadamente 0,5V, más que suficiente para esta aplicación. Este regulador precisa, tanto en su entrada como en su salida unos condensadores para que no tenga problemas de estabilidad y así añadirle un polo dominante. Se puede ver en la figura siguiente como quedaría configurado:

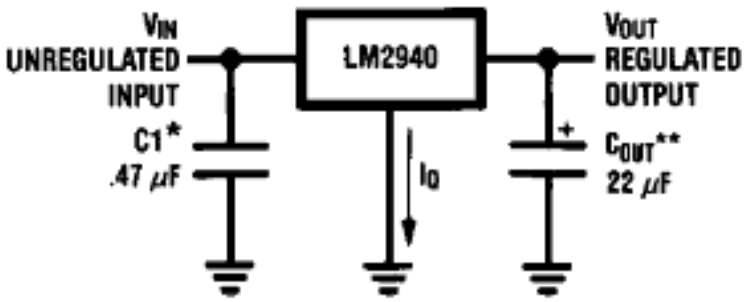


Fig. 38: Esquema de conexión de LM2940.

Se observa que los condensadores necesarios para su buen funcionamiento son de 470nF en su entrada y de 22uF en su salida.

3.3 Servomotores

Para lograr movimientos en el robot se utilizan servomotores, los cuales están compuestos de un motor DC y un sistema de control de posición. No es necesario que tengan revolución continua, su posición angular normalmente varía entre 0º a 180º, pero también existen servomotores multivuelta, pero en nuestro caso no son necesarios. Este tipo de servomotores tiene una gran aplicación en sistemas de aeromodelismo, además son útiles en robótica como por ejemplo en robots manipuladores, pinzas, brazos, etc. Sus motores son pequeños y sumamente poderosos para su tamaño. Existe una gran variedad de marcas de servomotores, así como de tamaño, velocidad o torque, pero todos ellos se

controlan de la misma manera. El servomotor internamente realiza un control de posición en lazo cerrado con realimentación de la posición, para lo cual utiliza un potenciómetro colocado en el eje central del motor y una circuitería de control. En la Figura siguiente se muestra la composición interna de un servomotor, el mismo que está compuesto de un motor DC, la circuitería de control, juego de piñones, los cuales permiten disminuir su velocidad e incrementar su torque denominada caja reductora, y la carcasa del mismo, así como también de tres cables de conexión externa, de los cuales: uno es para alimentación, otro para conexión a tierra y el otro de control.

3.3.1 Funcionamiento

Los servomotores, sin importar su tamaño, su torque y su velocidad, se controlan utilizando la técnica de control PWM (modulación por ancho de pulso). Este sistema consiste en generar una onda cuadrada, en la cual se varía el tiempo en que el pulso se encuentra en nivel alto. Al variar este ancho de pulso se consigue variar el ángulo de giro del servomotor. Cada servomotor tiene rangos de operación, el mínimo y el máximo ancho de pulso que el servomotor entiende. Los valores más generales de operación varían de 1ms a 2ms de ancho de pulso, los cuales permiten trabajar al mismo entre 0º y 180º respectivamente. El valor 1.5ms corresponde el valor central o neutro (90º), los valores antes mencionados son referenciales y pueden tener ciertas variaciones. Si se sobrepasan los valores recomendados por el fabricante, el servomotor emitirá un zumbido indicando que se debe cambiar la longitud del pulso. Además, se cuenta con un limitante dado por el tope del potenciómetro y los límites mecánicos constructivos.

El periodo en donde el pulso se encuentra en nivel bajo no es el crítico; puede variar entre un pulso y otro pulso, por lo general suele estar alrededor de 20ms. En la Figura siguiente se muestra la señal de control PWM y la posición del eje del servomotor para los valores máximo, mínimo y central de operación del mismo.

Se ha preferido usar este tipo de motores a causa de que, como ya se comentaba anteriormente, solo necesitan un pulso PWM para su gobernación, y ni tiene que ser de alta potencia ese pulso sino que solo es una señal. La potencia realmente se transmite a través de los cables de alimentación del

mismo. Gracias a este sistema, el uso de servomotores facilita el circuito de control al no necesitar drivers de potencia para cada uno de ellos.

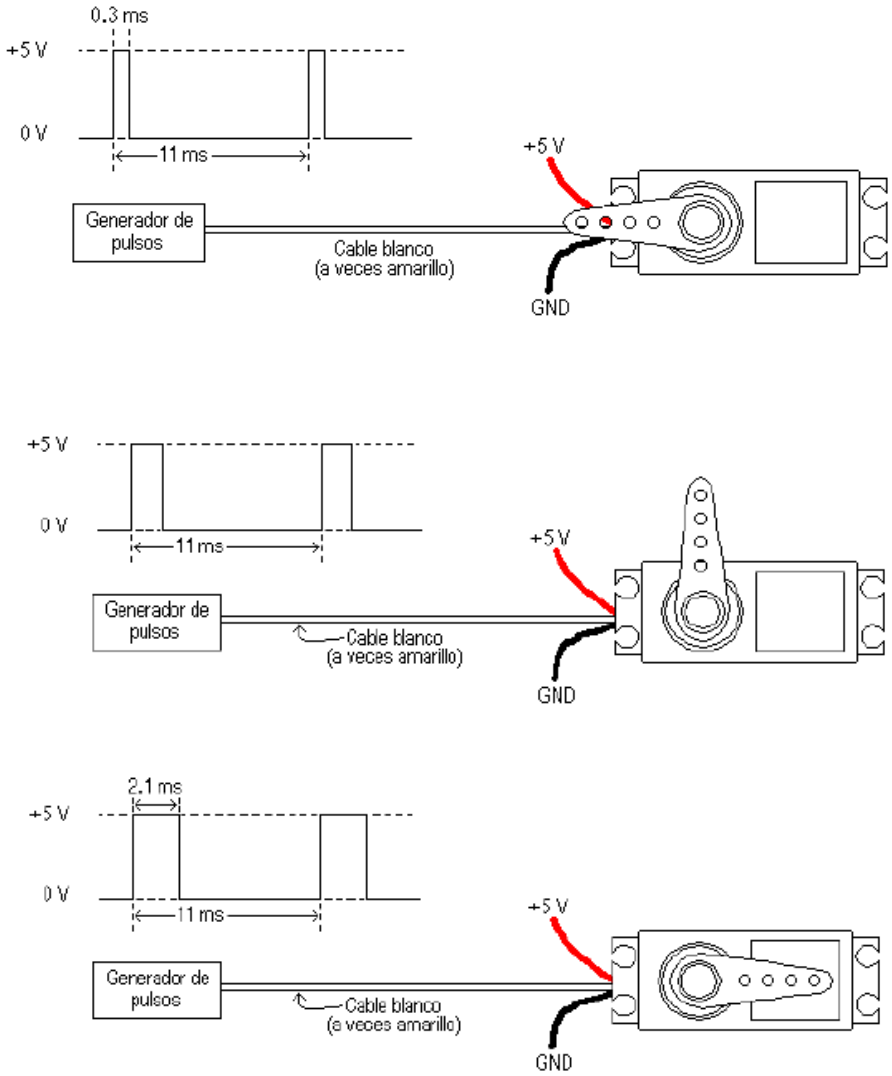


Fig. 39. Funcionamiento de servomotor.

De los tres cables que posee el servomotor para su funcionamiento, se tiene que el cable rojo es la alimentación (por lo general 5V), el cable negro es el de la tierra y el cable amarillo es el de control, al cual se le aplica la técnica PWM.

En la Tabla siguiente se muestra los valores de control y la disposición de los colores de los cables de los fabricantes más relevantes de servomotores.

Tabla 2. Comparativa de servomotores.

	Duración de pulso (ms)				*Disposición de cables*		
Fabricante	*Min.*	*Neutral*	*Max.*	*Hz*	*+ batt*	*- batt*	*PWM*
Futaba	0.9	1.5	2.1	50	rojo	negro	blanco
Hitech	0.9	1.5	2.1	50	rojo	negro	amarillo
Graupner/Jr	0.8	1.5	2.1	50	rojo	marron	naranja
Multiplex	1.05	1.6	2.15	40	rojo	negro	amarillo
Robbe	0.65	1.3	1.95	50	rojo	negro	blanco
Simprop	1.2	1.7	2.2	50	rojo	azul	negro

3.3.2 Servomotor HITEC HS 645-MG.

Fig. 40: Servomotor Hitec HS645-MG.

El servomotor usado para las pruebas de funcionamiento el servomotor Hitec HS 645-MG. Sus características principales son las de la tabla siguiente.

Motor Type:	3 Pole
Bearing Type:	Dual Ball Bearing
Speed (4.8V/6.0V):	0.24 / 0.20
Torque oz./in. (4.8V/6.0V):	107 / 133
Torque kg./cm. (4.8V/6.0V):	8.0 / 10.0
Size in Inches:	1.59 x 0.77 x 1.48
Size in Millimeters:	40.39 x 19.56 x 37.59

Weight ounces:	1.94
Weight grams:	55

A más a más, según pruebas realizadas se puede afirmar que su consumo medio es de aproximadamente 300mA en funcionamiento normal i alimentado a 6V, pudiendo llegar a picos de casi 1ª en momentos puntuales.

Y a continuación podemos ver el funcionamiento de este según sea la señal que recibe.

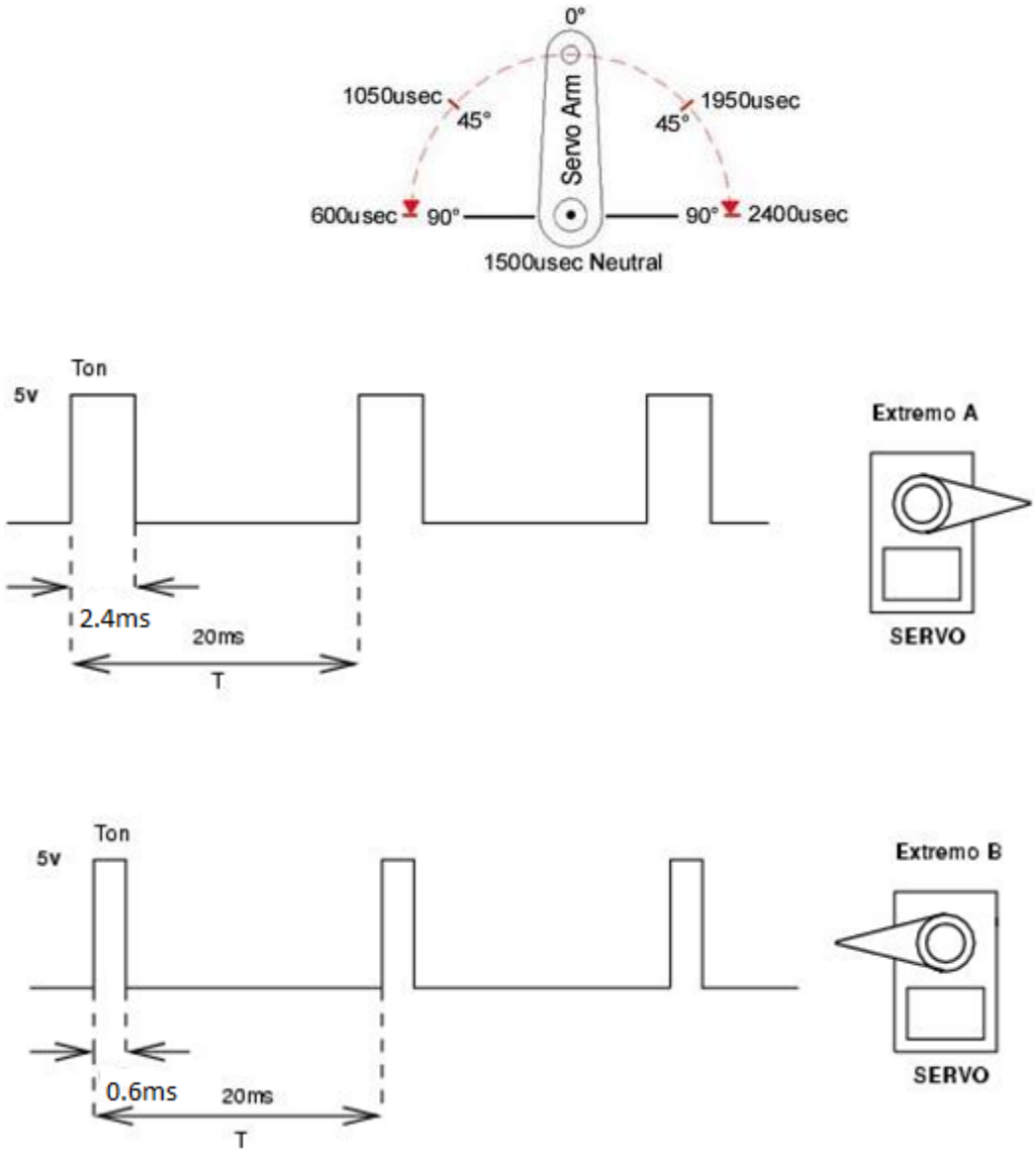


Fig. 41: Funcionamiento servo Hitec.

Para posicionar el servo hay que aplicar una señal periódica, de 50Hz (20ms de periodo). La anchura del pulso determina la posición del servo. Si la anchura es de 2.4ms, el servo se sitúa en un extremo y si la anchura es de 0.6ms se sitúa en el opuesto. Cualquier otra anchura entre 0.6 y 2.4 sitúa el servo en una posición comprendida entre un extremo y otro. Por ejemplo, si queremos que se sitúe exactamente en el centro, aplicamos una anchura de 1.3ms.

Cuando se deja de enviar la señal, el servo entra en un estado de reposo, y por tanto se podrá mover con la mano. Mientras se le aplique la señal, permanecerá fijado en su posición, haciendo fuerza para permanecer en ella.

3.4 Sensor ultrasónico

El funcionamiento de este tipo de sensores se basa en la utilización de ondas ultrasónicas, las cuales se caracterizan porque su frecuencia supera la capacidad de audición de los humanos. El oído humano es capaz de detectar ondas sonoras de frecuencias comprendidas entre unos 20 y 20000 Hertz, a esto se le conoce como espectro audible. Toda señal sonora que se encuentre por encima de este rango, se cataloga como ultrasónica.

3.4.1 Sensor ultrasónico PING

Una de las alternativas posibles como sensor de ultrasonidos puede ser el sensor ultrasónico PING, fabricado por PARALLAX, el mismo que permite efectuar medición de distancia de objetos comprendidos entre 3cm y 3m, es muy sencillo de manejar ya que solo es necesario utilizar un terminal entrada/salida de un microcontrolador.

Fig. 42: Sensor PING.

Tabla 3. Características del sensor PING.

Voltaje de Alimentación	5Vdc
Consumo de Corriente	30 -35mA Max
Rango de Medición	2cm hasta 3m
Entrada de disparo	Pulso ascendente TTL con duración de 5us
Pulso de salida	Pulso ascendente TTL comprendido entre 115us y 18.5ms
Tiempo de espera para la medición	750us luego del disparo
Frecuencia del ultrasonido	40KHz
Tiempo de emisión del ultrasónico	200us
Tiempo mínimo de espera entre medidas	200us
Dimensiones	22x46x16 mm
Diodo led indicador de actividad	

El sensor ultrasónico PING)) tiene tres terminales el pin GND es la referencia o tierra, el pin 5V es la alimentación del sensor y el pin SIG es el terminal E/S, el cual se usa para producir el pulso de activación y recibir la medición generada, este pin se conecta directamente al terminal E/S del microcontrolador.

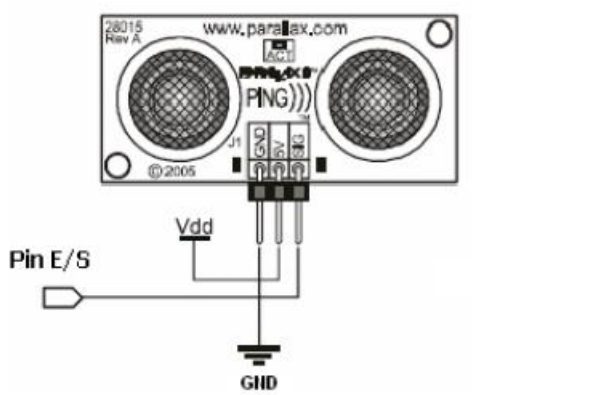


Fig. 43. Conexionado sensor PING.

El sensor ultrasónico PING trasmite una ráfaga ultrasónica, luego mide el tiempo que tarda en regresar el eco. Este eco se produce cuando las ondas sonoras golpean con un objeto ya sea fijo o móvil, siempre y cuando se encuentre dentro del rango de medición del sensor. Este sensor entrega un pulso proporcional al tiempo requerido por el ultrasónico para ir desde el emisor, golpear en el objeto y regresar al receptor.

Este sensor es una buena elección cuando se quiere realizar mediciones de objetos fijos o móviles, también tiene una gran aplicación en la robótica móvil, sistemas de seguridad o como remplazo de sistemas con sensores infrarrojos.

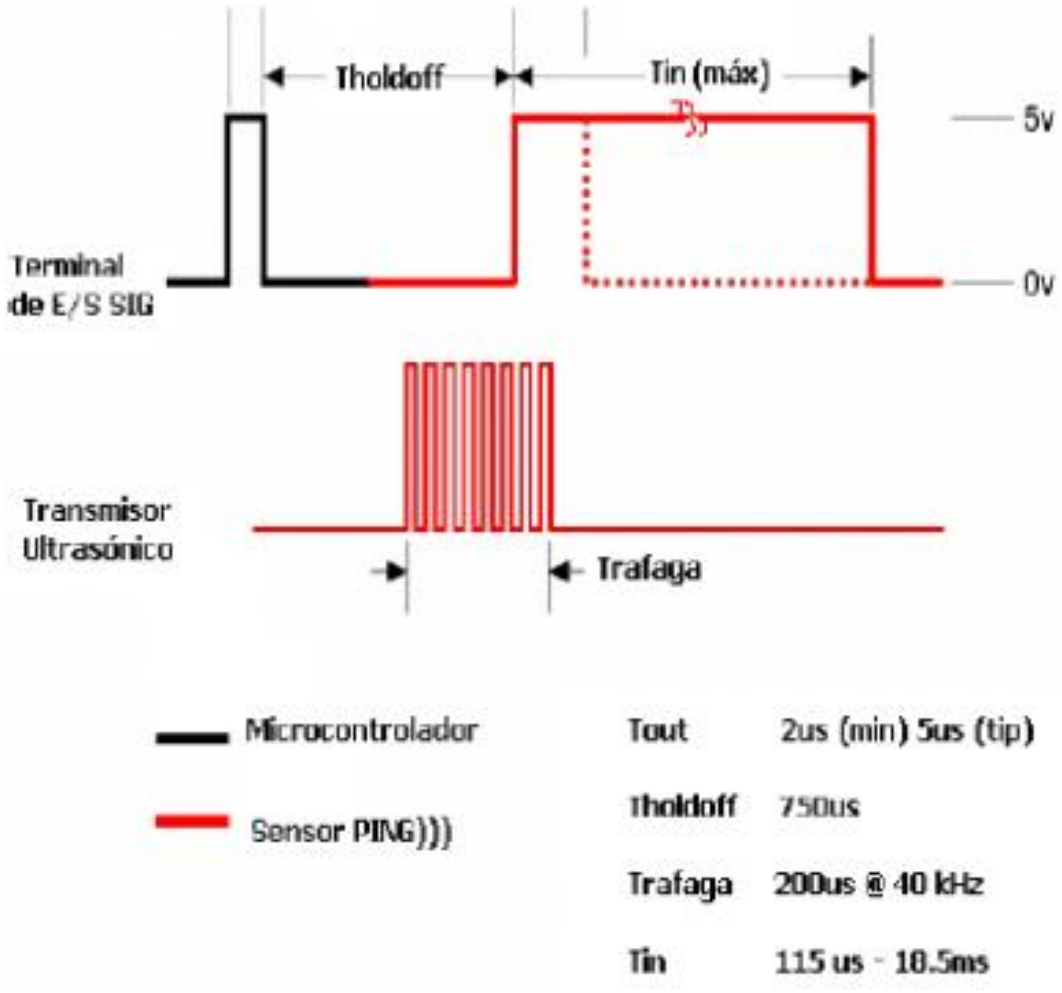


Fig. 44. Funcionamiento sensor PING.

Para empezar a trabajar con el sensor ultrasónico, el microcontrolador debe garantizar un estado en bajo en el pin SIG antes de la operación del sensor. Seguidamente se da un pulso para activar, por datos de fabricante de 5us, al terminar ese pulso el terminal E/S del microcontrolador conectado al pin SIG debe cambiar a entrada para que el PING, tome el control del mismo. El sensor activa el trasmisor ultrasónico durante unos 200us enviando una ráfaga a 40KHz, esta ráfaga viaja por el aire a una velocidad de 1239,93 Km/h y golpea al objeto que tiene al frente y se genera una señal de rebote, la cual es "escuchada" por el receptor del ultrasónico. El pin SIG se colocará en estado en alto luego de ser enviada la ráfaga de 40KHz y permanecerá en ese estado por un tiempo proporcional al tiempo de vuelo.

Para la lectura de la distancia se debe considerar que el pulso recibido tiene una duración proporcional al doble de la distancia recorrida por la onda sonora, para ello en la figura 27 se ilustra el funcionamiento del sensor, en donde se puede notar que la señal demora un tiempo T1 en alcanzar al objeto y posteriormente le toma un tiempo T2 para llegar al sensor PING, ya que las dos se propagan por el mismo medio (aire). El tiempo T1 y T2 serán iguales.

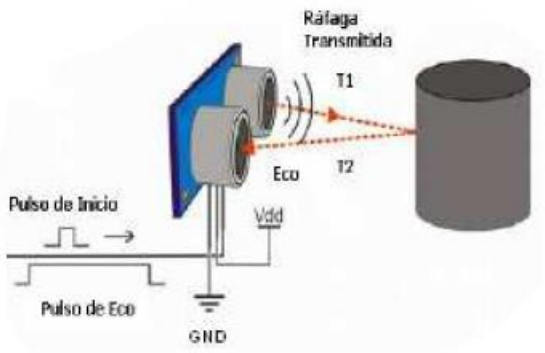


Fig. 45. Detección sensor PING.

3.5 Acelerómetro

Un acelerómetro es un instrumento de medida que proporciona lecturas de la variación de la aceleración (lineal o transversal) con el tiempo. Los valores obtenidos se expresan en fuerza g (g es la aceleración de la gravedad, 9,8 m/s2) por cada segundo.

Acelerómetro MMA7260QT

El acelerómetro utilizado para las pruebas es el MMA7260Q. Se trata de un sensor que tiene la capacidad de modificar su rango de medición de 1.5g, 2g, 4g y 6g en tres ejes. Es un sensor muy agradable con el interfaz analógico fácil. El MMA7260QT tiene partes que se alimentan a 3,3 V y ofrece una tensión de salida analógica para cada uno de los tres ejes. Esta tensión está en relación a la aceleración medida y la tensión de alimentación (proporcional). Tiene sensibilidad seleccionable por los interruptores que ya lleva la placa. Se necesita algo de hardware adicional para convertir la señal analógica a una digital utilizable que, en nuestro caso, el elemento convertidor será el propio PIC que se encargará de convertir la medida analógica a información digital mediante su convertidor analógico-digital.

El acelerómetro de bajo costo MMA7260QT contiene un filtro pasa bajos para evitar medidas erróneas y un compensador de temperatura. Incluye un modo "*sleep*" que la hace ideal para la electrónica a pilas.

Fig. 46. Acelerómetro.

3.6 Batería

Se trata de dispositivo que almacena energía eléctrica, usando procedimientos electroquímicos y que posteriormente la devuelve casi en su totalidad. Este ciclo puede repetirse por un determinado número de veces. En este caso, la batería es un elemento importante ya que se trata de un robot que debe poder funcionar en cualquier sitio, sin que sea necesaria una toma de corriente.

Esta batería debe tener una tensión máxima de 6V. Eso es a causa de que los servomotores estarán conectados directamente a ella sin ningún tipo de regulador y estos servomotores se pueden alimentar como máximo a 6V.

Una posible batería para este proyecto es la siguiente:

Fig. 47: Batería.

Se trata de una batería de Ni-MH de 2800mAh. Con esta batería y sabiéndose aproximadamente lo que consume un servomotor, se asegura un mínimo de autonomía aceptable para el robot gracias a su gran capacidad.

CAPÍTULO 4: CIRCUITO Y ESTRUCTURA IMPLEMENTADOS

4.1 Diseño del circuito de control

Para poder controlar todos los servos, y gracias a su sistema de funcionamiento, se puede ahorrar el driver o drivers que serían necesarios si se tratara de motores convencionales, por tanto, lo que hará falta en este circuito de control es simplemente el microcontrolador, el cristal de cuarzo, un regulador de tensión para estabilizar la tensión de alimentación del micocontrolador y algunos componentes pasivos más. En la figura siguiente podemos ver el esquema básico.

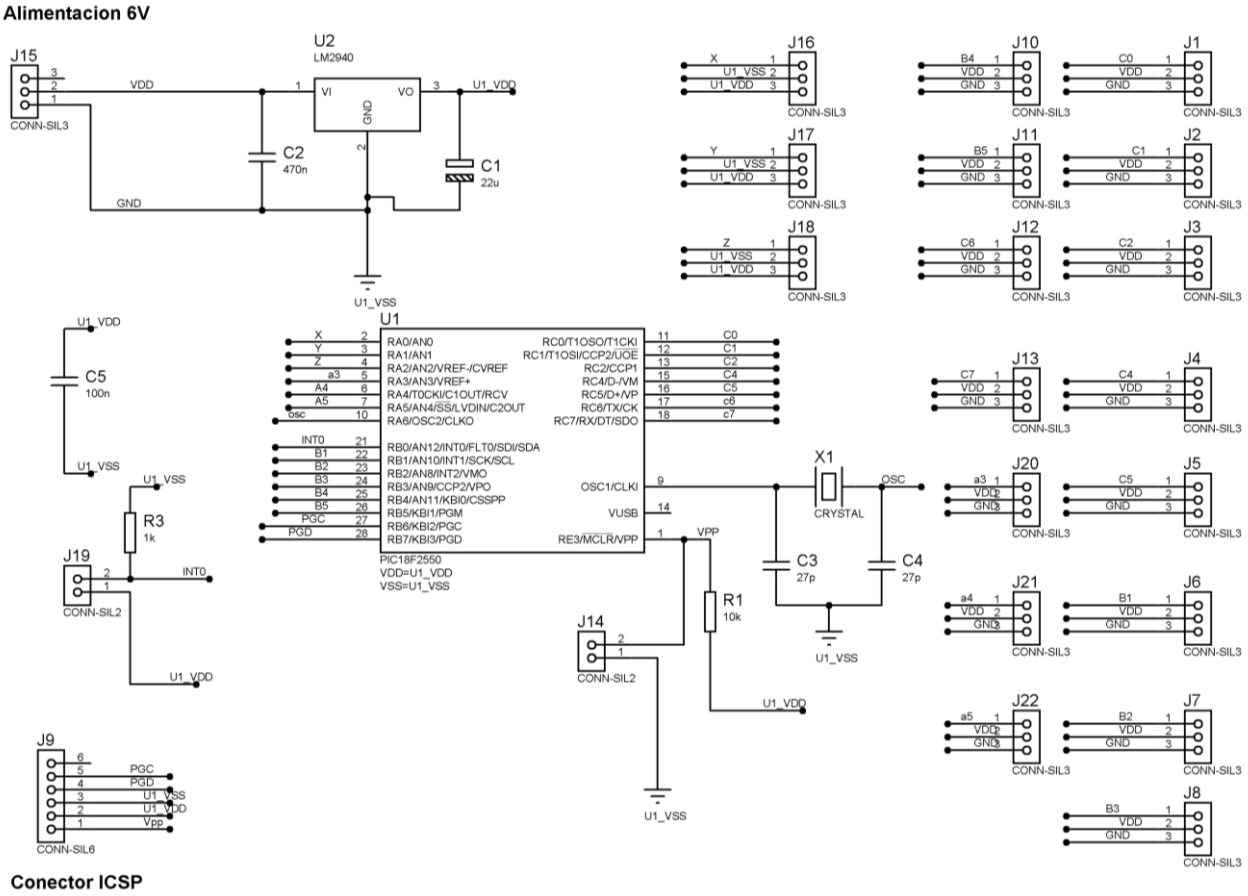


Fig. 48: Esquema del circuito de control.

En él se pueden ver claramente distintas partes

4.1.1 Regulador

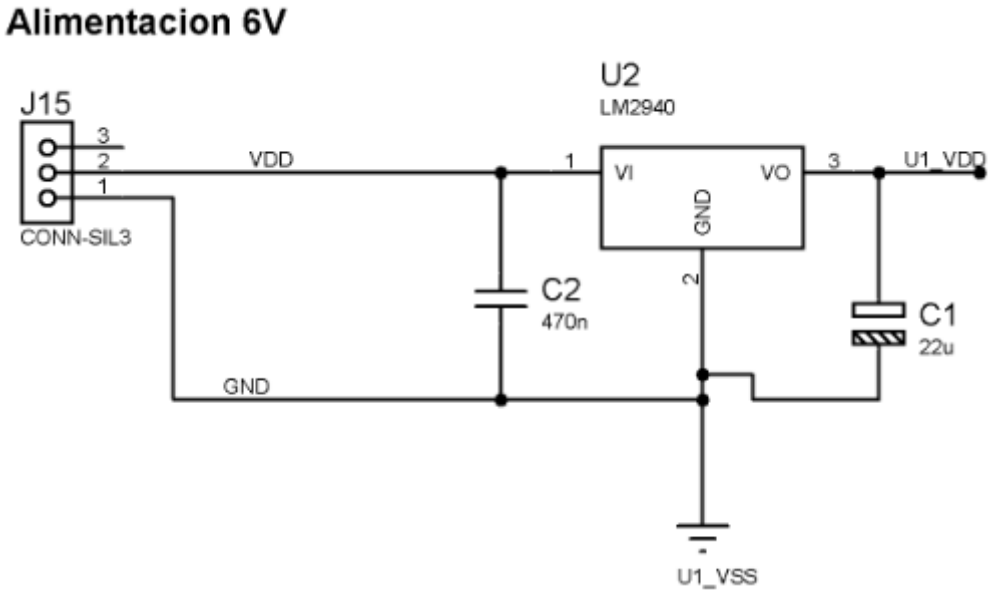


Fig. 49: Esquema del regulador.

En la parte superior izquierda de la figura 48 se encuentra el regulador LM2940. Este regulador, como ya se comentaba, se encarga de estabilizar la tensión de

alimentación a una tensión en la que el microcontrolador pueda trabajar bien. Esta tensión está establecida a 5V. Tanto en su entrada como en su salida aparecen unos condensadores que sirven para dar estabilidad, como aconseja el fabricante. Si no se pusieran existiría la posibilidad de que el LM2940 empezara a oscilar y entonces sería mucho peor que no tenerlo.

Como estos condensadores son para el regulador, a la hora de colocar la disposición de componentes en el circuito impreso se debe tener en cuenta que esos condensadores deben estar lo más próximos posible al mismo para asegurar una buena estabilización.

4.1.2 Microcontrolador

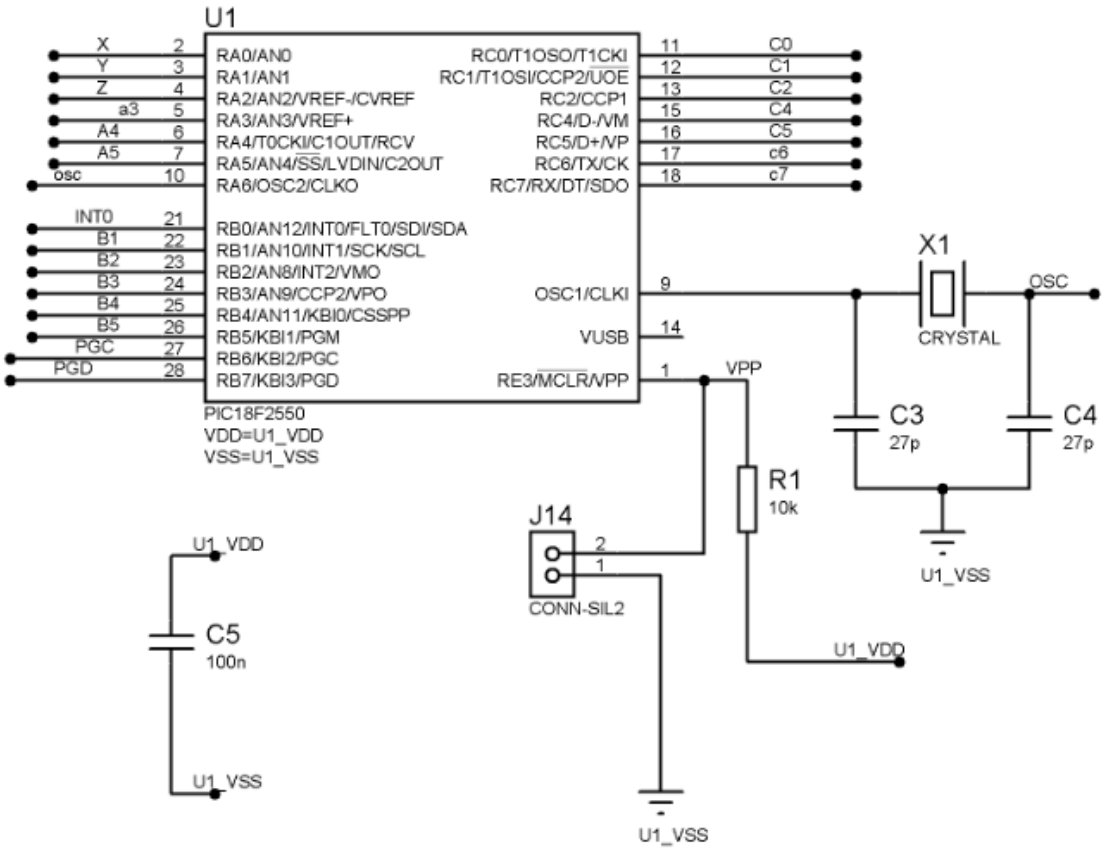


Fig. 50: Esquema del Microcontrolador.

El microcontrolador escogido para esta aplicación es el 18F2550. Se ha escogido este microcontrolador ya que, gracias a sus prestaciones en cuanto a velocidades, es muy adecuado para las tareas que tendrá que desarrollar.

La distribución que se le hace a sus pines es la siguiente.

- RA0, configurado como entrada analógica para el eje X del acelerómetro.
- RA1, configurado como entrada analógica para el eje Y del acelerómetro.
- RA2, configurado como entrada analógica para el eje Z del acelerómetro.
- RA3, pin de propósito no especificado por si hace falta para cualquier cosa o comprobación del correcto funcionamiento del programa.
- RA4, pin de propósito no especificado por si hace falta para cualquier cosa o comprobación del correcto funcionamiento del programa.

- RA5, pin de propósito no especificado por si hace falta para cualquier cosa o comprobación del correcto funcionamiento del programa.
- RA6, pin de conexión al cristal oscilador.
- RB0, pin configurado como entrada por interrupción mediante pulsador en el conector J19.
- RB1 a RB5, pines utilizados para las señales PWM hacia los servomotores.
- RB6, pin reservado para la programación *in circuit,* subministra la señal de sincronización de la programación (PGC).
- RB7, pin reservado para la programación *in circuit,* subministra la señal de datos de la programación (PGD).
- RC0 a RC5, pines utilizados para las señales PWM hacia los servomotores.
- RC6 y RC7, pin de propósito no especificado por si hace falta para cualquier cosa o comprobación del correcto funcionamiento del programa.
- OSC1/CLK, pin de conexión del otro extremo del oscilador de cuarzo.
- VUSB, pin reservado para conexiones USB. No usado en esta aplicación.
- RE3/noMCLR/VPP, pin configurado para doble funcionalidad en este caso. Por una parte y mediante un pulsador, sirve para aplicarle un *reset* al microcontrolador y por otra sirve para pasar al modo programación mediante la aplicación de la tensión de programación en el momento del grabado. (Aproximadamente 13V).

Como se puede observar en la figura anterior, aparece el condensador C5 de 100n. Este condensador es sumamente importante para que no aparezcan ruidos e interferencias hacia dentro del chip. En pruebas de laboratorio se pudo comprobar que la ausencia de este componente provocaba comportamientos anormales en el funcionamiento final.

Al ser un condensador que va junto con el microcontrolador, se debe tener en cuenta que ambos se encuentren lo más próximo posibles a la hora de diseñar el circuito impreso.

4.1.3 Conexión ICSP

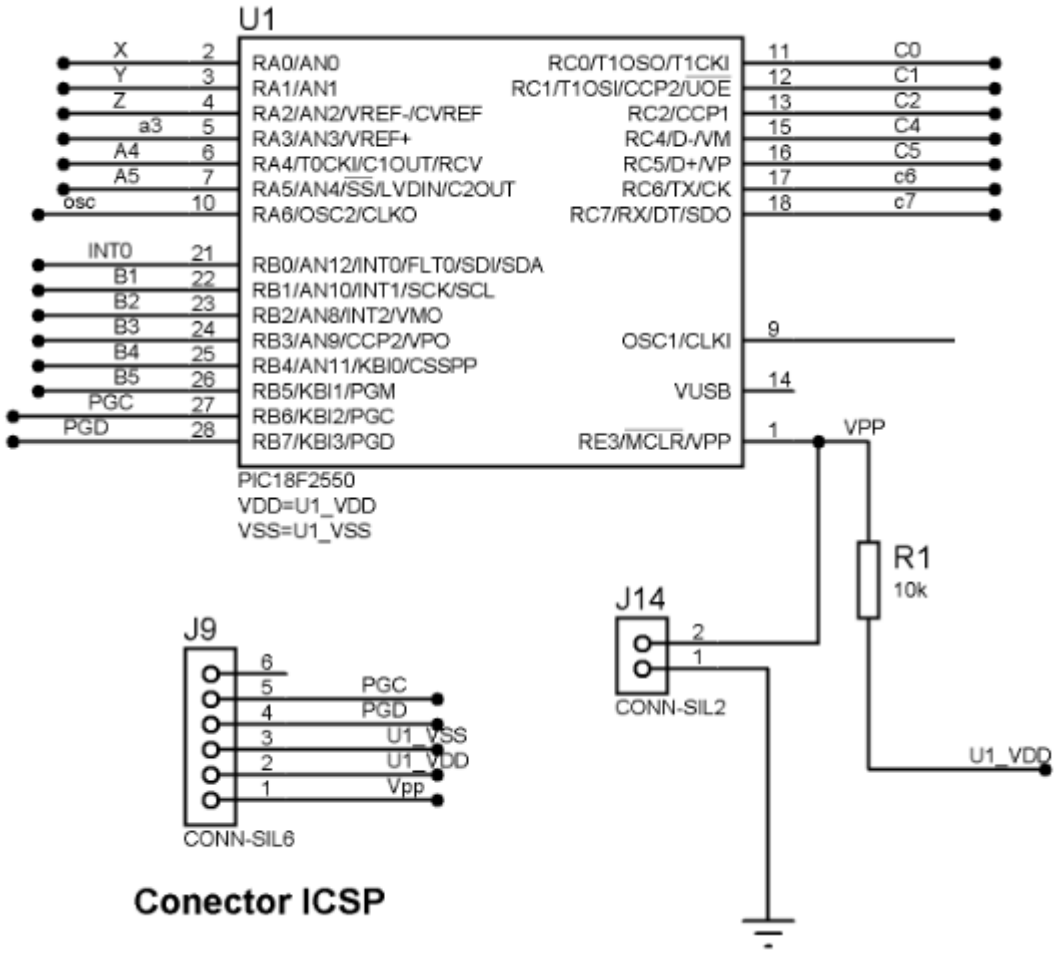


Fig. 51: Esquema del microcontrolador con la conexión ICSP.

Para poder programar el pic sin tener que conectarlo y desconectarlo cada vez que queremos realizar alguna modificación del programa, cuenta con un sistema de programación llamado *in circuit serial programming* (ICSP) que consiste en programar el microcontrolador solo con dos pines, uno de datos (PGD) y otro de sincronización (PGC). A más a más es necesario alimentar a 13V el pin VPP para que se entre en el modo "programación". A causa de esos 13V necesarios, i a que en funcionamiento normal deben haber siempre un nivel alto en el pin noMCLR, se debe poner una resistencia entre la alimentación del circuito y dicho pin. De lo contrario, a la hora de programar aparecería, por un lado 5V de alimentación fijos, y por el otro lado 13V de programación, y las dos tensiones en paralelo. Aparecería un cortocircuito y gracias a la resistencia de R1 10KΩ se solventa el problema.

4.1.4 Conectores de los servomotores y otros dispositivos

En el esquema también se puede observar que aparecen un grupo de conectores con tres conexiones, se trata de los conectores J1 a J8, J10 a J13, J16 a J18 y J20 a J22. Todos estos sirven para poder conectar los distintos componentes externos a la placa, como por ejemplo los servomotores y el acelerómetro.

Por una parte, los conectores de los servomotores tienen VDD (tensión directa de alimentación) en la conexión central y tanto masa como la conexión de señal de control queda en los extremos. Se trata J1 a J8 y J10 y J11.

Por otra parte, tenemos los conectores J16 a J18 que están destinados a la conexión del acelerómetro. Se puede comprobar cómo estos tienen una configuración distinta a los anteriores. La conexión 1 vemos que es la de la señal recibida pero las conexiones 2 y 3 varían considerablemente. En la 2 observamos que en este casi queda conectada a masa mientras que la 3 se conecta a la alimentación de 5V obtenida del regulador.

A más a más, quedan los conectores J12 a J13 y J20 a J22 que van conectados a el resto de pines sobrantes. Estos también cuentan con conexión a la alimentación por si hiciera falta del mismo modo que los conectores de los servomotores.

4.2 Diseño de la placa impresa PCB

Una vez definida la distribución de los pines y el esquema completo que se va a necesitar, se procede al diseño del circuito impreso teniendo en cuenta todos los aspectos mencionados en el apartado 5.1. El circuito diseñado constará de dos caras para poder conseguir un tamaño suficientemente reducido para que no sobresalga demasiado del conjunto. El diseño final es el de la figura siguiente.

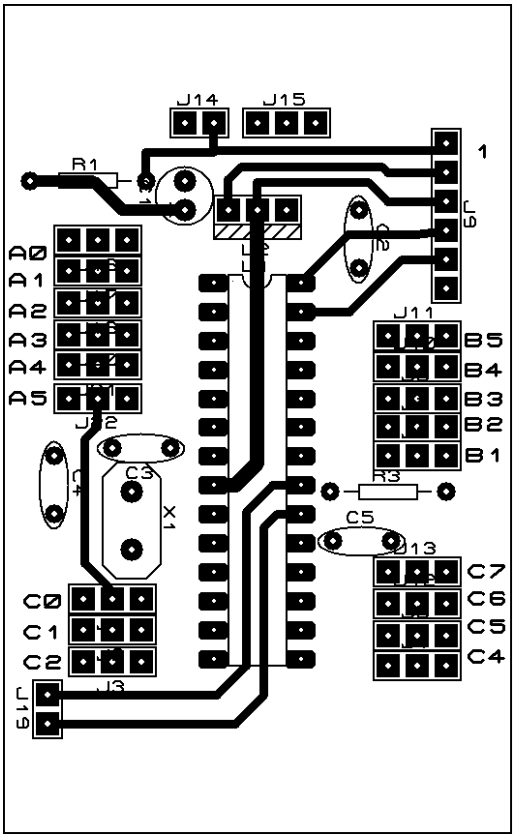


Fig. 52: Diseño del circuito impreso. Cara superior.

Podemos observar como los componentes siguen las especificaciones de su posicionamiento ya marcadas anteriormente. Esta cara del circuito es la llamada cara superior o cara de componentes y en ella van situados los componentes

utilizados y las pistas superiores. Por otro lado tenemos la cara inferior que se muestra en la figura siguiente.

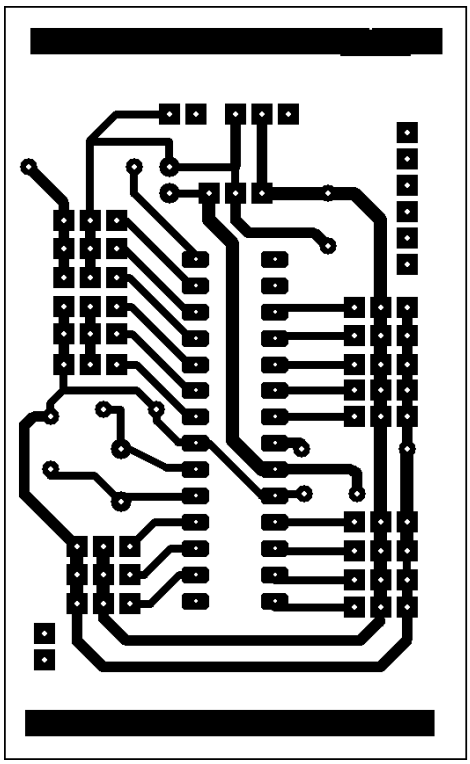


Fig. 53: Diseño del circuito impreso. Cara inferior.

Seguidamente se puede observar el circuito impreso en 3D para poder observar, antes de montarlo, cómo sería su resultado final.

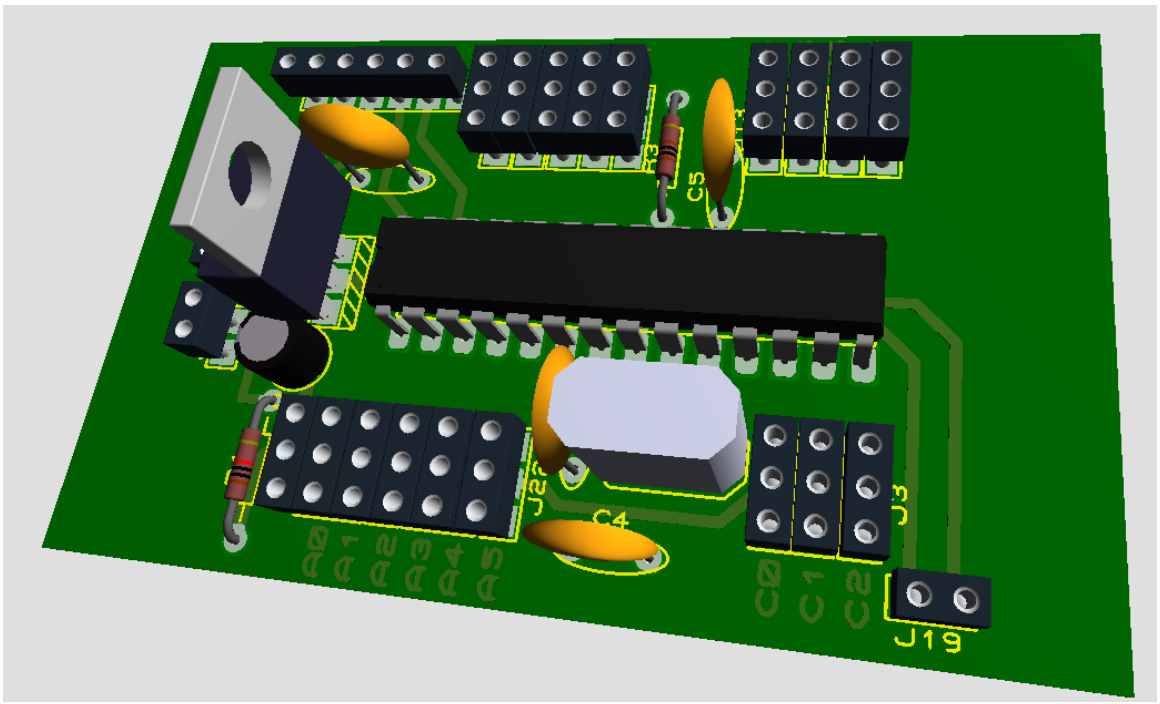


Fig. 54: Diseño del circuito impreso en 3D.

Finalmente, después de haber construido el circuito impreso el resultado es el siguiente.

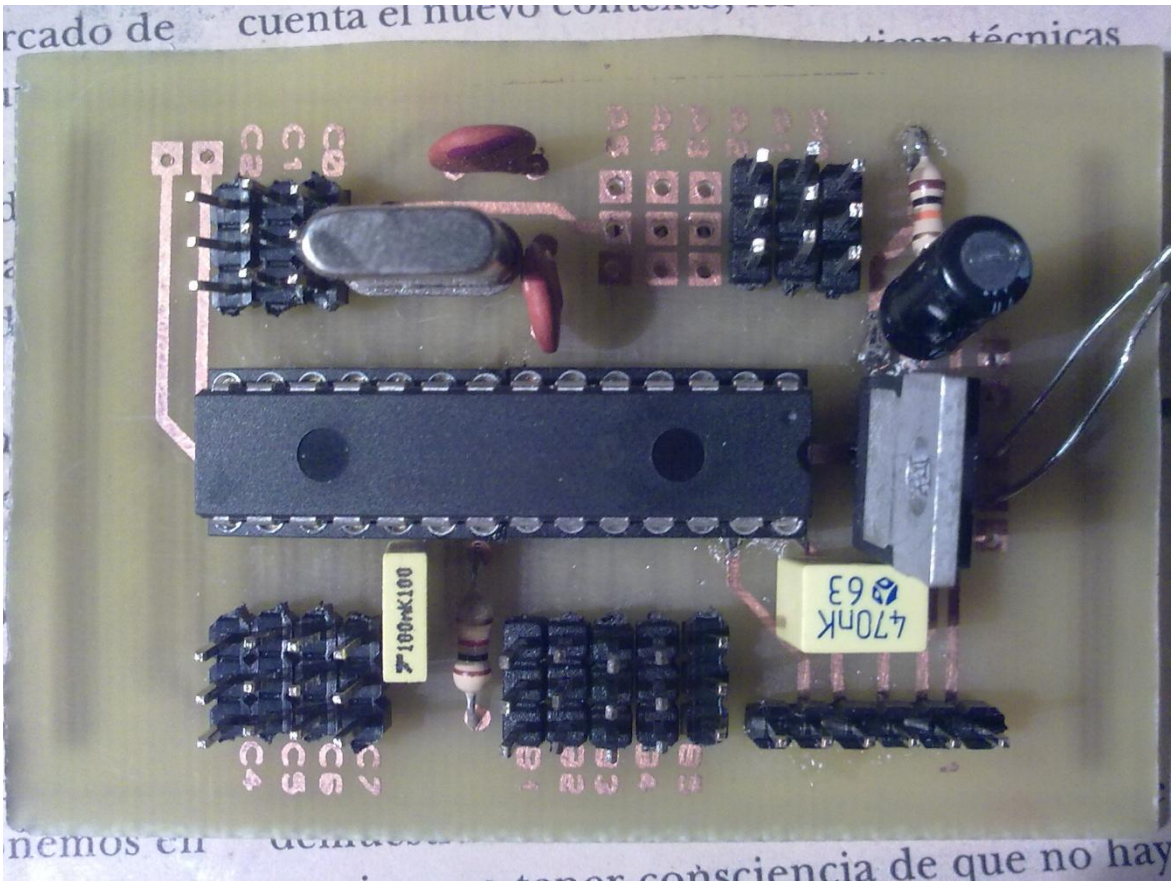


Fig. 55: Circuito impreso (Cara superior).

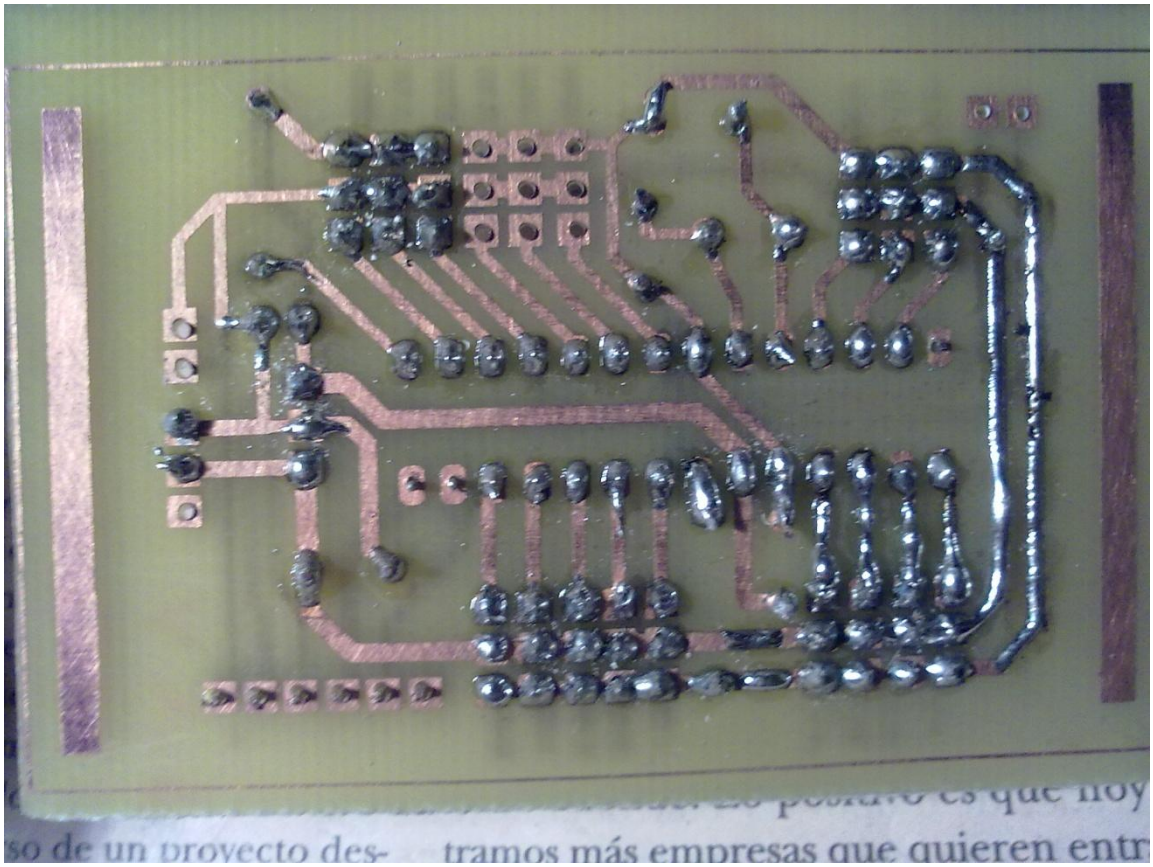


Fig. 56: Circuito impreso (Cara inferior).

4.3 Estructura del robot

4.3.1 Piezas

Para conseguir la parte mecánica del robot, lo que se hace es aprovechar piezas de la casa Lynxmotion. Esta empresa se dedica a piezas de robot de propósito general para poder fabricar brazos y piernas robóticas y así cada usuario se puede construir su robot a medida como por ejemplo el robot de la figura siguiente:

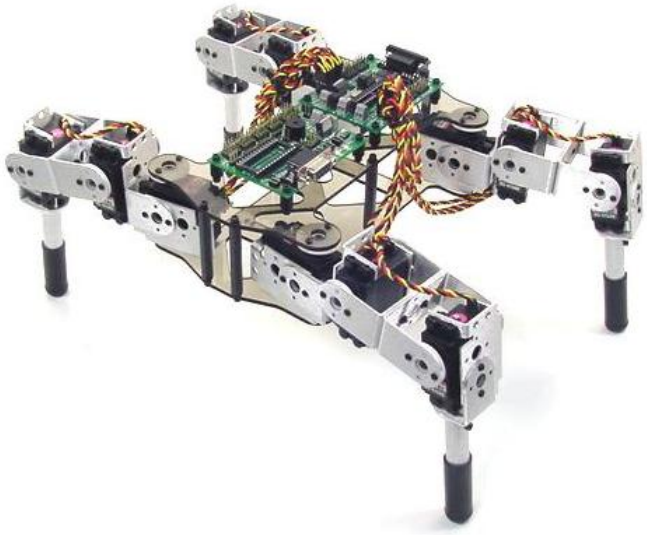


Fig. 57: Robot cuadrúpedo.

O como por ejemplo el de la figura siguiente.

Fig. 58: Robot bípedo.

En nuestro caso y a partir de piezas de esta casa lo que se hará es construir unas piernas con 5 grados de libertad cada una para no tener problemas en cuanto a posibilidades de movimiento. Las piezas están construidas en aluminio, cosa que reduce el peso del conjunto y dota de rigidez a la estructura. Las podemos ver en las de las figuras siguientes.

Fig. 59: Pieza ASB-04.

Fig. 60: Pieza ASB-09.

Fig. 61: Pieza BT-01.

Como se puede observar, estas piezas están expresamente pensadas para la utilización de servomotores. Disponen de puntos de sujeción acorde con los puntos de sujeción de los servos y a más a más disponen de unos pequeños cojinetes para el ensamblado entre las piezas para minimizar el rozamiento.

A continuación vemos los planos acotados de ASB-04 y ASB-06.

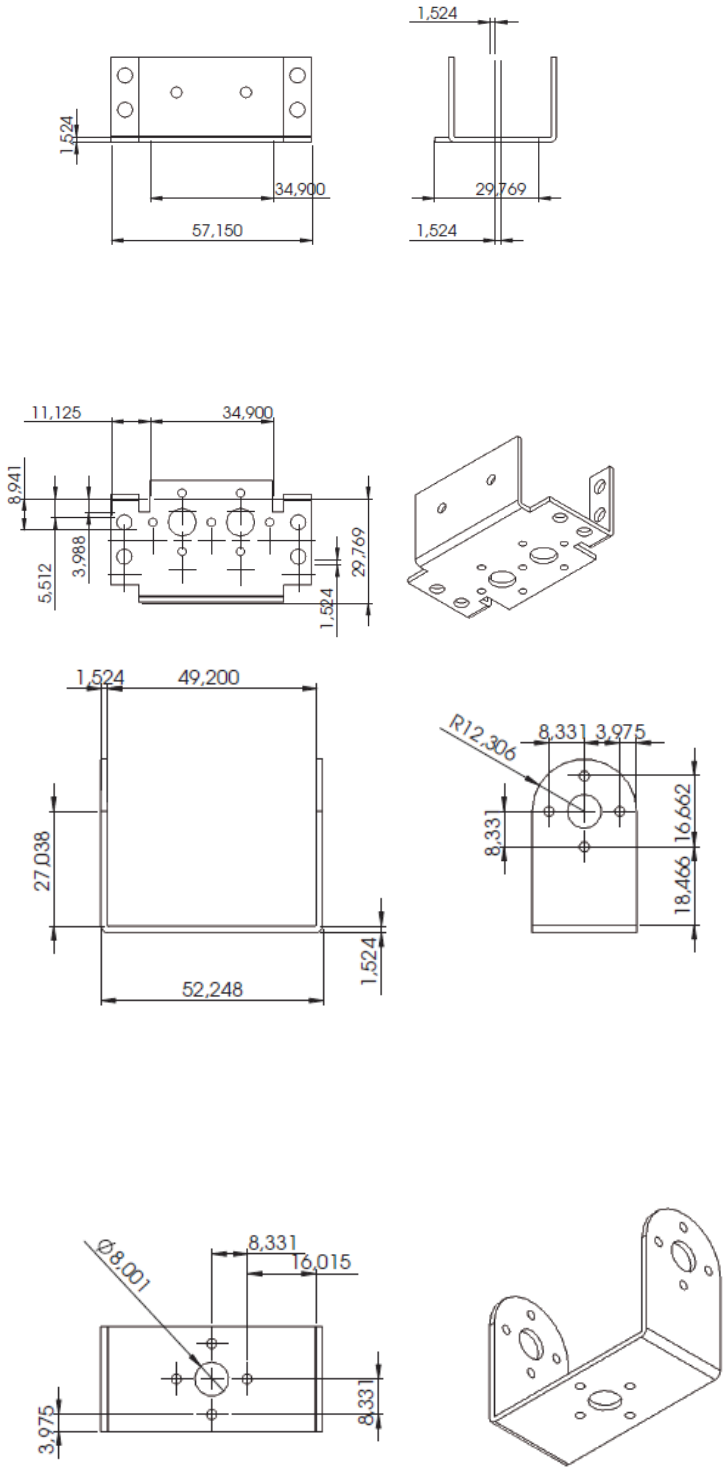


Fig. 62: Acotaciones piezas ASB-04 y ASB-09.

4.3.2 Montaje

El resultado de juntar estas piezas de forma que se creen piernas mediante ellas se observa en las figuras siguientes.

Fig. 63: Proceso de montaje.

Fig. 64: Proceso de montaje.

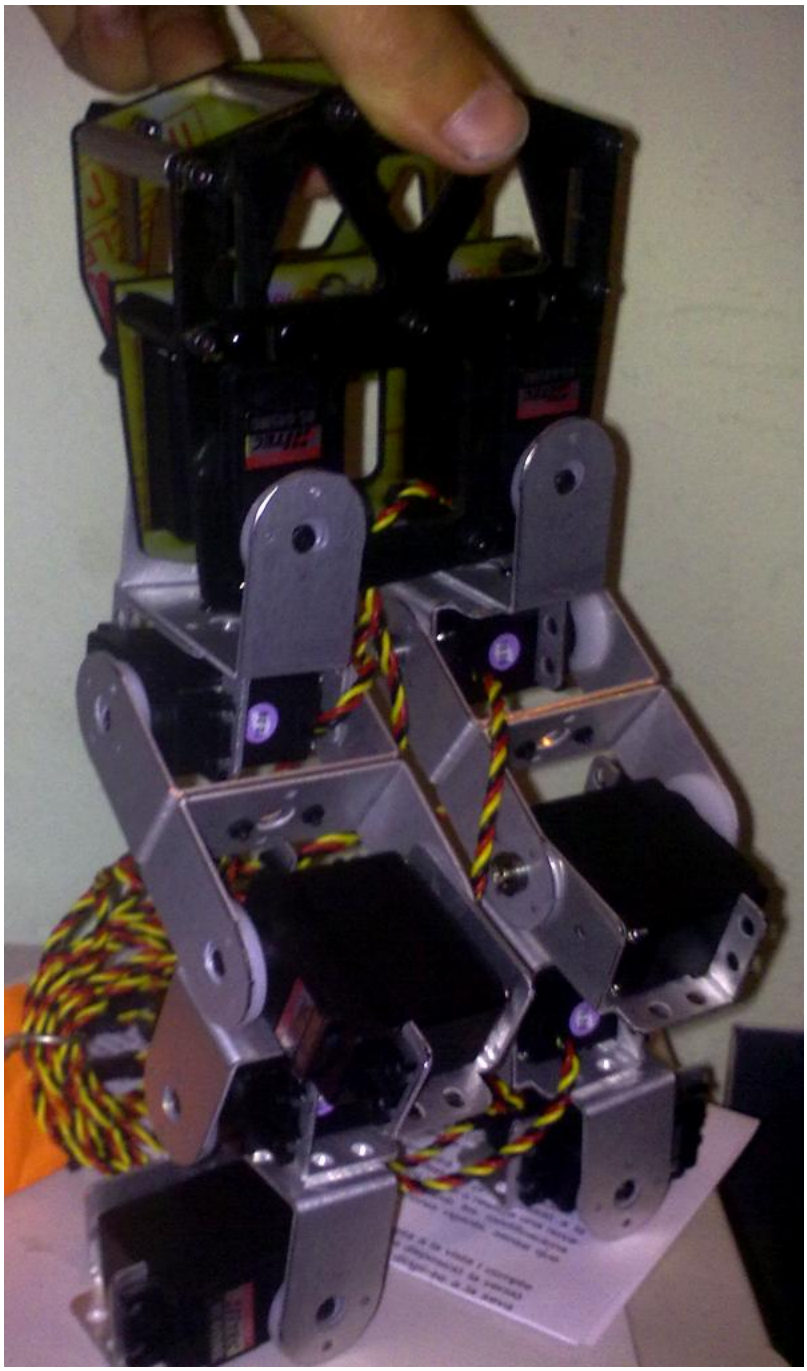


Fig. 65: Proceso de montaje.

Finalmente, debemos conectar los servomotores y los demás periféricos a la placa de control. Una vez hecho eso programado el robot para que quede en posición estática queda como en las figuras siguientes.

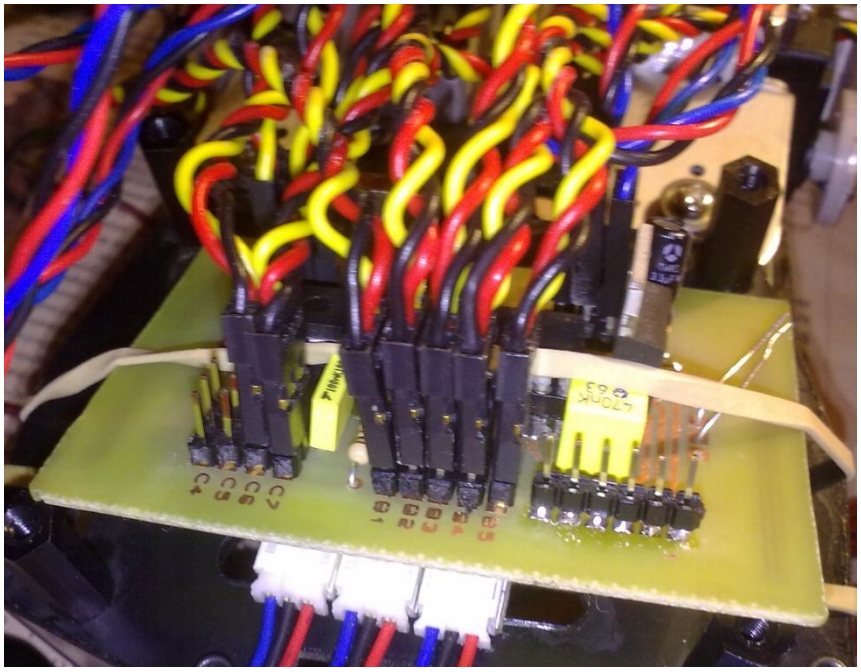


Fig. 66: Conexionado a la placa de control.

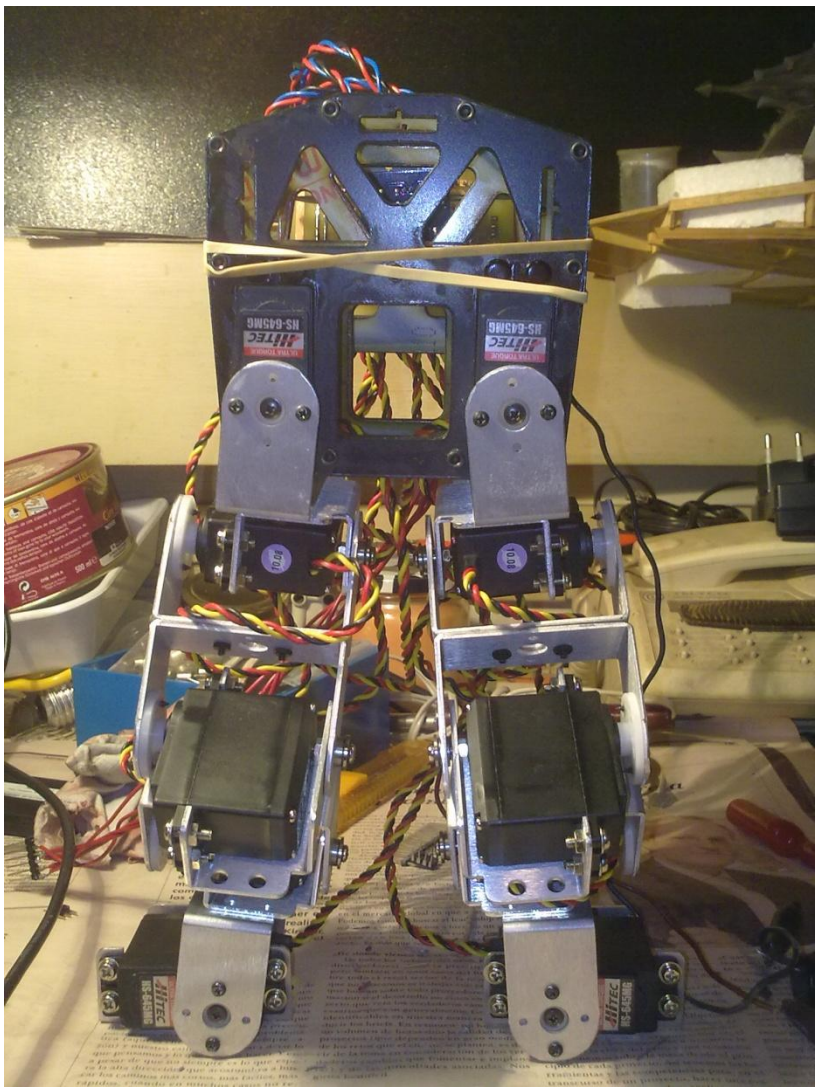


Fig. 67: Finalizado.

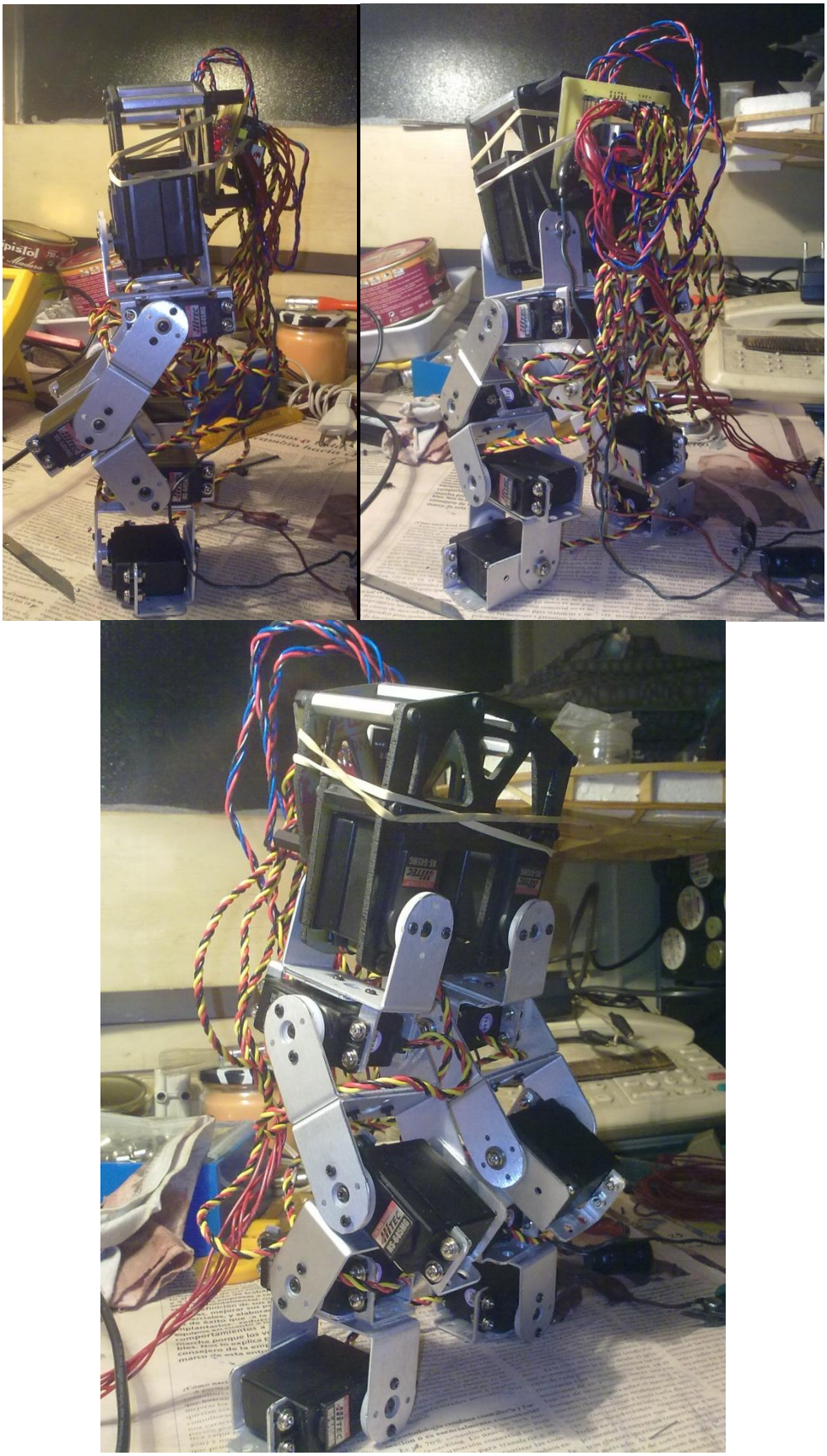


Fig. 68: Finalizado.

CAPÍTULO 5: PROGRAMA DESARROLLADO

5.1 Control de servos

Aunque existen soluciones interesantes en el mercado, uno de los primeros problemas es el de controlar múltiples servos.

La idea de cómo controlarlos pasó por distintas fases, y en cada una de ellas se puede observar una mejora respecto la anterior. La resultante sobre la que finalmente se basa el control es aparentemente sencilla de entender, pero no tanto de realizar, sobre todo dependiendo de las posibilidades del microcontrolador que se utilice.

Lo primero que se debe tener en cuenta, es que para controlar un servo, hay que enviarle cada 20ms un pulso de duración entre aproximadamente 0,6ms a 2,4ms. Es decir, que dentro de esos 20ms, deben dedicarse entre 0,6ms a 2,4ms de tiempo de nuestro procesador a atender al servo y a poner/quitar el pulso que necesita. Con lo cual se tendrá en el peor de los casos (cuando hay que enviarle al servo un pulso de 2.3ms de duración), tendremos 20 ms – 2,4 ms = 17,7ms el procesador libre para hacer otras tareas.

Las soluciones más sencillas que se encuentran para controlar más de un servo, son las que los atienden de modo secuencial. Es decir, que si el mirocontrolador tiene que controlar 4 servos, va a dedicar entre 0,6 y 2,4 ms para el primer servo, entre 0,6 y 2,4ms para el segundo, etc… hasta atender, por ejemplo a cuatro servos. Con lo cual, en el peor de los casos (2,4ms por servo) tendremos que emplear 2,4ms x 4 = 9,6 ms para atender a los 4 servos. Y nos quedan 20ms -9,6ms = 10,4ms para que el microcontrolador pueda dedicarse a otras tareas.

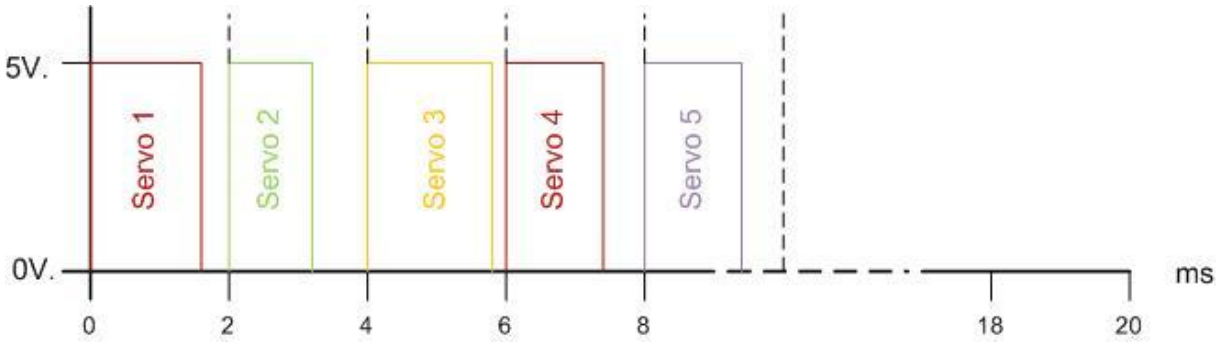


Fig. 69: Distribución de pulsos.

Usando este sistema, la cosa se complica cuantos más servos se quieran controlar. Y si además se requiere que nuestro microcontrolador reciba las ordenes de posicionamiento de los servos de un dispositivo externo, se debe añadir un margen, o tiempo mínimo para que el microcontrolador se dedique a esta tarea de recepción de información (y quizá a otras tareas).
Los 20 ms divididos entre 2,4 ms para cada servo, daría un total de 8 servos que se podrían controlar como máximo, y esto sería casi sin dejar tiempo libre el procesador para ninguna otra tarea. Luego para tener un mínimo de margen, lo máximo serían 7 servos para controlar y restarían 3,2 ms para otras tareas del procesador. Tiempo muy reducido para según qué aplicaciones.
Es cierto, que se pueden utilizar interrupciones para que durante esos 2,4ms de duración del pulso que se envía a un servo, el procesador se dedique a otras tareas, pero para eso se necesitaría que esas otras tareas fuesen "interrumpibles" , cosa que las rutinas de tx rx serie no siempre lo son. Por tanto, tampoco parece demasiado viable.
Es por eso que, a raíz de los problemas comentados anteriormente, se plantea la siguiente idea: procesar todos los servos a la vez.
Esto sería no de manera secuencial, sino dentro del margen de 2,4 ms de tiempo que se necesita para cada servo. Y de este modo se procesarían todos los pulsos de todos los servos a la vez. En la figura siguiente podemos observar la nueva propuesta.

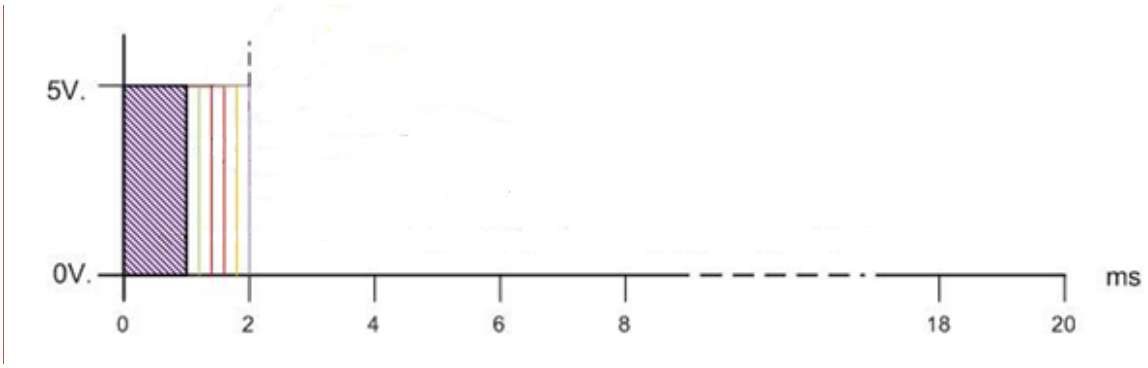


Fig. 70: Distribución de pulsos.

Con este nuevo planteamiento se tendrían 17,6 ms enteros para que el procesador se dedicase a hacer cualquier otra tarea, que no sería interrumpida, y

durante 2,4 ms, estuviese pendiente de x servos. Siendo en paralelo, no sería demasiado problema que esa cifra fuese mayor que 8.
Ahora la cuestión está en cómo hacer que se procesen todos los servos a la vez con los recursos que nos da el pic.
Tenemos ya algunas ideas para el procesado de los servos.
Todos los servos deben recibir el cambio de pulso de 0 a 1 en el milisegundo 0. Entonces lo que se debe hacer es, una vez se conocen los anchos de los pulsos de cada servomotor, se deben mirar uno por uno cada vez para saber cuál es el servo al que lo toca ser desconectado su pulso. Planteada esta idea se ve rápidamente que no es la mejor solución ya que para poder conseguir eso, el microprocesador necesario sería de una velocidad de procesamiento extremadamente elevada para poder mirar cada vez todos los servos y comprobar si se debe bajar el pulso o no.
Visto el problema de velocidad que se plantea, puede ser fácilmente solucionado usando un algoritmo de ordenación que se encargue de ordenar todos los servos por orden de tiempo de pulso. De esta forma, al tener todos los tiempos ordenados ya se sabe en todo momento cual va a ser el próximo servo al que se le debe bajar el pulso, así que ya no es necesario observarlos todos, sino solo uno. Por tanto, lo que se debería hacer es una rutina a parte de la del control de servos para ordenarlos por tiempos.
Este planteamiento ya abre muchas más posibilidades en cuanto a microcontroladores que puedan desarrollar este nuevo algoritmo. Pero este algoritmo todavía es mejorable.
Sabemos que todos los servomotores tienen un tiempo mínimo de ancho de pulso y, siguiendo con el ultimo algoritmo planteado, durante ese tiempo el microcontrolador lo único que hace es esperar, pues bien, la nueva idea que se plantea es la de aprovechar ese tiempo. El resultado es que si se aprovecha ese tiempo para trabajar en vez de esperarse lo que obtenemos es que se quita faena al microcontrolador en el tiempo restante al del control de servos, o sea, los 17,6ms).
Durante ese tiempo lo que se hará es la rutina de ordenación de los pulsos de los servos, por tanto, ya no ocupa tiempo fuera de la rutina de control de estos.
El resultado final que obtenemos a partir de las deducciones anteriores se puede observar en la figura siguiente:

Fig. 71: Distribución de pulsos.

Y a continuación podemos ver como quedaría el algoritmo:

Al entrar en la rutina de control de los servos utilizar el tiempo común a todos ellos para poner a nivel alto todos los pines asignados para esa rutina en el milisegundo 0.
Inmediatamente después crear una tabla ordenada de anchos de pulso, estando ordenado por ejemplo de menor a mayor ancho de pulso. Habría que utilizar un algoritmo que lo permitiese en función de la cantidad de servos.
A partir del milisegundo 0,6 y hasta el milisegundo 2,4, todos los servos deben ser cambiados de estado de 1 a 0 de nuevo, individualmente. Excepto en el caso de que dos o más servos requieran el mismo ancho de pulso. En ese caso esos dos o más servos deberán ser cambiados de estado a la vez. Para procesar los anchos de pulso de cada servo, se deberá recorrer la tabla ya ordenada anteriormente de menor a mayor, que será el orden en que se deberá modificar cada pin del microcontrolador, y se irá poniendo a nivel bajo a los servos correspondientes en sus momentos correspondientes.

5.2 Algoritmos de ordenamiento

Existen distintos tipos de algoritmos de ordenación, entre ellos, los más utilizados son los siguientes.

5.2.1 Ordenamiento burbuja

La Ordenación de burbuja (Bubble Sort en inglés) es un sencillo algoritmo de ordenamiento. Funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada. Este algoritmo obtiene su nombre de la forma con la que suben por la lista los elementos durante los intercambios, como si fueran pequeñas "burbujas". También es conocido como el método del intercambio directo. Dado que solo usa comparaciones para operar elementos, se lo considera un algoritmo de comparación, siendo el más sencillo de implementar.

Una manera simple de expresar el ordenamiento de burbuja en pseudocódigo es la siguiente:

$$
\begin{aligned}
&\textit{procedimiento } \textit{DeLaBurbuja } (a_0, a_1, a_2, \ldots, a_{(n-1)}) \\
&\textit{para } i \leftarrow 2 \textit{ hasta } n \textit{ hacer} \\
&\quad \textit{para } j \leftarrow 0 \textit{ hasta } n - i \textit{ hacer} \\
&\quad\quad \textit{si } a_{(j)} > a_{(j+1)} \textit{ entonces} \\
&\quad\quad aux \leftarrow a_{(j)} \\
&\quad\quad a_{(j)} \leftarrow a_{(j+1)} \\
&\quad\quad a_{(j+1)} \leftarrow aux \\
&\quad\quad \textit{fin si} \\
&\quad \textit{fin para} \\
&\textit{fin para}
\end{aligned}
$$

fin procedimiento

Este algoritmo realiza el ordenamiento de una lista a de n valores, en este caso de n términos numerados del 0 al n-1, consta de dos bucles anidados uno con el índice i, que da un tamaño menor al recorrido de la burbuja en sentido inverso de 2 a n, y un segundo bucle con el índice j, con un recorrido desde 0 hasta n-i, para cada iteración del primer bucle, que indica el lugar de la burbuja.

La burbuja son dos términos de la lista seguidos, j y j+1, que se comparan, si el primero es menor que el segundo sus valores se intercambian.

Esta comparación se repite en el centro de los dos bucles, dando lugar a la postre a una lista ordenada, puede verse que el número de repeticiones sola depende de n, y no del orden de los términos, esto es, si pasamos al algoritmo una lista ya ordenada, realizara todas las comparaciones exactamente igual que para una lista no ordenada, esta es una característica de este algoritmo, luego verano una variante que evita este inconveniente.

Para comprender el funcionamiento, veamos un ejemplo sencillo:

Tenemos una lista de números que hay que ordenar:

$$a = \{55, 86, 48, 16, 82\}$$

$$a_4 = 82$$
$$a_3 = 16$$
$$a_2 = 48$$
$$a_1 = 86$$
$$a_0 = 55$$

Podemos ver que la lista que tiene cinco términos, luego:

$$n = 5$$

El índice **i** hará un recorrido de **2** hasta **n**:

$$para\ i \leftarrow 2\ hasta\ n\ hacer$$

Que en este caso será de 2 a 5. Para cada uno de los valores de **i**, **j** tomara sucesivamente los valores de **0**hasta **n-i**:

$$para\ j \leftarrow 0\ hasta\ n - i\ hacer$$

Para cada valor de **j**, obtenido en ese orden, se compara el valor del índice **j**con el siguiente:

$$si\ a_{(j)} > a_{(j+1)}\ entonces$$

Si el termino **j** es menor, en su caso podría se mayor, que el termino **j+1**, los valores se permutan, en caso contrario se continúa con la iteración.

Para el caso del ejemplo, tenemos que:

$$n = 5$$

Para la primera iteración del primer bucle:

$$i = 2$$

j tomara los valores de **0** hasta **3**:

$para\ j \leftarrow 0\ hasta\ 3\ hacer$

Cuando **j** vale **0**, se comparan $a_0\ a_1$, el 55 y el 86, dado que 55 < 86 no se permutan el orden.

Ahora **j** vale **1** y se comparan $a_1\ a_2$ el 86 y el 48 Como 86 > 48, se permutan, dando lugar a una nueva lista.

	$j=0$	$j=1$	$j=2$	$j=3$	
a_4	82	82	82	$\rightarrow 82$	86
a_3	16	16	$\rightarrow 16$	$\rightarrow 86$	82
a_2	48	$\rightarrow 48$	$\rightarrow 86$	16	16
a_1	$\rightarrow 86$	$\rightarrow 86$	48	48	48
a_0	$\rightarrow 55$	55	55	55	55

Se repite el proceso hasta que **j** valga **3**, dando lugar a una lista parcialmente ordenada, podemos ver que el termino de mayor valor esta en el lugar más alto.

Ahora **i** vale **3**, y **j** hará un recorrido de **0** a **2**.

Primero **j** vale **0**, se comparan $a_0\ a_1$, el 55 y el 48, como 55 > 48 se permutan dando lugar a la nueva lista.

Para j = 1 se compara el 55 con el 16 y se cambian de orden.

	$j=0$	$j=1$	$j=2$	
a_4	86	86	86	86
a_3	82	82	$\rightarrow 82$	82
a_2	16	$\rightarrow 16$	$\rightarrow 55$	55
a_1	$\rightarrow 48$	$\rightarrow 55$	16	16
a_0	$\rightarrow 55$	48	48	48

Para j = 2 se compara el 55 y el 82 y se dejan como están, finalizando el bucle con una lista mejor ordenada, puede verse que los dos valores más altos ya ocupan su lugar. No se ha realizado ninguna comparación con el término cuarto, dado que ya se sabe que después del primer ciclo es el mayor de la lista.

El algoritmo consiste en comparaciones sucesivas de dos términos consecutivos, ascendiendo de abajo a arriba en cada iteración, como la ascensión de las burbujas de aire en el agua, de ahí el nombre del procedimiento, en la primera iteración el recorrido ha sido completo, en el segundo se ha dejado él último término, al tener ya el mayor de los valores, en los sucesivos sé ira dejando re realizar las ultimas comparaciones, como se puede ver.

Ahora ya **i** vale 4 y **j** recorrerá los valores de 0 a 1.

Cuando **j** vale 0, se comparan $a_0\ a_1$ esto es el 48 y el 16 dado que 48 es mayor que 16 se permutan los valores, dando lugar a una lista algo más ordenada que la anterior, desde esta nueva ordenación, **j** pasa a valer 1, con lo que se comparan los términos $a_1\ a_2$ el 48 y el 55 que quedan en el mismo orden.

	$j=0$	$j=1$	
a_4	86	86	86
a_3	82	82	82
a_2	55	$\rightarrow$ 55	55
a_1	$\rightarrow$ 16	$\rightarrow$ 48	48
a_0	$\rightarrow$ 48	16	16

En este caso la burbuja ha ascendido menos que en los casos anteriores, y la lista está ya ordenada, pero el algoritmo tendrá que completarse, realizando una última iteración.

Hay que tener en cuenta que el bucle para realiza un número fijo de repeticiones y para finalizar tendrán que completarse, aun en el caso extremo, de que la lista estaría previamente ordenada.

Por último **i** vale 5 y **j** solo puede vale 0, con lo que solo se realizara una comparación de a_0 a_1 el 16 y el 48, que ya están ordenados y se dejan igual.

Los bucles finalizan y también el procedimiento, dejando la lista ordenada.

	$j=0$	
a_4	86	86
a_3	82	82
a_2	55	55
a_1	$\rightarrow$ 48	48
a_0	$\rightarrow$ 16	16

5.2.2 Ordenamiento de burbuja bidireccional

El ordenamiento de burbuja bidireccional (*cocktail sort* en inglés) es un algoritmo de ordenamiento que surge como una mejora del algoritmo ordenamiento de burbuja.

La manera de trabajar de este algoritmo es ir ordenando al mismo tiempo por los dos extremos del vector. De manera que tras la primera iteración, tanto el menor como el mayor elemento estarán en sus posiciones finales. De esta manera se reduce el número de comparaciones aunque la complejidad del algoritmo sigue siendo O(n^2).

A continuación se muestra el pseudo-código del algoritmo:

```
Procedimiento Ordenacion_Sacudida (v:vector, tam:entero)
Variables
  i, j, izq, der, ultimo: tipoposicion;
  aux: tipoelemento;
Inicio
  //Limites superior e inferior de elementos ordenados
  izq <- 2
  der <- tam
  ultimo <- tam

  Repetir
    //Burbuja hacia la izquierda}
    //Los valores menores van a la izquierda
    Para i <- der hasta izq hacer
        Si v(i-1) > v(i) entonces
            aux <- v(i)
            v(i) <- v(i-1)
            v(i-1) <- aux
            ultimo <- i
        Fin si
    Fin_para

    izq <- ultimo+1

    //Burbuja hacia la derecha
    //Los valores mayores van a la derecha
    Para j <- izq hasta der hacer
        Si v(j-1) > v(j) entonces
            aux <- v(j)
            v(j) <- v(j-1)
            v(j-1) <- aux
            ultimo <- j
        Fin si
    Fin_para

    der <- ultimo-1

  Hasta (izq > der)
Fin
```

5.2.3 Ordenamiento por inserción

El ordenamiento por inserción (*insertion sort* en inglés) es una manera muy natural de ordenar para un ser humano, y puede usarse fácilmente para ordenar un mazo de cartas numeradas en forma arbitraria. Requiere $O(n^2)$ operaciones para ordenar una lista de n elementos.

Inicialmente se tiene un solo elemento, que obviamente es un conjunto ordenado. Después, cuando hay *k* elementos ordenados de menor a mayor, se toma el elemento *k+1* y se compara con todos los elementos ya ordenados, deteniéndose cuando se encuentra un elemento menor (todos los elementos mayores han sido desplazados una posición a la derecha). En este punto se inserta el elemento *k+1* debiendo desplazarse los demás elementos.

Ejemplo de funcionamiento.

En el siguiente ejemplo, 32 debe ser insertado entre 26 y 47, y por lo tanto 47, 59 y 96 deben ser desplazados.

```
    k+1
11 26 47 59 96 32
11 26   47 59 96
11 26 32 47 59 96
```

En la implementación computacional, el elemento *k+1* va comparándose de atrás para adelante, deteniéndose con el primer elemento menor. Simultáneamente se van haciendo los desplazamientos.

```
11 26 47 59 96 32
11 26 47 59   96
11 26 47   59 96
11 26   47 59 96
11 26 32 47 59 96
```

El algoritmo en pseudocódigo (con listas que empiezan por 0) debería ser como el siguiente:

```
algoritmo insertSort( A : lista de elementos ordenables )
   para i=1 hasta longitud(A) hacer
       index=A[i]
       j=i-1
       mientras j>=0 y A[j]>index hacer
           A[j+1] = A[j]
           j = j - 1
       fin mientras
       A[j+1] = index
   fin para
fin algoritmo
```

Aunque este algoritmo tiene un mejor orden de complejidad que el de burbuja, es muy ineficiente al compararlo con otros algoritmos como *quicksort*. Sin embargo, para listas relativamente pequeñas el orden por inserción es una buena elección, no sólo porque puede ser más rápido para cantidades pequeñas de elementos sino particularmente debido a su facilidad de programación.

5.2.4 Ordenamiento por casilleros

El ordenamiento por casilleros (*bucket sort* en inglés) es un algoritmo de ordenamiento que distribuye todos los elementos a ordenar entre un número

finito de casilleros. Cada casillero sólo puede contener los elementos que cumplan unas determinadas condiciones. En el ejemplo esas condiciones son intervalos de números. Las condiciones deben ser excluyentes entre sí, para evitar que un elemento pueda ser clasificado en dos casilleros distintos. Después cada uno de esos casilleros se ordena individualmente con otro algoritmo de ordenación (que podría ser distinto según el casillero), o se aplica recursivamente este algoritmo para obtener casilleros con menos elementos. Se trata de una generalización del algoritmo *Pigeonhole sort*. Cuando los elementos a ordenar están uniformemente distribuidos la complejidad computacional de este algoritmo es de O(n).

El algoritmo contiene los siguientes pasos:

- Crear una colección de casilleros vacíos
- Colocar cada elemento a ordenar en un único casillero
- Ordenar individualmente cada casillero
- devolver los elementos de cada casillero concatenados por orden

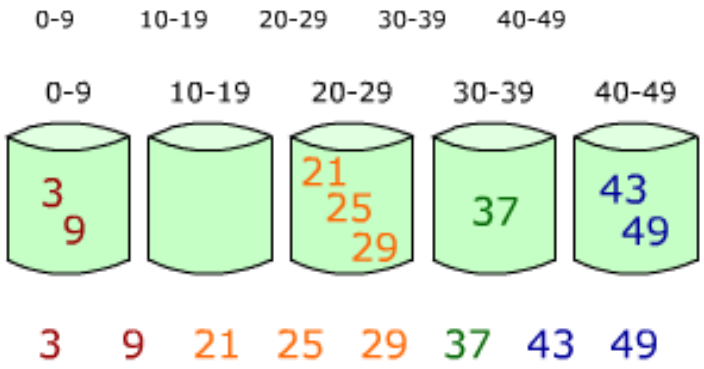


Fig. 72: Ejemplo casilleros.

A continuación vemos el pseudocódigo:

```
función bucket-sort(elementos, n)
  casilleros ← colección de n listas
  para i = 1 hasta longitud(elementos) hacer
    c ← buscar el casillero adecuado
    insertar elementos[i] en casillero[c]
  fin para
  para i = 1 hasta n hacer
    ordenar(casilleros[i])
  fin para
  devolver la concatenación de casilleros[1],..., casilleros[n]
```

Aquí elementos es la lista de datos a ordenar y n el número de casilleros que queremos usar. Para buscar el casillero adecuado para un elemento se puede

utilizar la técnica que más convenga, según cómo queramos ordenar los datos. La función ordenar puede ser cualquier función de ordenamiento, incluso la propia *bucket-sort.*

Después de todos estos tipos de algoritmos, el utilizado en esta aplicación es el ordenamiento burbuja ya que es el más sencillo de utilizar y siempre da buenos resultados.

5.3 Rutina de control de servomotores en C

```
#int_TIMER0

void TIMER0_isr(void){

  set_timer0(0x1B);                              //cargamos timer 0a 20mS
  set_timer1(59560);                              //timer1 500uS
  output_B(0xff);
  output_C(0xff);
  //output_low(PIN_C7);
  disable_interrupts(int_ext);

  for (i=0;i<numservos-1;i++){
    s[0][i]=s2[0][i];
    s[1][i]=s2[1][i];
  }
  t1=1;                                          //activación timer 1 necessario

      for (i = 0;i < numservos-2; i++) {                  //ordenacion

            for (j = i + 1; j < numservos-1 ; j++) {
                  if (s[0][i] > s[0][j]) {
                        temp = s[0][i];
                        s[0][i] = s[0][j];
                        s[0][j] = temp;
                        temp = s[1][i];
                        s[1][i]=s[1][j];
                        s[1][j] = temp;
                   }
            }
      }
```

```
  temps=500;                                  //variable de tiempo
   for (i=0;i<numservos;i++){                 // Servo correspondiente
     while (s[0][i]>temps){
      }
     switch (s[1][i]){
       case 0:output_low(PIN_B1);
            break;
       case 1:output_low(PIN_B2);
            break;
       case 2:output_low(PIN_B3);
            break;
       case 3:output_low(PIN_B4);
            break;
       case 4:output_low(PIN_B5);
            break;
       case 5:output_low(PIN_C0);
            break;
       case 6:output_low(PIN_C1);
            break;
       case 7:output_low(PIN_C2);
            break;
       case 8:output_low(PIN_C6);
            break;
       case 9:output_low(PIN_C7);
            break;
     }
  }
  t1=0;                                  //desconectar timer 1 no necessario
  enable_interrupts(int_ext);

}
```

```
#int_TIMER1 high

void TIMER1_isr(void){
  if (t1==1){
     set_timer1(65466);                          //10uS
     temps=temps+10;                              // variable temps + 10us

  }
}
```

5.4 configuraciones iniciales

```
set_tris_A(0b00000111);                    //configuración de puertos
set_tris_B(0b00000001);
set_tris_C(0b00000000);
port_b_pullups(TRUE);                      //R de pullup activada
setup_adc_ports(AN0_to_AN2);               //entradas ADC
setup_adc(ADC_CLOCK_INTERNAL);             //Reloj ADC
setup_adc(ADC_CLOCK_DIV_64);
setup_wdt(WDT_OFF);
setup_timer_0(RTCC_INTERNAL|RTCC_DIV_4);     //timer0
setup_timer_1(T1_INTERNAL|T1_DIV_BY_1);      //timer1
enable_interrupts(INT_TIMER1);               //interrupcion timer0
enable_interrupts(INT_TIMER0);               //interrupcion timer1
enable_interrupts(int_ext);                  //interrupcion externa
ext_int_edge(L_TO_H);
enable_interrupts(global);                   //interrupciones

set_timer0(0x1B);                            //timer0 para 20ms
```

CAPÍTULO 6:

SIMULACIONES

6.1 Convertidor A/D

A continuación podemos ver una simulación en el software Proteus del funcionamiento del convertidor A/D. Como podemos observar, el potenciómetro se encuentra al 50% de su posición y la respuesta del convertidor A/D es de 511 cuentas, que corresponde a 2.49 V. teniendo en cuenta que el sistema se encuentra alimentado a 5V podemos afirmar que el convertidor funciona correctamente.

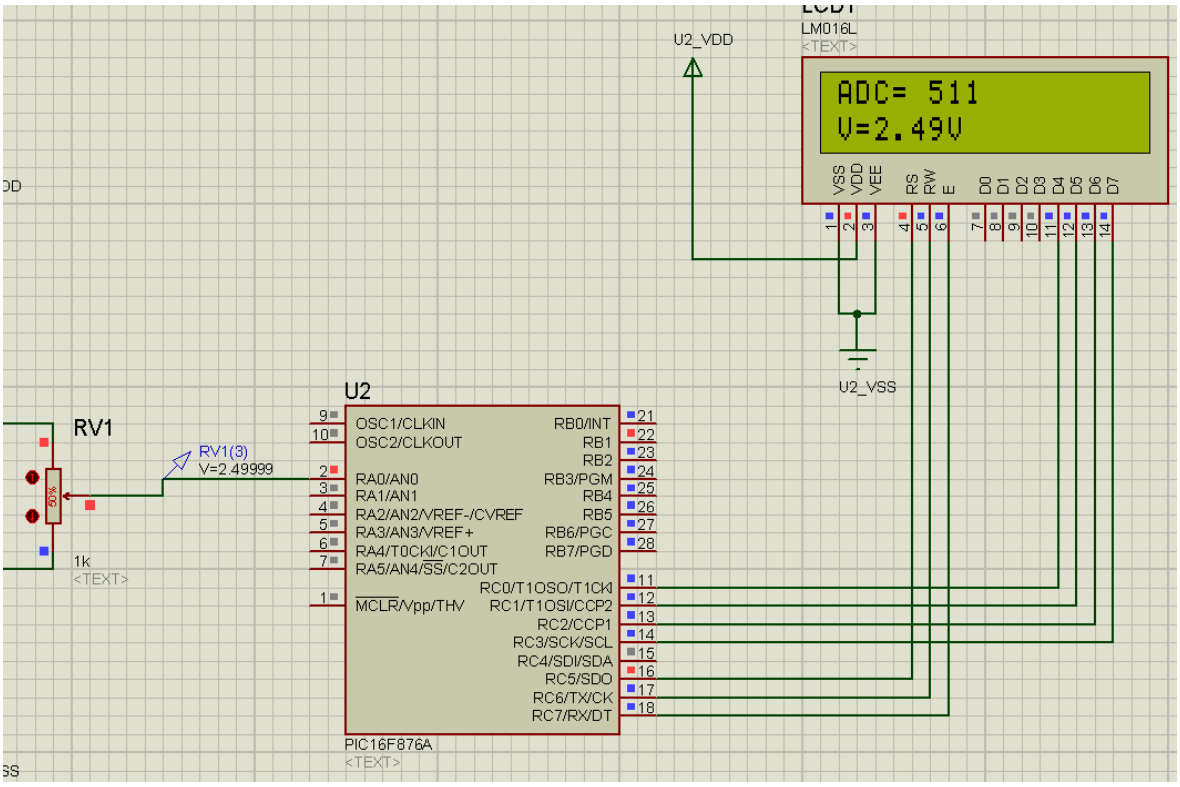


Fig. 73. Esquema de prueba del convertidor A/D.

A continuación vemos el resultado de tener el potenciómetro al 80%:

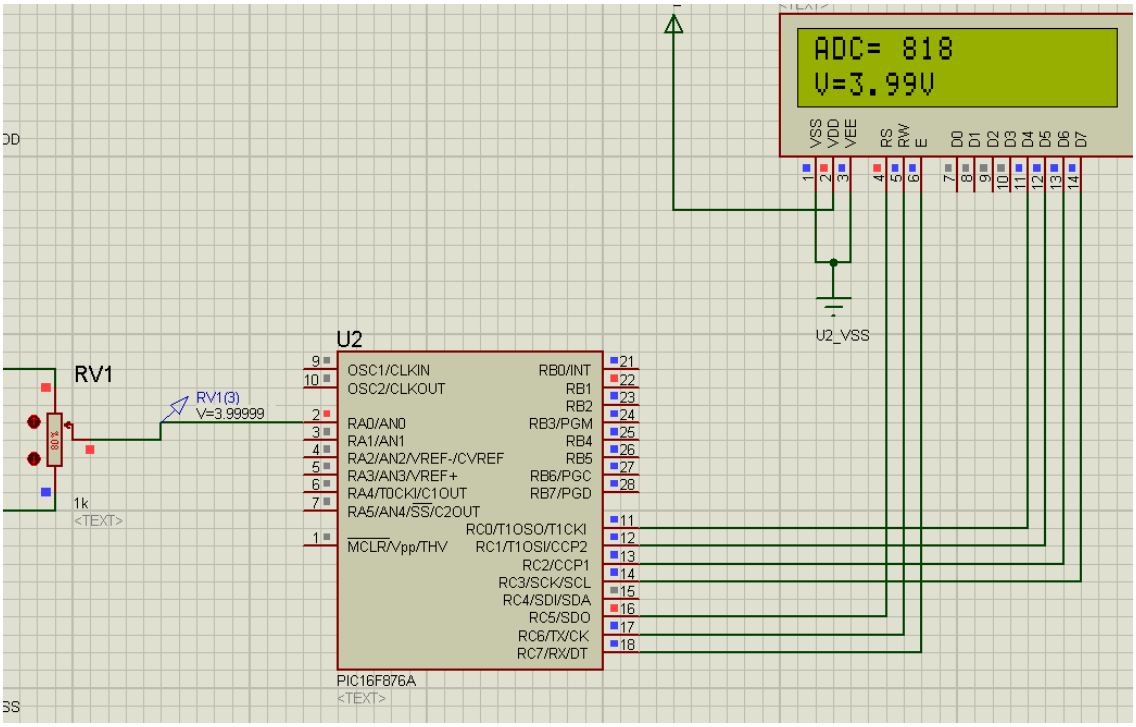


Fig. 74. Esquema de prueba del convertidor A/D.

A continuación el resultado de tener el potenciómetro a 100%:

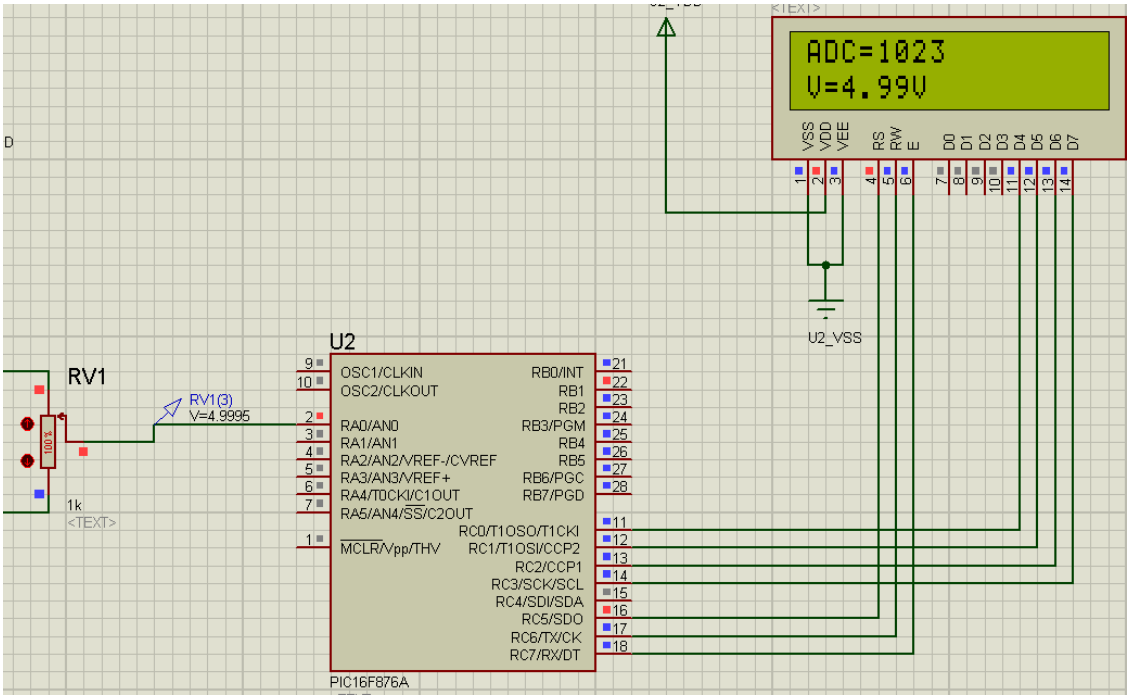


Fig. 75. Esquema de prueba del convertidor A/D.

6.2 Funcionamiento PWM

A continuación podemos ver la simulación en Proteus.

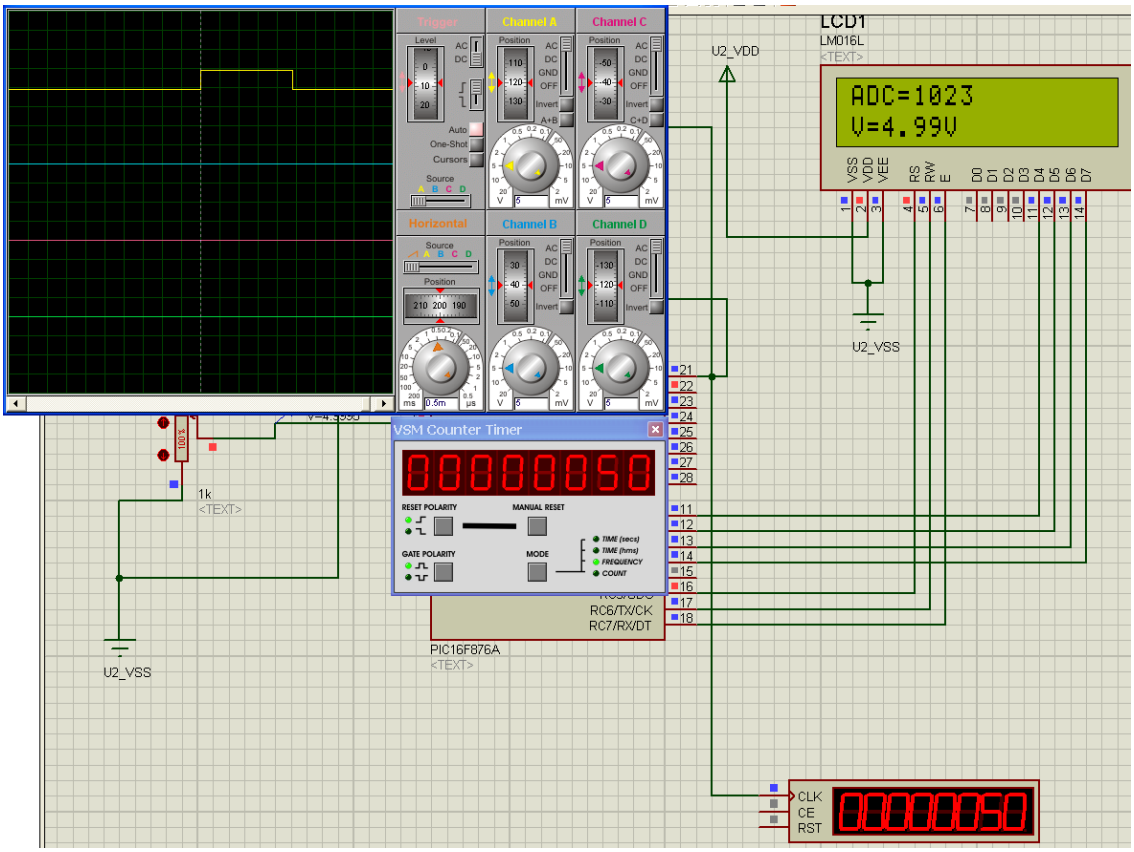


Fig. 76. Simulación del PWM.

La salida en PWM creada por el timer0 es el PIN RB0. Como se puede observar en la simulación, la frecuencia de oscilación de TIMER0 es de 50Hz. El programa está pensado para que, según el valor de tensión recibido en la entrada analógica, se varíe el ancho de pulso del PWM. En este caso el potenciómetro se encuentra al 100% como podemos observar en el display. Mediante el osciloscopio podemos observar el tiempo que la señal se encuentra en estado alto.

En la siguiente figura podemos ver el resultado de variar este potenciómetro al 50%:

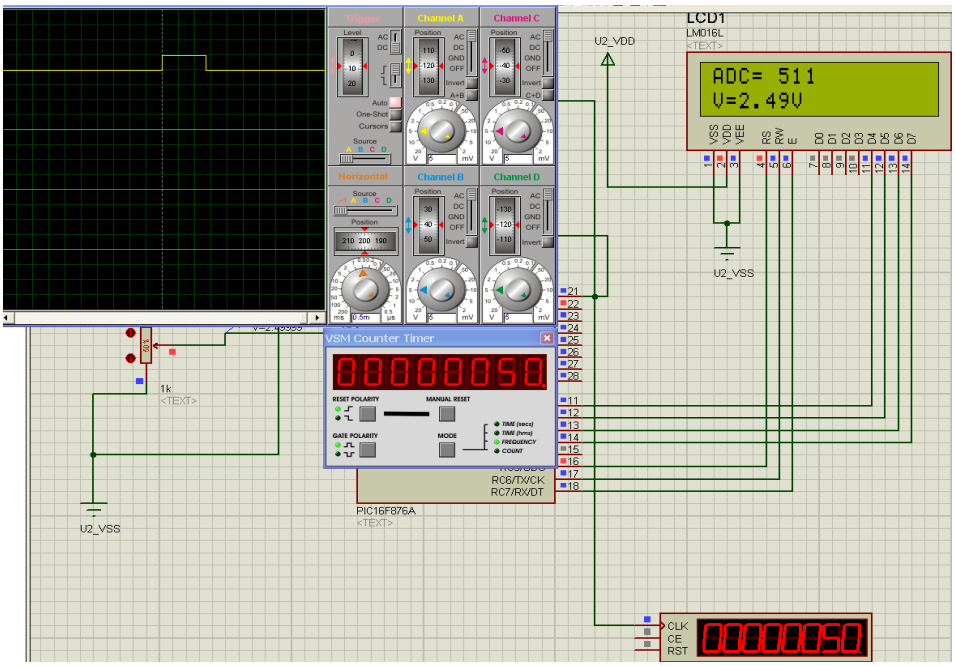


Fig. 77. Simulación del PWM

Como se puede observar, en este caso el potenciómetro se encuentra al 50%, verificable mediante la pantalla LCD. También podemos observar como el valor de frecuencia del PINB0 sigue siendo de 50Hz mientras que el ancho de pulso visto en el osciloscopio a variado a casi la mitad de la figura anterior.

A continuación vemos la simulación para el caso del potenciómetro al 0%:

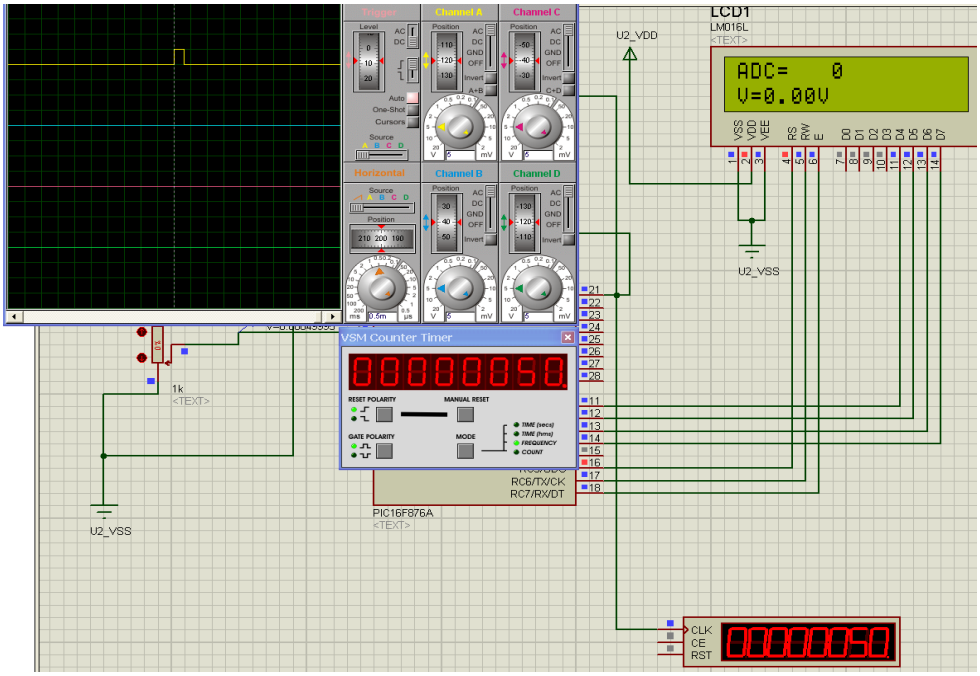


Fig. 78. Simulación PWM.

Se observa claramente como en este caso el ancho de pulso del PWM es más pequeño que el caso del potenciómetro al 50%. El duty cicle no acaba de llegar a 0 ya que no nos interesa. Esta aplicación es para usarla en servomotores que tienen un tiempo máximo y mínimo de estado alto de PWM, por tanto , esta simulación está pensada para el servo futaba S3003 que tiene un rango que va de 0,3 ms a 2,3 ms a la frecuencia de 50Hz.

Podemos observar a la figura siguiente que estando el potenciómetro al 100%, por tanto, al máximo valor admisible de PWM, la posición del servomotor se encuentra a +90º:

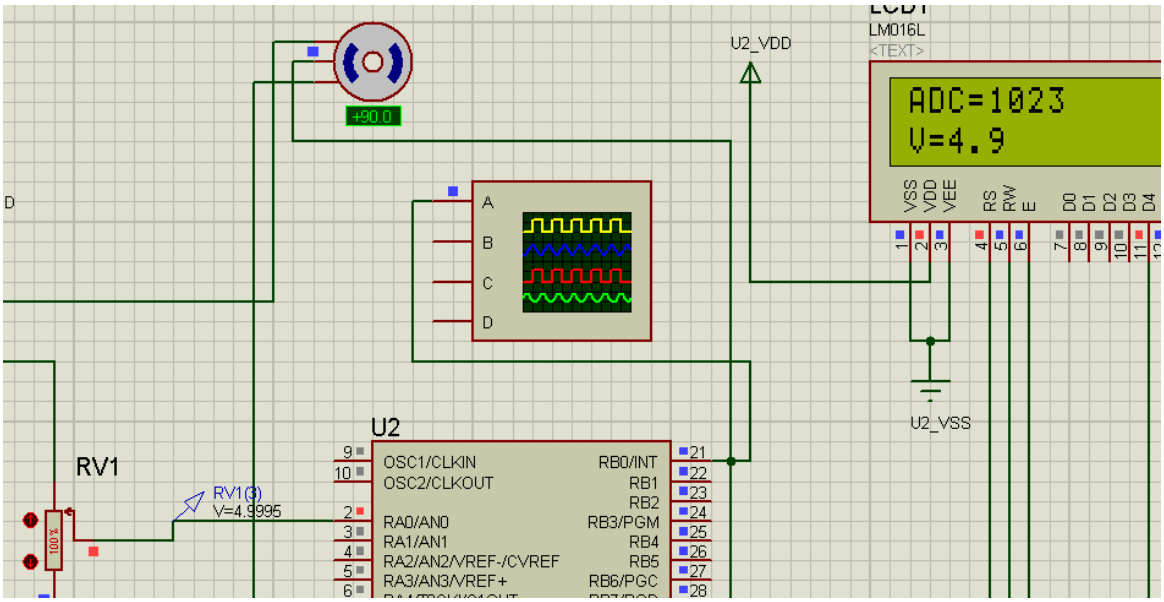


Fig. 79. Simulación servomotor.

Por el contrario, si llevamos el potenciómetro al otro extremo(al 0%) vemos como el servomotor se desplaza hacia el -90º como vemos a la figura siguiente:

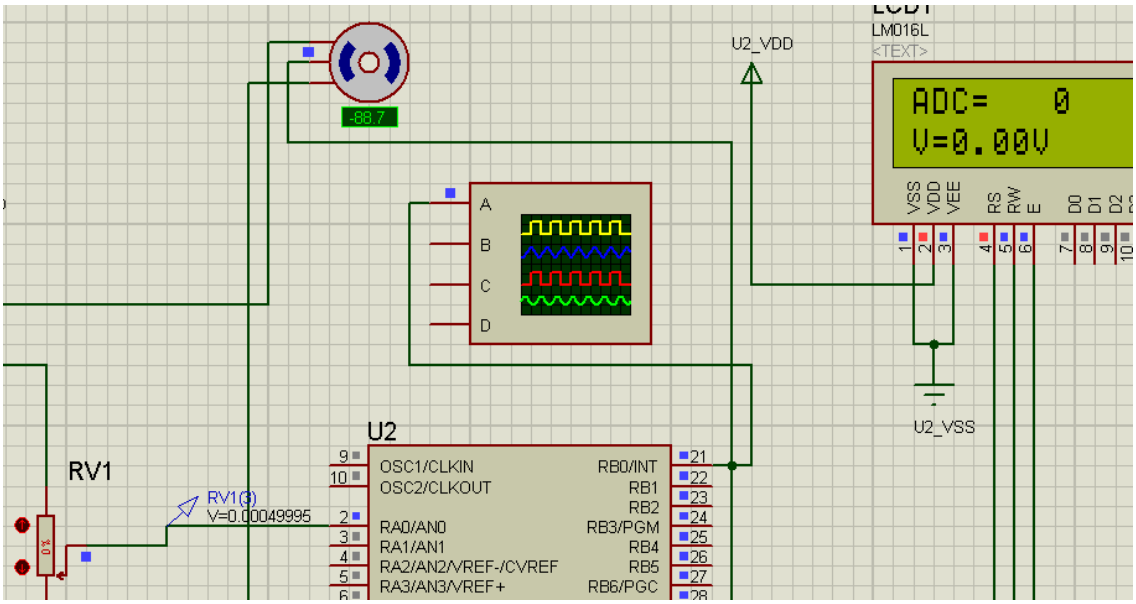


Fig. 80. Simulación servomotor.

Si dejamos el potenciómetro más o menos a la mitad de su recorrido observamos que el servomotor se coloca en su posición central, aproximadamente a 0º. Lo podemos ver a la figura siguiente:

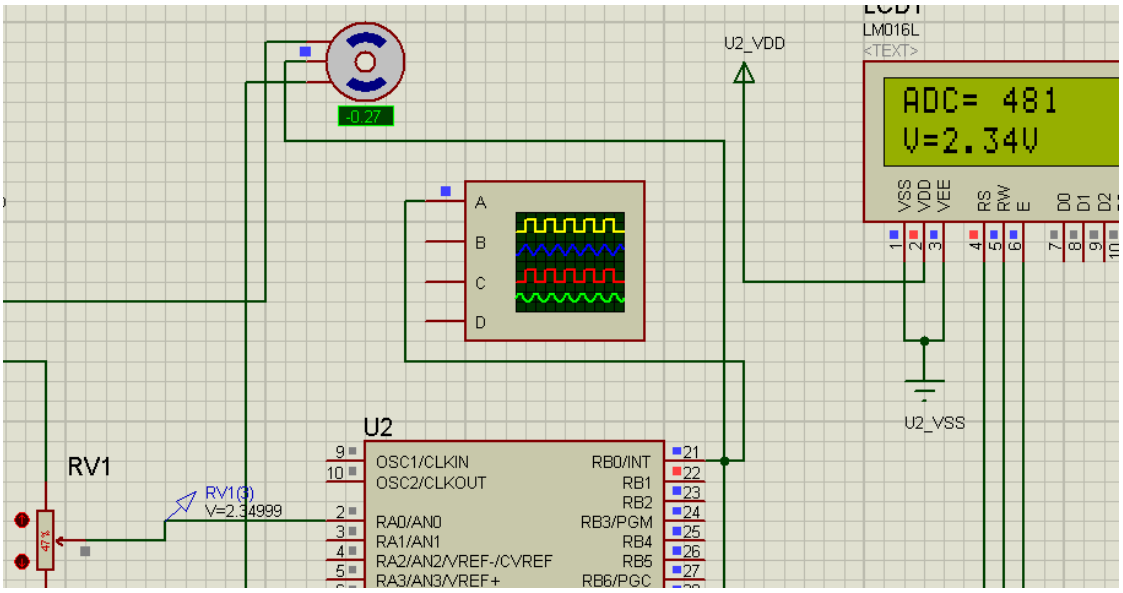


Fig. 81. Simulación servomotor.

6.3 Funcionamiento de la rutina de control de servomotores.

Para poder comprobar que la rutina de ordenamiento tiene una ejecución suficientemente veloz se ha modificado dicha rutina para que nos dé información a través de dos pines. Para poder conocer si realmente está funcionando y los tiempos necesarios son los correctos lo que se hace es que, en un primer momento se le dice al principio de la rutina que ponga a nivel alto uno de los pines (Canal A figura 80). A continuación empieza el ordenamiento de los servos y solo cuando ha finalizado todo el proceso de ordenación se vuelve a bajar ese pin a nivel bajo. Inmediatamente después se vuelve a subir. Mientras tanto, el *timer1,* se ha cargado desde el principio con el valor correspondiente para que actúe al cabo de 500us, tiempo en el cual ya debería haber acabado la conversión. De no ser así la rutina no funcionaría a causa de ser demasiado lenta. Una vez se llega a sobrepasar esos 500us, el *timer1* queda configurado para que dé una interrupción cada 10us. Esta interrupción servirá para contar el tiempo y así saber en qué momento se debe bajar el estado de cada servomotor. En la simulación se ve claramente que cuando acaba la ordenación de los tiempos de los servos, aun no han pasado esos 500us. En el momento que se llega a ellos, se puede observar como la interrupción por *timer1* empieza a contar.

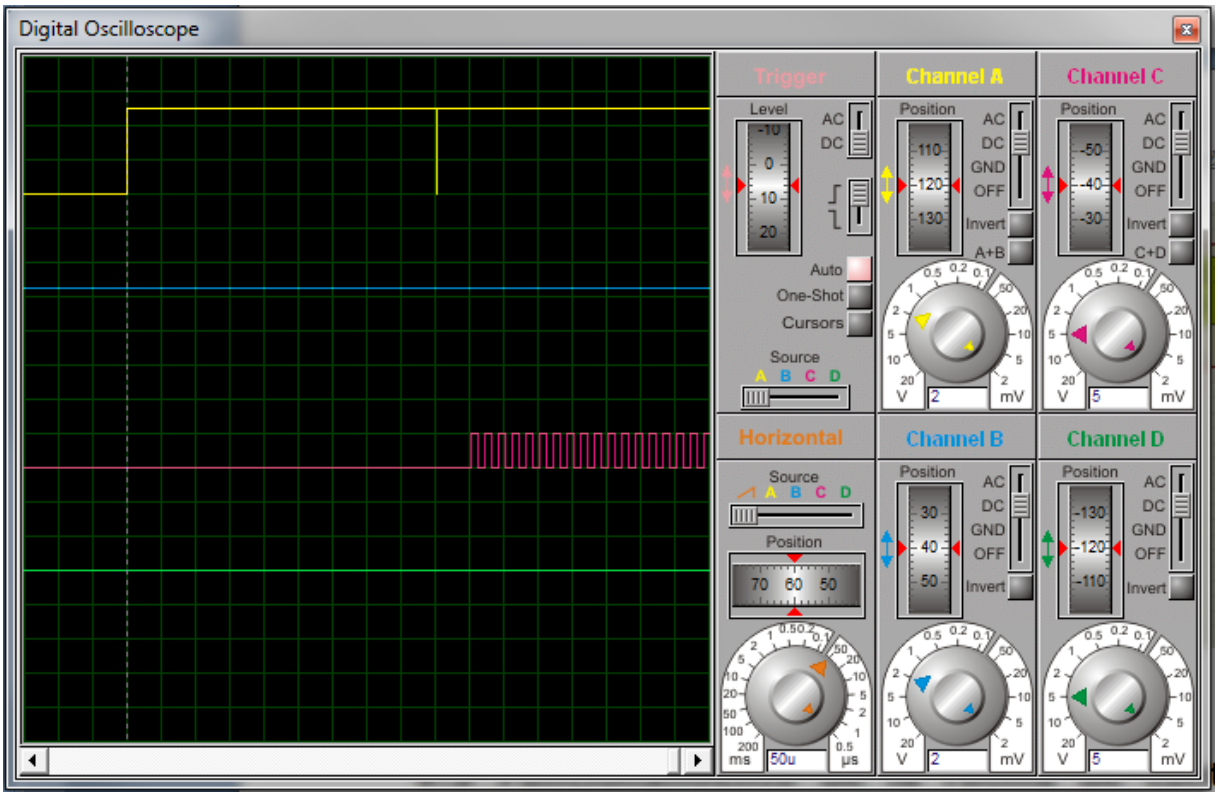


Fig. 81. Simulación servomotor. Canal A seguimiento de rutina. Canal C seguimiento de contador *timer1*.

El tiempo máximo admisible para realizar todo el algoritmo de ordenación queda establecido por el tiempo mínimo al que podría estar configurado alguno de los servomotores. En el caso de esta aplicación y con los servos utilizados el tiempo mínimo se establece en 600us.

Se puede afirmar por tanto, que la rutina de ordenación funciona correctamente y su tiempo es menor que el mínimo necesario.

Para poder ver que realmente el sistema funciona la siguiente prueba es configurar todos los servomotores al mínimo tiempo posible. Esta prueba servirá para poder comprobar que el máximo tiempo en que el microcontrolador se mantiene en la rutina de control de servos es el máximo tiempo que tiene asignado alguno de ellos. En este caso, configurando todos los servomotores a 600us el tiempo máximo que debería permanecer en la rutina es precisamente ese tiempo. Para poder visualizarlo lo que haremos es mediante el pin conectado al canal A del osciloscopio, que había quedado a nivel alto, se bajará a nivel bajo. El resultado de la simulación de la prueba se puede visualizar en la figura siguiente.

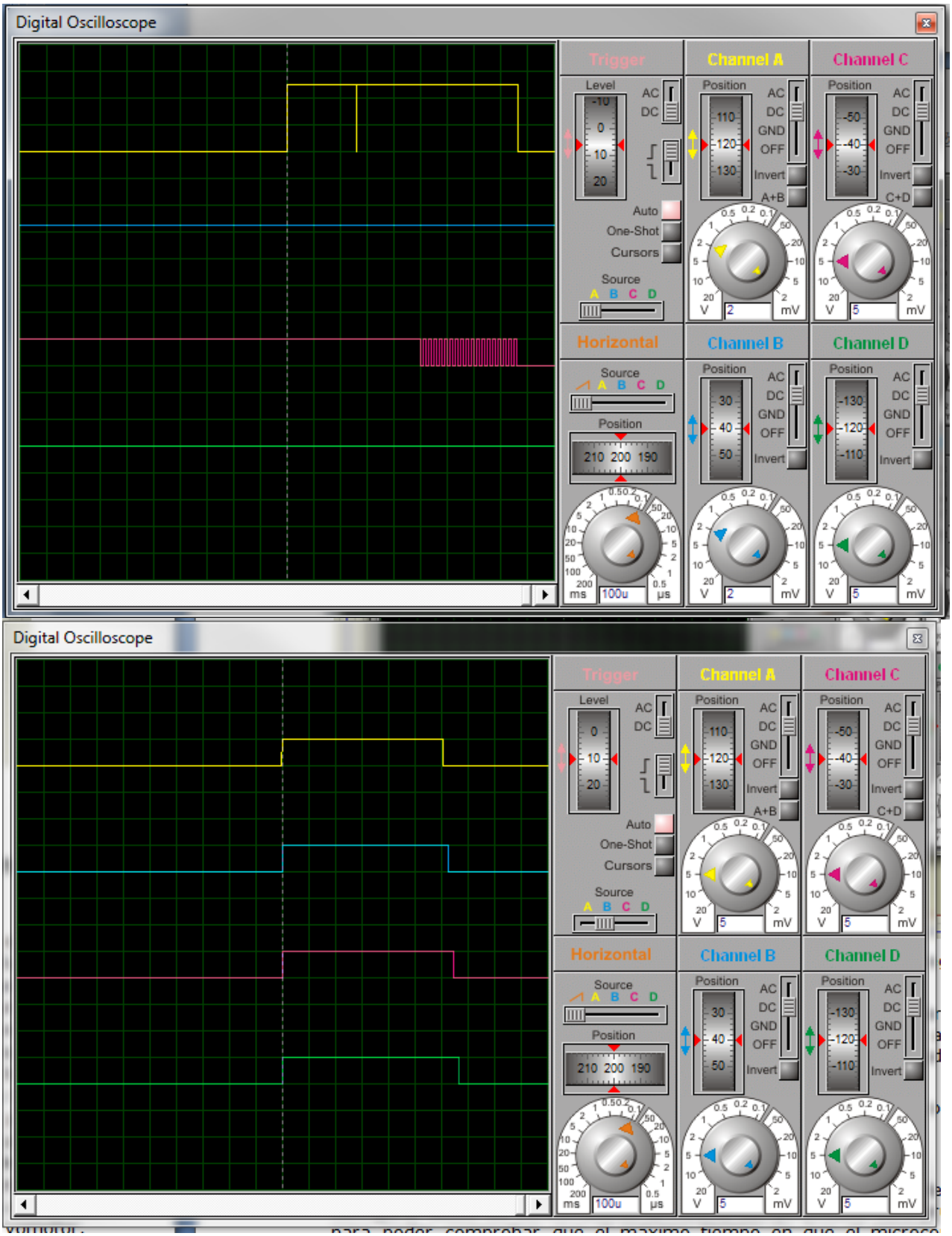


Fig. 83. Simulación servomotor. Osciloscopio superior: Canal A seguimiento de rutina. Canal C seguimiento de contador *timer1.* Osciloscopio inferior: pulsos de salida de la rutina.

Podemos comprobar mediante la figura anterior que la rutina de control funciona correctamente. Por un lado, en el osciloscopio superior vemos en los momento

en que empieza a ordenar (Canal A asciende) y cuando ya ha ordenado todo (canal A con un pico) a continuación se empieza a contar y hasta que no se llegan a los 600us no se sale de la rutina. En el osciloscopio inferior se puede observar 4 señales PWM de distintos pines.

A continuación se verá que pasa en caso de que alguno de los servos tenga un pulso de mayor duración.

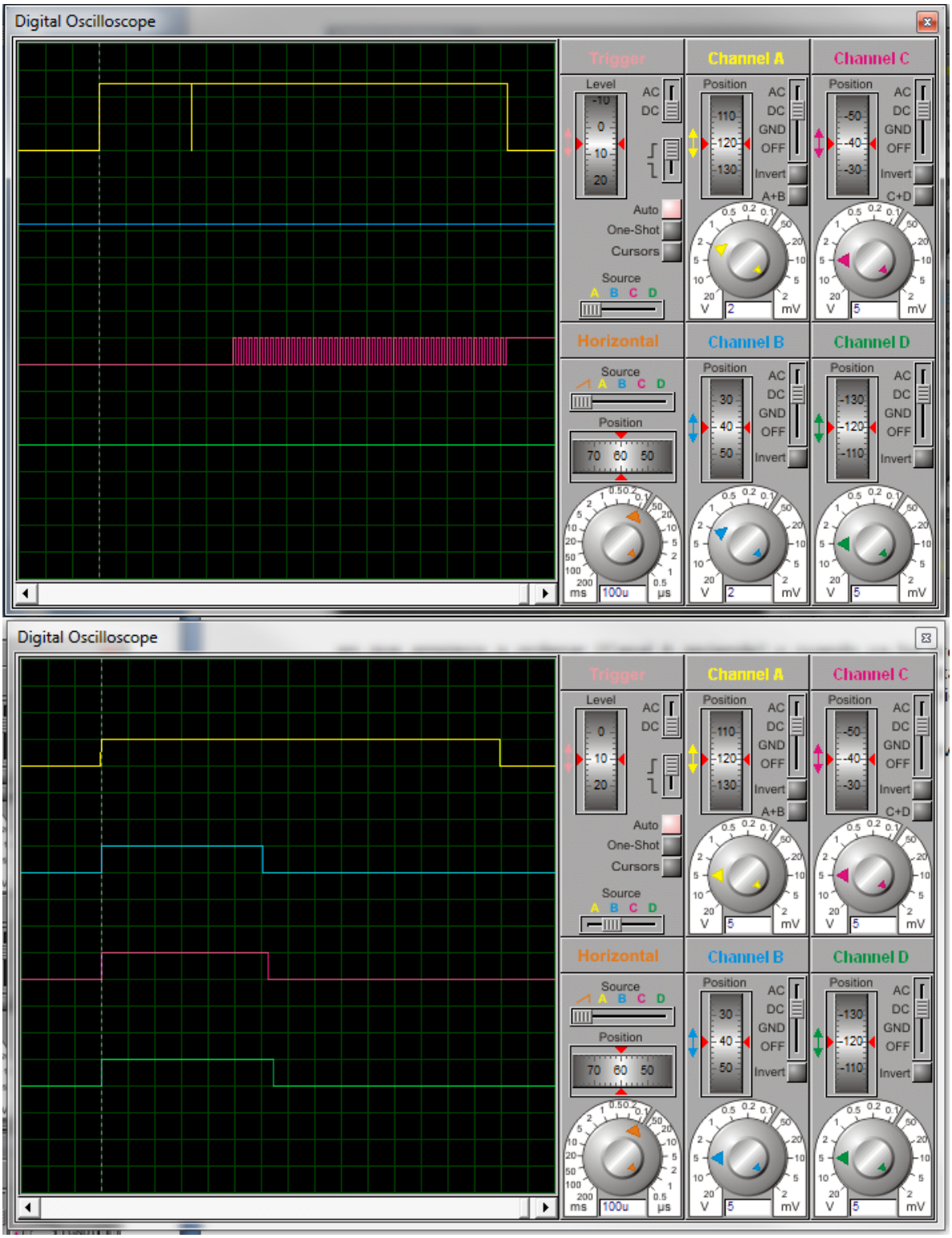


Fig. 84. Osciloscopio superior: Canal A seguimiento de rutina. Canal C seguimiento de contador *timer1.* Osciloscopio inferior: pulsos de salida de la rutina.

En este caso se ha aplicado un pulso de 1500us al primer servo. Se puede comprobar en el osciloscopio inferior. Se puede observar como ahora la rutina de control de servomotores se alarga hasta exactamente lo que vale el pulso del servomotor más duradero (osciloscopio superior, canal A), 1500us.

Este sistema está pensado para microcontroladores de gama media que no necesitan tener las velocidades que adoptan los dispositivos DSP y otros del mismo nivel pero aun así se necesita un mínimo de velocidad para poder realizar los algoritmos de ordenación con un tiempo aceptable dentro de los límites establecidos por los servomotores. De lo contrario, si el microcontrolador no fuera capaz de de realizar el algoritmo en el tiempo máximo se empezaría a "comer" el tiempo mínimo de pulso del servomotor y eso causaría que no se dispondría de todas las posiciones posibles de este.

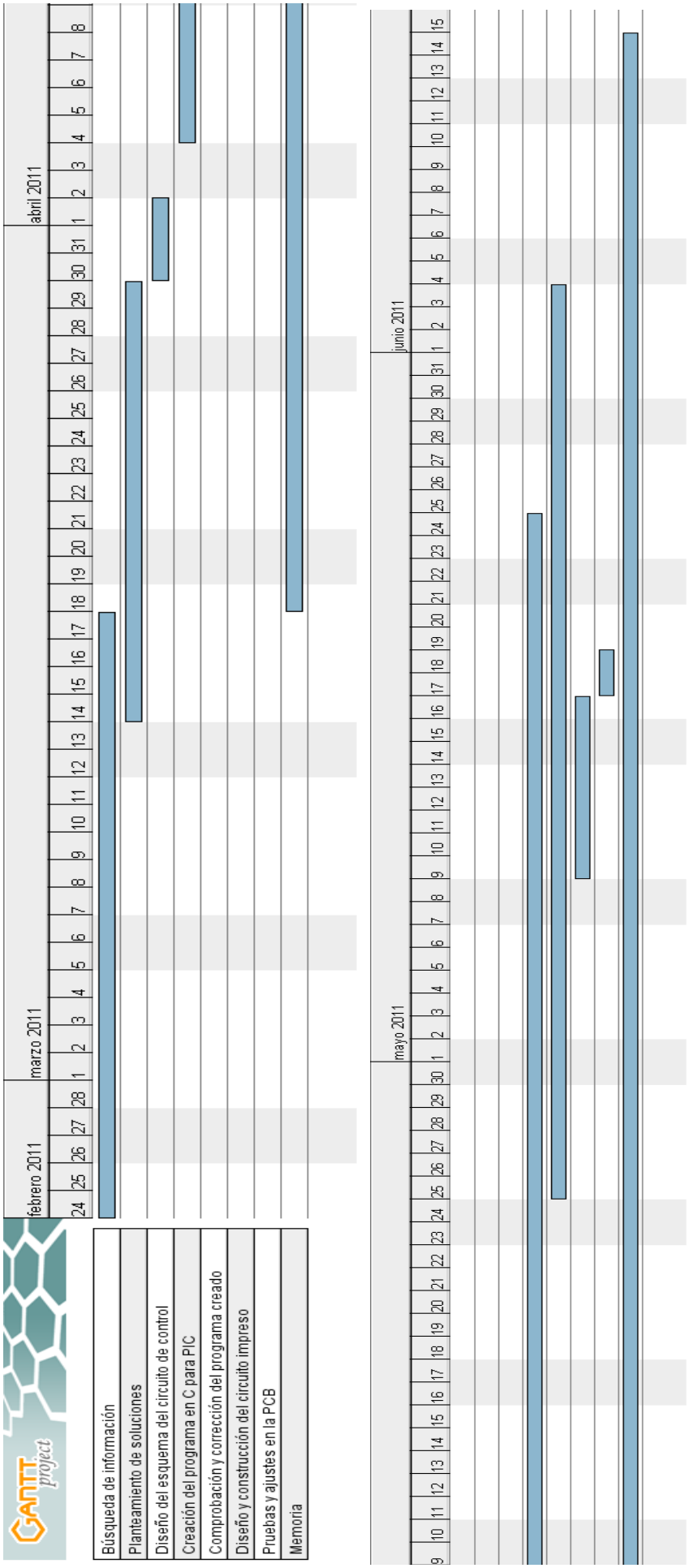
En la práctica se ha podido comprobar que con este tipo de algoritmo este microcontrolador es capaz de gobernar hasta 14 servomotores sin problemas en cuanto a velocidad.

CAPÍTULO 7: PLANIFICACIÓN

A continuación se muestra como se ha estructurado el trabajo en el presente proyecto a lo largo del actual cuatrimestre.

Las categorías mostradas son:

- Búsqueda de información.
- Planteamiento de soluciones.
- Diseño del esquema del circuito de control.
- Creación del programa en C para PIC.
- Comprobación del funcionamiento del programa.
- Diseño y construcción del circuito impreso.
- Pruebas y ajustes en la placa
- Memoria Técnica


GANTT project
febrero 2011
marzo 2011
abril 2011
mayo 2011
junio 2011
Búsqueda de información
Planteamiento de soluciones
Diseño del esquema del circuito de control
Creación del programa en C para PIC
Comprobación y corrección del programa creado
Diseño y construcción del circuito impreso
Pruebas y ajustes en la PCB
Memoria

CAPÍTULO 8: CONCLUSIONES Y POSIBLES MEJORAS

El presente proyecto ha llevado a la puesta en práctica de muchos conocimientos adquiridos a lo largo de los estudios cursados, no obstante, se ha tenido que profundizar en distintos campos tanto en electrónica como en programación, y eso ha servido para aumentar el conocimiento de dichos estudios.

Ha sido necesario el estudio y la reflexión en profundidad de todos los elementos que componen un robot móvil a fin de comprender las necesidades que estos pueden entrañar, en especial el funcionamiento de los servomotores y del acelerómetro.

Antes de comenzar a gestar las librerías, fue necesario documentarse sobre el funcionamiento y lenguaje utilizado por las herramientas de desarrollo de programación de microcontroladores (CCS, PICKIT2).

Por otra parte, se podrían realizar mejoras. Se podría dotar a la placa de un cargador específico de baterías NI-MH, proporcionando una integración absoluta, dejando como elemento exterior un mero transformador para conectar a la red eléctrica, no sería una tarea muy complicada porque existen integrados que realizan completamente esta tarea, pero teniendo en cuenta que es un prototipo, a fin de no suponga un coste añadido. Queda reservada esta modificación en el caso de tratarse de una producción serie. Por tanto se debería añadir en el estudio económico de una producción seriada.

Otra mejora interesante sería la de realizar el circuito de control en SMD, de esta forma se conseguiría poder integrar mejor todos los componentes permitiendo incluso variar el microcontrolador por una que

Un elemento más a añadir y que aumentaría las posibilidades de estabilidad y movilidad del proyecto es la adición de dos brazos robóticos que en harmonía con las piernas dotarían al robot de movimientos casi ilimitados.

También sería interesante la posibilidad de añadir una cámara para poder dotar al robot y, mediante el procesado de las imágenes recibidas, poder reconocer según qué objetos que serian importantes que conociera.

Otras mejoras podrían ser la realización de un entorno gráfico para generar el código para el robot. A este se podría añadir un programa *bootloader* para poder enviar ficheros directamente a la controladora, añadiendo a este entorno gráfico la posibilidad de programar desde la misma aplicación el robot, configurando una aplicación completa y sin que tener que programar el microcontrolador en sí cada vez que se realiza alguna modificación de programa.

Otra mejoría a añadir dentro del tema de las comunicaciones las comunicaciones, añadir un módulo externo de ZegBee y así crear una posible comunicación con el exterior, bien con una estación central (PC) o con otros robots. Ello daría pie al estudio de algoritmos de cooperación entre individuos. Si se quiere realizar una comunidad de trabajo de estos dispositivos será necesario que identifiquen su posición y la de los objetos que los rodean.

En resumen el campo de desarrollo e investigación puede ser ilimitado.

CAPÍTULO 9: BIBLIOGRAFÍA Y WEBGRAFÍA

Eduardo García Breijo. 'Compilador C CCS y simulador Proteus para Microcontroladores PIC' Ed.

TodoBasicstamp.com, "Robótica Móvil y Autónoma," 2009,
http://www.todomicrostamp.com/robotica.php?grupo=movil

HONDA, "ASIMO, History Robot development Process," 2009, http://world.honda.com/ASIMO/history/

X-robotics, "Robótica y µControladores PIC, mecánica," 2009,
http://www.xrobotics.com/

Parallax, "PING))) Ultrasonic Distance Sensor Data Sheet," febrero 2008,
http://www.parallax.com/Portals/0/Downloads/docs/prod/acc/28015-PINGv1.5.pdf

http://www.iearobotics.com/personal/juan/publicaciones/art9/html/node2.html

http://www.iearobotics.com/proyectos/cuadernos/ct3/ct3.html

http://www.heli-system.com/hitec/252-servo-hitec-hs-645mg-ultra-torque.html

http://www.microchip.com

http://www.ccsinfo.com/

http://es.rs-online.com

http://www.lynxmotion.com/

http://linuxdroids.wordpress.com/2010/05/07/control-de-servos-en-paralelo-con-pic/