UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLÁN

CONTROL DE ROBOT TIPO SCARA CON MICROCONTROLADORES PIC

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO MECÁNICO ELECTRICISTA

P R E S E N T A :

JULIO CÉSAR FRANCO MEJÍA

ASESOR: ING. CARLOS ERNESTO PINEDA GARCÍA

CUAUTITLÁN IZCALLI, EDO. DE MÉXICO. 2005

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLAN
UNIDAD DE LA ADMINISTRACION ESCOLAR
DEPARTAMENTO DE EXAMENES PROFESIONALES

ASUNTO: VOTOS APROBATORIOS

U. N. A. M.
FACULTAD DE ESTUDIOS
SUPERIORES-CUAUTITLAN

DEPARTAMENTO DE
EXAMENES PROFESIONALES

DR. JUAN ANTONIO MONTARAZ CRESPO
DIRECTOR DE LA FES CUAUTITLAN
P R E S E N T E

ATN: Q. Ma. del Carmen García Mijares
Jefe del Departamento de Exámenes
Profesionales de la FES Cuautitlán

Con base en el art. 28 del Reglamento General de Exámenes, nos permitimos comunicar a usted que revisamos la TESIS:

Control de Robot tipo SCARA con Microcontroladores PIC

que presenta el pasante: Julio Cesar Franco Mejia
con número de cuenta: 09851541-9 para obtener el título de
Ingeniero Mecanico Electricista

Considerando que dicho trabajo reúne los requisitos necesarios para ser discutido en el EXAMEN PROFESIONAL correspondiente, otorgamos nuestro VOTO APROBATORIO.

A T E N T A M E N T E
"POR MI RAZA HABLARA EL ESPIRITU"
Cuautitlán Izcalli, Méx. a 28 de mayo de 2004

PRESIDENTE	Ing. Filiberto Leyva Piña	
VOCAL	Ing. Alejandro Martinez Moncada	
SECRETARIO	Ing. Carlos Ernesto Pineda Garcia	
PRIMER SUPLENTE	Ing. Julio Cesar Vazquez Fuentes	
SEGUNDO SUPLENTE	Ing. Leonardo Sergio Lara Flores	

Agradecimientos

A mis padres:

Por su apoyo y orientación en la vida para ser una persona integra, y por darme la oportunidad de seguir mi camino cualquiera que este sea.

A mi pequeña Adrianita:

Por darme ese pequeño impulso e inspiración que a veces necesitamos para sembrar éxitos futuros

"CONTROL DE ROBOT TIPO SCARA CON MICROCONTROLADORES PIC"

OBJETIVOS:

1. Comprobar experimentalmente la programación y el control de un robot manipulador tipo SCARA[1] de configuración PRRR basándose en microcontroladores tipo PIC[2] para poder ejecutar las instrucciones fuera de línea una vez programado.

2. Construir el modelo cinemático de posición, velocidad y aceleración de un robot SCARA, a través de la teoría de mecanismos y el álgebra compleja, para obtener los parámetros de desplazamiento angular de los eslabones y así determinar una secuencia de movimientos asociada con el robot.

3. Elaborar un software de control y simulación que permita crear el programa en lenguaje de PIC (ensamblador) para que, mediante la utilización de uno de los puertos de la PC[3], se puedan programar dentro del mismo circuito (ICSP)[4], los microcontroladores con la secuencia de comandos que den el movimiento y posición de los eslabones del Robot.

4. Diseñar y construir un robot manipulador tipo SCARA, así como los elementos que se utilizaran para su control.

---

[1] SCARA: Selective Compilance Assemble Robot Arm
[2] Pic: Microcontraldor Programable
[3] PC: Computadora Personal IBM compatible.
[4] In Circuit Serial Programing (Programación serial en circuito)

ÍNDICE:

INTRODUCCIÓN

CAPÍTULO I

Marco teórico.

1.1. Fundamentos de Robótica y Álgebra compleja

1.2. Teoría de Mecanismos

1.3. Motores de Pasos

1.4. El PIC 16F84A

1.5. Etapas de Potencia

CAPÍTULO II

Construcción del perfil de velocidad, posición y aceleración de un robot de cuatro grados de libertad.

2.1. Perfil de posición.

2.2. Perfil de velocidad.

2.3. Perfil de aceleración.

CAPÍTULO III

Software

3.1. Programación Básica en Visual Basic®

3.2. MPLAB®

3.3. PIC PROG

3.4. Software para simulación y programación escrito en Visual Basic®

CAPÍTULO IV

Diseño del la estructura del brazo manipulador.

4.1. Materiales a utilizar

4.2. Diseño mecánico

CAPÍTULO V

Análisis de Trayectorias

5.1 Simulación de trayectorias con el software desarrollado.

5.2 Análisis de velocidad y aceleración para trayectorias punto a punto con un perfil estacionario.

CONCLUSIONES.

APÉNDICE A "Comandos del Lenguaje Ensamblador"

BIBLIOGRAFÍA.

INTRODUCCIÓN:

La robótica es en la actualidad una ciencia multidisciplinaria, en la cual intervienen la mecánica, la electrónica, la programación, entre otras. Pero los antecedentes de lo que son los robots e incluso el vocablo mismo ha tenido un origen que se remonta siglos atrás. Algunas fuentes sostienen que la palabra fue utilizada por primera vez por Karel Capek hacia el año 1917, en su novela 'Opilec' mientras otros sostiene que fue Isaac Asimov hacia el año de 1940, quien le dio un significado más cercano a la palabra de lo que hoy en día, la mayoría de nosotros consideramos un robot. En sus obras, Asimov hacia referencia a mecanismos humanoides que convivían con los humanos. A últimas fechas se sigue hablando de este tipo de máquinas, como por ejemplo en el filme 'Inteligencia Artificial' donde se crea un robot capaz no solo de convivir y actuar como un humano, sino incluso de sentir.

En realidad un robot industrial dista mucho de estas definiciones. Un robot o brazo mecánico industrial es un mecanismo que puede realizar tareas usualmente asignadas a los humanos, con la diferencia de que éste no se cansa, realiza la operación con gran precisión y repetibilidad, pero requiere de una inversión inicial fuerte en comparación con contratar un obrero.

La industria en la actualidad requiere de un sistema de producción capaz de reducir los costos mejorando la calidad de los productos.

La idea de construir una máquina con tales capacidades, que librará al hombre de tareas repetitivas y tediosas, o de trabajos peligrosos para su integridad, surgió hace bastante tiempo, pero técnicamente no existían los medios para llevar a cabo los proyectos de este tipo. Ejemplos de maquinas que libraran al hombre de algunas tareas como las mencionadas, son por ejemplo, las primeras calculadoras que se basaban en sistemas mecánicos o el telar de Jacard desarrollado en 1881, el cual utilizaba cintas perforadas para los patrones a tejer. Esto fue sin duda la base del control numérico actual.

Con el desarrollo de la electrónica y la computación en los últimos 50 años, se han tenido grandes avances en las áreas de control y automatización industrial,

pero los primeros sistemas de automatización resultaban ser muy inflexibles. Estos sistemas capaces de automatizar un proceso, perdían rentabilidad cuando se trataba de producir algo diferente de lo que originalmente había sido proyectado, ya que el grado de modificaciones que requería era a veces tan grande que era más fácil desecharlo y construir una línea de producción nueva. Debido a esto, la industria volteo su mirada a un sistema capaz de manufacturar una gran cantidad de productos de diversas características, fue entonces que surgieron los Sistemas de Manufactura Flexible, de los cuales, el robot industrial es pieza clave.

El robot industrial puede ser considerado como una máquina de uso general. Un mismo robot que puede participar en el proceso de soldadura, puede ser reprogramado para tareas de pintura o ensamble.

Basados en el álgebra compleja y la teoría de mecanismos, se construirá la modelación matemática de un robot manipulador de cuatro grados de libertad tipo SCARA. Mediante la ecuación de la cinemática inversa, se lograra que el robot manipulador pueda desplazarse a través de una trayectoria definida mediante una ecuación, sin el uso de elementos externos (joystick), y utilizando solo como dispositivo de direccionamiento de ordenes los microcontroladores, para el envío de información a los motores que imprimirán movimiento a los eslabones.

Para lograrlo se efectuara primero una simulación computacional apoyándonos en el desarrollado para este robot en lenguaje Visual Basic®, en donde se utilizara de forma practica la ecuación de la cinemática inversa y se obtendrán los ángulos y desplazamientos que deberán tener cada uno de los eslabones de la cadena para alcanzar la posición deseada, con estos datos se elaborara el programa de control que posteriormente será cargado en los microcontroladores los cuales se encargarán de dar salida a las señales hacia los diferentes dispositivos electromecánicos que conforman nuestro robot manipulador.

Este hardware estará conformado por una tarjeta interfase que será la encargada de recibir las señales de la PC para programar sus microcontroladores quienes tienen como función controlar cada movimiento deseado de los eslabones así como su correcta aceleración de cada uno, señales que finalmente serán recibidas por los motores PAP y rectificados por codificadores ópticos para hacer el lazo de control cerrado.

Capitulo 1

Marco Teórico

1.1 Fundamentos de robótica

Definición de Robot Industrial

Existen dificultades en el momento de establecer formalmente qué es un robot industrial, una de ellas es el concepto que se maneja en oriente y occidente sobre lo que es un robot. En Japón por ejemplo, un robot es un dispositivo mecánico cualquiera dotado de articulaciones móviles y destinado a la manipulación, mientras que en Europa y América, se es más exigente, dando la definición de robot a un mecanismo más complejo sobre todo en lo que a control se refiere. Es común por ejemplo que un mecanismo de manipulación controlado por un PLC[1], como los que ofrece FESTO® por ejemplo, Es considerado como un manipulador, mientras que en un robot se exige que posea un control más preciso, flexible y que interactúe más con el entorno.

También ha sido difícil establecer una definición porque con los avances tecnológicos se ha ido modernizando ésta también.

Una de las definiciones más aceptadas, es la que propone la RIA, por sus siglas en inglés, Asociación de Industrias Robóticas, según la cual:

- "Un robot industrial es un manipulador multifuncional reprogramable, capaz de mover materias, piezas, herramientas o dispositivos especiales, según trayectorias variables, programadas para realizar tareas diversas"[2].

Ésta misma definición, con algunas modificaciones ha sido adoptada por la Organización Internacional de Estándares, ISO, y define a un robot industrial como:

- "Un manipulador reprogramable con varios grados de libertad, capaz de manipular materias, piezas, herramientas o dispositivos especiales según trayectorias variables programadas para realizar tareas diversas."[3]

[1] Controlador Lógico Programable
[2] Fundamentos de Robótica. Antonio Barrientos. Mc Graw Hill. 1997. p 10.
[3] Idem.

Se incluye en esta definición lo referente a los grados de libertad para poder ser considerado un robot. Existe también la definición que nos da la Asociación Francesa de Normalización (AFNOR) que define primero el manipulador y posteriormente el robot.

- "Manipulador: mecanismo formado generalmente por elementos en serie, articulados entre sí, destinado al agarre y desplazamiento de objetos. Es multifuncional y puede ser gobernado directamente por un operador humano o mediante dispositivo lógico."[4]
- "Robot: Manipulador automático servocontrolado, reprogramable, polivalente, capaz de posicionar y orientar piezas, útiles o dispositivos especiales, siguiendo trayectorias variables reprogramables, para la ejecución de tareas variadas. Normalmente tiene la forma de uno o varios brazos terminados en muñeca. Su unidad de control incluye un dispositivo de memoria y ocasionalmente de percepción de su entorno. Normalmente su uso es el de realizar una tarea de manera cíclica, pudiéndose adaptar a otra sin cambios permanentes en su material."[5]

Por último, la definición más reciente que se tiene es la proporcionada por la Federación Internacional de Robótica en su informe técnico ISO/TR 83737 de septiembre de 1998, distingue el manipulador industrial de otros robots.

- Un robot industrial de manipulación se entiende a una máquina de manipulación automática, reprogramable y multifuncional con tres o más ejes que pueden posicionar y orientar materias, piezas, herramientas o dispositivos especiales para la ejecución de trabajos diversos en las diferentes etapas de la producción industrial, ya sea en una posición fija o móvil.

También vale la pena considerar la clasificación que nos proporciona la AFRI[6], la cual divide los manipuladores por tipos, que son:

[4] Idem
[5] Cómo y cuándo aplicar un robot industrial. Daniel Audí Piera. p 13 .
[6] Asociación Francesa de Robótica Industrial

Tipo A	Manipulador con control manual o telemando
Tipo B	Manipulador automático con ciclos preajustados, regulación mediante fines de carrera y topes, control por PLC y accionamiento neumático, hidráulico o eléctrico.
Tipo C	Robot reprogramable con trayectoria continua o punto a punto. Carece de conocimientos sobre su entorno.
Tipo D	Robot capaz de adquirir datos sobre su entorno, readaptando su tarea en función de éstos.

Por otra parte se establece la clasificación de los robots por medio de generaciones, las cuales son:

1ª Generación	Repite la tarea programada secuencialmente No toma en cuenta las posibles alteraciones de su entorno
2ª Generación	Adquiere información limitada de su entorno y actúa en consecuencia. Puede localizar, clasificar y detectar esfuerzos y adaptar sus movimientos en consecuencia.
3ª Generación	Su programación se realiza mediante el uso de lenguaje natural. Posee capacidad para la planificación automática de tareas.

Modelos matemáticos en Robótica

Dentro de la robótica clásica se encuentran varias herramientas matemáticas, de gran utilidad y muchas de ellas realmente populares en la mayoría de los trabajos y tratados sobre el tema.

La primera de ellas son los sistemas coordenados, entre los cuales están, las coordenadas polares, las cartesianas, cilíndricas y esféricas. De ellas se optará por utilizar las coordenadas cartesianas, por ser las más comunes y entendibles en la mayoría de los campos, además de que son las que interesan cuando uno desea hacer interpolaciones lineales. La resolución de la cinemática inversa en un robot tipo SCARA que tiene la misma dimensión en los eslabones de posicionamiento, como en el caso de este prototipo, es extremadamente sencilla en coordenadas polares, debido que para resolver **r** sólo necesitaríamos encontrar el ángulo correcto en la junta 3, y Θ estaría dado por la posición angular de la junta 2.

Otra de las herramientas matemáticas de gran utilidad son las representaciones de la orientación, de las cuales se citarán las más conocidas y se dará una breve descripción de las mismas.

Matrices de Rotación y Matrices de transformación homogéneas.

Las matrices de rotación, se dicen ser la herramienta más usada en la representación de orientaciones, gracias a la comodidad de cálculo que implica el álgebra matricial. Pero a diferencia de otros métodos, no pueden representar traslaciones del sistema coordenado de referencia. Para representar orientaciones y traslaciones se utilizan las matrices de transformación homogéneas, que son un arreglo matricial de 4x4 que adicionalmente representan la escala y perspectiva, aunque estas dos ultimas sin aplicación en la robótica.

Por otra parte, y de hecho la teoría de interés en esta ocasión, son los cuaterniones, de gran aplicación por su facilidad de implementación en los sistemas computacionales y los cuales son empleados incluso en robots comerciales como ABB.

Origen de los cuaterniones

Los cuaterniones fueron descubiertos por Sir William Rowan Hamilton en 1843. Hamilton buscaba formas de extender los números complejos (que pueden interpretarse como puntos en el plano complejo) a un número mayor de dimensiones. No pudo hacerlo para 3 dimensiones, pero para 4 dimensiones obtuvo los cuaterniones. Según una historia relatada por el propio Hamilton, la solución al problema que le ocupaba le sobrevino un día que estaba paseando con su esposa, bajo la forma de la ecuación: $i^2 = j^2 = k^2 = ijk = -1$. Inmediatamente, grabó esta expresión en el lateral del puente de Brougham, que estaba muy cerca del lugar.

Hamilton popularizó los cuaterniones con varios libros, el último de los cuales, *Elements of Quaternions*, tenía 800 páginas y fue publicado poco después de su muerte.

Definición

Un cuaternión Q esta constituido por cuatro componentes (Q_0, Q_1, Q_2, Q_3), que representan las coordenadas del cuaternión en cuestión en una base {e, i, j, k}, de las cuales suele llamarse parte escalar a *e*, y parte vectorial al resto de los componentes. Es decir

$$Q = [Q_0, Q_1, Q_2, Q_3] = [S, V] \qquad [1.1]$$

Donde ***S*** es la parte escalar y ***V*** la parte vectorial

Para definir la rotación de un ángulo – sobre el vector k, se expresa como

$$Q=Rot(K,\theta)=(\cos \theta/2, k \sin \theta/2) \qquad [1.2]$$

Esto nos da una importante herramienta para la composición de rotaciones

Por otra parte, se define una ley de composición interna ° (producto) según la tabla 1.1

Tabla 1.1. Ley de composición interna de los cuaterniones				
°	e	i	j	k
e	e	i	j	k
i	i	-e	k	-j
j	j	-k	-e	i
k	k	j	-i	-e

Cuaternión conjugado

A todo cuaternión q se le puede asociar su conjugado $Q*$, en el que se mantiene el signo de la parte escalar y se invierte el de la vectorial.

$$Q*=(S,-V) \qquad [1.3]$$

Operaciones algebraicas

Se definen tres operaciones en el álgebra de cuaterniones: producto, suma y producto con un escalar

El producto de dos cuaterniones Q1 y Q2, utilizado en la composición de transformaciones, esta dado por

$$Q_3=Q_1 \circ Q_2=(s_1,v_1)\circ(s_2,v_2)=(s_1s_2 - v_1v_2,\ v_1\times v_2+s_1v_2+s_2v_1) \qquad [1.4]$$

Como es común, cuando se componen rotaciones, no es un producto conmutativo. Se puede apreciar componente a componente:

$$Q_{30}=Q_{10}Q_{20}-(Q_{11}Q_{21}+Q_{12}Q_{22}+Q_{13}Q_{23}) \qquad [1.5]$$
$$Q_{31}=Q_{10}Q_{21}+Q_{11}Q_{20}+Q_{12}Q_{23}-Q_{13}Q_{22}$$
$$Q_{32}=Q_{10}Q_{22}+Q_{12}Q_{20}+Q_{13}Q_{21}-Q_{11}Q_{23}$$
$$Q_{33}=Q_{10}Q_{23}+Q_{13}Q_{20}+Q_{11}Q_{22}-Q_{12}Q_{21}$$

La suma de dos cuaterniones Q_1 y Q_2, se define como:

$$Q_3 = Q_1 + Q_2 = (s_1, v_1) + (s_2, v_2) = (s_1 + 8s_2, v_1 + v_2) \quad [1.6]$$

Mientras que el producto con un escalar esta definido por:

$$Q_3 = aQ_2 = a(s_2, v_2) = (as_2, av_2) \quad [1.7]$$

El producto de cuaterniones es entonces asociativo, aunque no conmutativo.

Norma e Inverso

Con las definiciones de cuaternión conjugado y producto de cuaterniones, se puede deducir que:

$$Q^{\circ}Q^{*} = (q_0^2 + q_1^2 + q_2^2 + q_3^2)e \quad [1.8]$$

Al numero real $(\theta_0^2 + \theta_1^2 + \theta_2^2 + \theta_3^2)^{1/2}$ se le denomina norma de Q y se representa por [[Q]]. El inverso de un cuaternión puede hallarse mediante la expresión:

$$Q^{-1} = Q^{*} / [[Q]] \quad [1.9]$$

siempre y cuando el cuaternión sea uno no nulo.

Por medio de las propiedades expuestas de los cuaterniones, se puede verificar su validez en el uso de representaciones en la composición de rotaciones. Por ejemplo, el cuaternión que representa el giro de un ángulo θ sobre un eje **k** es el siguiente:

$$Q = Rot\ (k, \theta) = (\cos \theta/2, k \operatorname{sen} \theta/2) \quad [1.10]$$

La aplicación de la rotación expresada por el cuaternión Q a un vector **r**, estará definida con el siguiente producto.

$$Q°(0, \mathbf{r})°Q \qquad [1.11]$$

La composición de rotaciones con cuaterniones es tan simple como multiplicar cuaterniones entre sí.

Lo único que se debe de considerar a la hora de componer rotaciones, es precisamente que la multiplicación de cuaterniones no es conmutativa, por lo cual se debe de tener en cuenta el orden en que se aplican cada una de las rotaciones.

Para tener un panorama más claro de lo que implica la composición de rotaciones, es recomendable tener presentes los conceptos de transformación lineal. Hay que recordar básicamente que por medio de la transformación lineal se puede conocer un vector, desde un sistema coordenado diferente, incluso aunque éste nuevo sistema no sea ortogonal.

Cuando se quiere componer rotaciones y traslaciones, se procede de la siguiente manera. El resultado de aplicar la traslación dada por el vector **p** seguida de un rotación Q al sistema OXYZ, es un nuevo sistema OUVW, tal que las coordenadas del vector **r** en el sistema OXYZ, conocidas en OUVW, serán:

$$(0, \mathbf{r}xyz) = Q °(0, \mathbf{r}uvw)° Q^* + (0, \mathbf{r}) \qquad [1.12]$$

El resultado de rotar y luego trasladar, estará definido por:

$$(0, \mathbf{r}xyz) = Q°(0, \mathbf{r}uvw + \mathbf{p}) ° Q^* \qquad [1.13]$$

Si se mantiene el sistema OXYZ fijo y se traslada el vector **r** según **p** y luego se rota según Q, se obtendrá $\mathbf{r}'$:

$$(0, \mathbf{r}') = Q ° (0, \mathbf{r}) ° Q^* + (0, \mathbf{p}) \qquad [1.14]$$

Por lo visto, para representar composiciones de rotaciones y traslaciones, el método de los cuaterniones es uno de lo más prácticos a la hora de aplicarse en sistemas computacionales, de ahí su uso, aunque en la actualidad limitado, pero en expansión.

Algoritmo de Denavit-Hartenberg para la obtención del modelo cinemático directo

1. Numerar los eslabones comenzando con 1 (primer eslabón móvil de la cadena) y acabando con *n* (ultimo eslabón móvil). Se numerara como eslabón 0 a la base fija del robot.

2. Numerar cada articulación comenzando por 1 (la correspondiente al primer grado de libertad) y acabando en *n*.

3. Localizar el eje de cada articulación. Si esta es rotativa, el eje será su propio eje de giro. Si es prismática, será el eje a lo largo del cual se produce el desplazamiento.

4. Para *i* de 0 a n-1 situar el eje $\mathbf{z}_i$, sobre el eje de la articulacion *i* + 1 .

5. Situar el origen del sistema de la base $\{S_0\}$ en cualquier punto del eje $\mathbf{z}_0$. Los ejes $\mathbf{x}_0$ e $\mathbf{y}_0$ se situaran de modo que formen un sistema dextrógiro con $\mathbf{z}_0$.

6. Para *i* de 1 a n-1, situar el sistema $\{S_i\}$ (solidario al eslabón i) en la intersección del eje z, con la línea normal común a $\mathbf{z}_{i-1}$ y $\mathbf{z}_i$. Si ambos ejes se cortasen se situaría $\{S_i\}$ en el punto de corte. Si fuesen paralelos {Si} se situaría en la articulación *i* + l.

7. Situar $\mathbf{x}_i$ en la línea normal común a $\mathbf{z}_{i-1}$ y $\mathbf{z}_i$.

8. Situar $\mathbf{y}_i$ de modo que forme un sistema dextrógiro con $\mathbf{x}_i$ y $\mathbf{z}_i$

9. Situar el sistema {Sn} en el extreme del robot de modo que $\mathbf{z}_n$ coincida con la dirección de $\mathbf{z}_{n-1}$ y $\mathbf{x}_n$ sea normal a $\mathbf{z}_{n-1}$ y $\mathbf{z}_n$.

10. Obtener θ_i como el ángulo que hay que girar en torno a $\mathbf{z}_{i-1}$ para que $\mathbf{x}_{i-1}$ y $\mathbf{x}_i$ queden paralelos.

11. Obtener **d**, como la distancia medida a lo largo de $\mathbf{z}_{i-1}$, que habría que desplazar $\{S_{i-1}\}$ para que $\mathbf{x}_i$ y $\mathbf{x}_{i-1}$ quedasen alineados.

12. Obtener $\mathbf{a}_i$ como la distancia medida a lo largo de **x**, (que ahora coincidiría con $\mathbf{x}_{i-1}$) que habría que desplazar el nuevo $\{S_{i-1}\}$ para que su origen coincidiese con $\{S_i\}$.

13. Obtener α_i como el ángulo que habría que girar en torno a $\mathbf{x}_i$ (que ahora coincidiría con $\mathbf{x}_{i-1}$ para que el nuevo $\{S_{i-1}\}$ coincidiese totalmente con {Si}.

14. Obtener las matrices de transformación ${}^{i-1}A_i$.

15. Obtener la matriz de transformación que relaciona el sistema de la base con el del extremo del robot $\mathbf{T} = {}^{0}\mathbf{A}_1, {}^{1}\mathbf{A}_2, \ldots {}^{n-1}\mathbf{A}_n$

16. La matriz T define la orientación (submatriz de rotación) y posición (submatriz de traslación) del extremo referido a la base en función de las n coordenadas articulares

Los cuatro parámetros de D-H (θi, di, ai, αi) dependen únicamente de las características geométricas de cada eslabón y de las articulaciones que le unen con el anterior y siguiente. En concreto estos representan:

θ_i Es el ángulo que forman los ejes x_{i-1} y x_i medido en un plano perpendicular al eje z_{i-1}, utilizando la regia de la mano derecha. Se trata de un parámetro variable en articulaciones giratorias.

d_i Es la distancia a lo largo del eje z_{i-1} desde el origen del sistema de coordenadas (i-1)esimo hasta la intersección del eje z_{i-1} con el eje x_i. Se trata de un parámetro variable en articulaciones prismáticas.

a_i Es la distancia a lo largo del eje x_i que va desde la intersección del eje z_{i-1} con el eje x_i hasta el origen del sistema i-esimo, en el caso de articulaciones giratorias. En el caso de articulaciones prismáticas, se calcula como la distancia mas corta entre los ejes z_{i-1} y z_i

α_i Es el ángulo de separación del eje z_{i-1} y el eje z_i, medido en un plano perpendicular al eje x_i, utilizando la regla de la mano derecha.

Una vez obtenidos los parámetros D-H, el cálculo de las relaciones entre los eslabones consecutivos del robot es inmediato, ya que vienen dadas por las matrices A, que se calculan según la expresión general de resolución para matrices de transformación homogéneas. Esas relaciones entre eslabones no consecutivos vienen dadas por las matrices T que, como ya se comento anteriormente, se obtienen como producto de un conjunto de matrices **A**.

Si bien, para la resolución del modelo cinemático por medio de cuaterniones no se utilizan los parámetros de Denavit Hartenberg, si nos sirven de guía para el establecimiento de los sistemas coordenados correspondientes a cada articulación del robot.

Ahora se verá la resolución del modelo cinemático directo de un robot tipo SCARA por medio de cuaterniones.

En la figura 1.2 se pueden apreciar los sistemas coordenados para cada uno de los eslabones del robot SCARA.

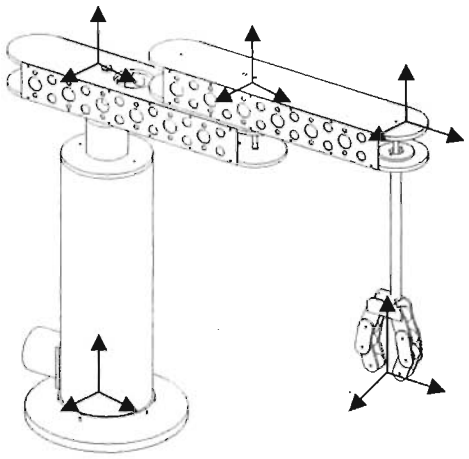

Figura 1.2. sistemas coordenados del robot tipo SCARA

El procedimiento por medio de cuaterniones consiste en obtener las expresión que permite conocer las coordenadas de posición y orientación del sistema de referencia que se asocie al robot {S4} con respecto al sistema de base {S0}. Para obtener dicha relación, se irá convirtiendo {S0}, en {S1}, {S2}, {S3} y {S4} según la siguiente serie de transformaciones:

1. Desplazamiento de {S0} una distancia q_3 a lo largo del eje Z_0 y giro un ángulo de $\mathbf{q}_1$ alrededor del eje Z_0, llegándose a {S1}.
2. Desplazamiento de {S1} una distancia $\mathbf{l}_2$ a lo largo del eje X_1 y un giro un ángulo $\mathbf{q}_2$ alrededor del nuevo eje Z, para llegar al sistema {S2}
3. desplazamiento a lo largo del eje X_2 una distancia $\mathbf{l}_3$ para llegar a la para llegar al sistema {S3}.
4. Desplazamiento de {S3} una distancia $\mathbf{l}_1$ a lo largo del eje Z_3 y un giro en torno a Z_4 de un ángulo $\mathbf{q}_4$, llegándose finalmente a {S4}.

Las transformaciones quedan definidas entonces como:

$$\begin{aligned} &S_0 - S_1\text{: } \mathbf{T}(Z, \mathbf{q}_3)\quad \mathbf{Rot}(z, \mathbf{q}_1) \\ &S_1 - S_2\text{: } \mathbf{T}(X, \mathbf{l}_2)\quad \mathbf{Rot}(z, \mathbf{q}_2) \\ &S_2 - S_3\text{: } \mathbf{T}(X, \mathbf{l}_3)\quad \mathbf{Rot}(z, 0) \\ &S_3 - S_4\text{: } \mathbf{T}(z, -\mathbf{l}_1)\quad \mathbf{Rot}(z,\mathbf{q}_4) \end{aligned} \qquad [1.15]$$

Donde se definen los desplazamientos y giros con los siguientes vectores:

$P1 = (0, 0, \mathbf{q_3})$ $\quad Q1 = (\mathbf{C_1}, 0, 0, \mathbf{S_1})$ [1.16]
$P2 = (\mathbf{l_2}, 0, 0)$ $\quad Q2 = (\mathbf{C_2}, 0, 0, \mathbf{S_2})$
$P3 = (\mathbf{l_3}, 0, 0)$ $\quad Q3 = (1, 0, 0, 0)$
$P4 = (0, 0, -\mathbf{l_1})$ $\quad Q4 = (\mathbf{C_4}, 0, 0, \mathbf{S_4})$

Donde:

$\mathbf{C_i} = \cos(\mathbf{q_i}/2)$ $\quad \mathbf{S_i} = \text{sen}(\mathbf{q_i}/2)$ [1.17]

Aplicando las ecuaciones definidas anteriormente sobre el uso de cuaterniones, un objeto localizado en el sistema de referencia $\{S_1\}$ por su vector de posición $\mathbf{a_i}$ y su cuaternión de rotación $\mathbf{R_i}$, tendrá en el sistema de referencia $\{S_{i-1}\}$ el vector de posición $\mathbf{a_{i-1}}$ y el cuaternión $\mathbf{R_{i-1}}$ siguientes:

$$(0,\mathbf{a_{i-1}}) = Q_i(0,\mathbf{a_i})Q_i^* + (0,\mathbf{p_i}) \quad [1.18]$$
$$\mathbf{R_{i-1}} = Q_1\mathbf{R_i}$$

Donde $\mathbf{p_i}$ y Q_i son respectivamente el desplazamiento y posterior rotación que permite convertir $\{S_{i-1}\}$ en $\{S_i\}$. Aplicando repetidamente la expresión a los demás sistemas de referencia, es decir i = 0,1,2,3,4, se tendrá:

$$(0, \mathbf{a_0}) = Q_1\,(0,\mathbf{a_1})Q_1^* + (0,\mathbf{p_1}) \quad [1.19]$$
$$\mathbf{R_0} = Q_1\mathbf{R_1}$$
$$(0, \mathbf{a_1}) = Q_2\,(0,\mathbf{a_2})Q_2^* + (0,\mathbf{p_2})$$
$$\mathbf{R_1} = Q_2\mathbf{R_2}$$
$$(0, \mathbf{a_2}) = Q_3\,(0,\mathbf{a_3})Q_3^* + (0,\mathbf{p_3})$$
$$\mathbf{R_2} = Q_3\mathbf{R_3}$$
$$(0, \mathbf{a_3}) = Q_4\,(0,\mathbf{a_4})Q_4^* + (0,\mathbf{p_4})$$
$$\mathbf{R_3} = Q_4\mathbf{R_4}$$

Se sustituyen de manera consecutiva las expresiones, como se había mencionado antes, solo hay que multiplicar cuaterniones entres sí de manera consecutiva para obtener el sistema deseado, y se obtiene:

$$(0, \mathbf{a}_0) = Q_1[Q_2[Q_3[Q_4(0,\mathbf{a}_4)Q_4^*+(0,\mathbf{p}_4)]Q_3^*+(0,\mathbf{p}_3)]Q_2^*+ (0,\mathbf{p}_2)]Q_1^*+(0,\mathbf{p}_1)=$$

$$Q_1Q_2Q_3Q_4(0,\mathbf{a}_4)Q_4^*Q_3^*Q_2^*Q_1^* + Q_1Q_2Q_3(0,\mathbf{p}_4)Q_3^*Q_2^*Q_1^* + Q_1Q_2(0,\mathbf{p}_3)Q_2^*Q_1^*+Q_1(0,\mathbf{p}_2)Q_1^*+(0,\mathbf{p}) =$$

$$Q_{1234}(0,\mathbf{a}_4)Q^*_{1234} + Q_{123}(0,\mathbf{p}_4)Q^*_{123} + Q_{12}(0,\mathbf{p}_3)Q^*_{12}+Q_1(0,\mathbf{p}_2)Q_1^*+ (0,\mathbf{p}) \qquad [1.20]$$

Donde:

$$Q_{1234} = Q_1Q_2Q_3Q_4 = (\mathbf{C}_{124},0,0,\mathbf{S}_{124}) \qquad [1.21]$$
$$Q_{123} = Q_1Q_2Q_3 = (\mathbf{C}_{12},0,0,\mathbf{S}_{12})$$
$$Q_{12} = Q_1Q_2 = (\mathbf{C}_{12},0,0,\mathbf{S}_{12})$$
$$Q^*_{ij} = (Q_iQ_j)^* = Q_i^*Q_j^*$$

Desarrollando productos se tiene

$$Q_{1234}(0,\mathbf{a}_4)Q^*_{1234} = Q_{1234}(0,\mathbf{a}_{4x}, \mathbf{a}_{4y}, \mathbf{a}_{4z})Q^*_{1234}=$$

$$(-\mathbf{S}_{124}\mathbf{a}_{4z}, \mathbf{C}_{124}\mathbf{a}_{4x} - \mathbf{S}_{124}\mathbf{a}_{4y}, \mathbf{C}_{124}\mathbf{a}_{4y} - \mathbf{S}_{124}\mathbf{a}_{4x}, \mathbf{C}_{124}\mathbf{a}_{4z})Q^*_{1234}=$$

$$(0,\mathbf{C}_{112244}\mathbf{a}_{4x}-\mathbf{S}_{112244}\mathbf{a}_{4y}, \mathbf{C}_{112244}\mathbf{a}_{4y}-\mathbf{S}_{112244}\mathbf{a}_{4x}, \mathbf{a}_{4z})$$

$$Q_{123}(0,\mathbf{p}_4)Q^*_{123} + Q_{12}(0,\mathbf{p}_3)Q^*_{12} =$$

$$Q_{12}(0,\mathbf{p}_4 + \mathbf{p}_3)Q^*_{12} = (0,\mathbf{l}_3\mathbf{C}_{1122}, \mathbf{l}_3\mathbf{S}_{1122},-\mathbf{l}_1$$

$$Q_1(0, \mathbf{p}_2)Q_1^* = (0,\mathbf{l}_2\mathbf{C}_{11},\mathbf{l}_2\mathbf{S}_{11},0) \qquad [1.22]$$

Y como $(0,\mathbf{p}_1) = (0,0,0,\mathbf{q}_3)$ finalmente resulta:

$$(0,\mathbf{a}_0) = (0,\mathbf{a}_{4x}\,\mathbf{C}_{112244} - \mathbf{a}_{4x}\,\mathbf{S}_{112244} - \mathbf{a}_{4y}\,\mathbf{S}_{112244} + \mathbf{l}_3\mathbf{C}_{1122} + \mathbf{l}_2\mathbf{C}_{11},$$
$$\mathbf{a}_{4y}\,\mathbf{C}_{112244} - \mathbf{a}_{4x}\mathbf{S}_{112244} + \mathbf{l}_3\mathbf{S}_{1122} + \mathbf{l}_2\mathbf{S}_{11},\ \mathbf{a}_{4z} - \mathbf{l}_1 + \mathbf{q}_3) \qquad [1.23]$$

Mientras para definir la orientación de un objeto, se tiene:

$$\mathbf{R}_0 = \mathbf{Q}_1\mathbf{Q}_2\mathbf{Q}_3\mathbf{Q}_4\mathbf{R}_4 = \mathbf{Q}_{1234}\mathbf{R}_4 = (\mathbf{C}_{124},0,0,\mathbf{S}_{124}) \qquad [1.24]$$

De esta forma, las expresiones 1.1.23 y 1.1.24 nos permiten conocer la posición $\mathbf{a}_0$ y orientación $\mathbf{R}_0$ de un objeto en el sistema $\{S_0\}$ conocidas en el sistema $\{S_4\}$, si este objeto esta posicionado en el extremo del robot, es decir en el origen de $\{S_4\}$, se tiene que:

$$\mathbf{a}_4 = (0,0,0) \qquad \mathbf{R}_4 = (1,0,0,0) \qquad [1.25]$$

con lo que

$$(0,\mathbf{a}_0) = (0,\mathbf{l}_3\mathbf{C}_{1122} + \mathbf{l}_2\mathbf{C}_{11},\ \mathbf{l}_3\mathbf{S}_{1122} + \mathbf{l}_2\mathbf{S}_{11},\ \mathbf{q}_3 - \mathbf{l}_1)$$
$$\mathbf{R}_0 = (\mathbf{C}_{124},0,0,\mathbf{S}_{124}) \qquad [1.26]$$

Lo que significa que el extremo del robot estará posicionado, con respecto a su base, en:

$$\mathbf{X} = \mathbf{a}_{0x} = \mathbf{l}_3\cos(\mathbf{q}_1+\mathbf{q}_2) + \mathbf{l}_2\cos\mathbf{q}_1 \qquad [1.27]$$
$$\mathbf{Y} = \mathbf{a}_{0y} = \mathbf{l}_3\,\mathrm{sen}(\mathbf{q}_1+\mathbf{q}_2) + \mathbf{l}_2\,\mathrm{sen}\mathbf{q}_1$$
$$\mathbf{Z} = \mathbf{a}_{0z} = \mathbf{q}_3 - \mathbf{l}_1$$

Y girado respecto a la base un ángulo $\mathbf{q}_1 + \mathbf{q}_2 + \mathbf{q}_4$ en torno al eje **z** según

$$\mathbf{Rot}\,(\mathbf{z},\ \mathbf{q}_1+\mathbf{q}_2+\mathbf{q}_4) \qquad [1.28]$$

Como se puede apreciar, el método de resolución cinemática por cuaterniones termina definiendo la orientación y posición del robot en términos trigonométricos, por lo cual, en el momento de resolver la cinemática inversa podríamos vernos envueltos en las indeterminaciones clásicas que nos causa la

función Arco tangente, cuando se tocan orientación con ángulos iguales a 90 grados en alguna de las articulaciones. Tienen como ventajas, el poder involucrar más grados e inclusive considerar con gran facilidad la herramienta con solo agregar un vector en el último sistema de referencia {S4}. El objetivo de este trabajo no será implementar los cuaterniones en este tipo de manipulador, pues es algo que se ha hecho con anterioridad, por el contrario, se intentará comprobar la validez de otra teoría, y ponerla en práctica para demostrar su funcionalidad en un modelo real.

1.2. Teoría de mecanismos

Análisis de Posición.

Dentro de la teoría de mecanismos, se estudia con detalle la integración de mecanismos de diversas configuraciones con fines específicos de movimiento. En la síntesis analítica de eslabonamientos se puede destacar la del mecanismo formado con cuatro barras, debido a la gran similitud que existe entre su metodología y la resolución de una cadena cinemática abierta por el método geométrico usada en algunos textos de robótica. En el método geométrico se considera que los dos grados principales del manipulador tipo SCARA, se encuentran dentro de un mismo plano, permitiendo que su resolución se pueda determinar mediante las ecuaciones inversas de las proyecciones de ambos eslabones en los ejes coordenados como se vio en el principio de este capítulo.

El método geométrico ofrece la ventaja de ser sencillo y adecuado en el caso de esta configuración de manipulador, pero su teoría no es muy amplia en el campo de la dinámica, se limita solamente a la cinemática y cinemática inversa, puesto que su estudio se abandona en este punto, mientras en el problema dinámico, los cuaterniones son empleados en sistemas computarizados para intentar resolverlo, aunque se encuentran todavía en líneas de investigación.

A diferencia del método geométrico, la teoría de mecanismos posee un amplio estudio en los campos cinemático y dinámico, lo que permite que si un mecanismo es modelado de la forma correcta, se obtengan los parámetros de velocidad y aceleración correspondientes en cada punto. Debe tenerse en cuenta que el manipular ecuaciones de movimiento con la ayuda de la computadora no debe de hacerse a la ligera, puesto que si no se conoce la teoría en la que se basa, se estará tratando el problema a ciegas.

Cabe destacar que la teoría de mecanismos nunca se ha empleado en robótica, debido a que su tratamiento es más complejo que los métodos tradicionales, pero si se esta familiarizado con ella, resulta sencillo modelar cualquier cadena cinemática abierta con dicha teoría. A fin de cuentas, mecanismos.

Se comenzará por el problema cinemático inverso, que nos interesa conocer en un principio para determinar las posiciones angulares de nuestro manipulador para alcanzar un determinado punto en el plano XY, el punto en el eje Z estará dado por la posición de la junta prismática exclusivamente, como sucede en todos los manipuladores de este tipo.

Se analizará ahora la síntesis analítica de un eslabonamiento de cuatro barras, por medio del método de lazo vectorial cerrado.

En la figura 1.2 se puede apreciar la configuración y parámetros de un eslabonamiento de cuatro barras.

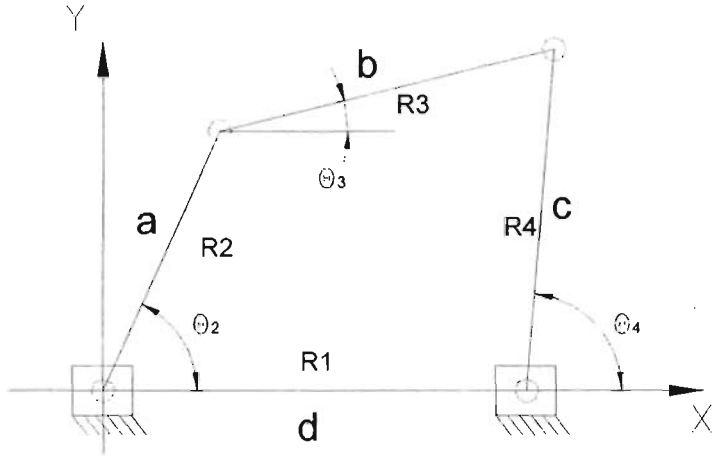

Figura 1.2. Mecanismo de 4 barras y Parámetros

En ella se puede apreciar el modelado de las cuatro barras como vectores, siendo la suma de ellos cero, representado cada uno por un vector de posición. En la teoría de mecanismos clásica, se conoce la magnitud de cada uno de los eslabones así como el ángulo de entrada θ_2, y en base a ello se determinan θ_3 y θ_4. En la configuración de nuestro manipulador también se conoce la longitud de los 4 eslabones, dos de ellos correspondientes a los eslabones propios del manipulador y los otros dos las coordenadas X y Y que se desean alcanzar. El ángulo que se conoce será el formado por las componentes X y Y precisamente,

que será siempre de 90°. Bien así, se trabajará en busca de los dos ángulos necesarios para la síntesis de la cadena cinemática deseada.

Para representar los vectores nos basaremos en el uso de números complejos, siendo así la parte real la coordenada en el eje X y la parte imaginaria la coordenada en el eje Y, formando así un plano complejo, con cualquiera de las siguientes notaciones:

Forma polar	Forma cartesiana
R@<T	$r\cos\theta\mathbf{i} + r\,\mathrm{sen}\theta\,\mathbf{j}$
$re^{j\theta}$	$r\cos\theta + j\,r\,\mathrm{sen}\theta$

Cabe destacar que el símbolo *j* se emplea como un operador y no como un valor, debido a que este no se puede evaluar numéricamente por ser la raíz cuadrada de –1.

La razón por la cual se emplea la notación de números complejos para representar vectores, se debe a la identidad de Euler:

$$e^{\pm j\theta} = \cos\theta \pm j\,\mathrm{sen}\theta \qquad [\ 1.29\ ]$$

De esta forma cualquier vector correspondiente a un espacio bidimensional se puede representar de la forma compacta apreciada en el lado izquierdo de la ecuación, siendo esta una función extremadamente fácil de diferenciar o integrar, ya que tal función es igual a su propia derivada.

$$\frac{de^{j\theta}}{dT} = je^{j\theta} \qquad [\ 1.30\ ]$$

así pues, es claro que modelando un mecanismo de esta manera, se puede fácilmente derivar las funciones obtenidas para buscar las ecuaciones de velocidad y aceleración.

Ahora se analizará la posición de los vectores mostrados en la figura 1.2, los cuales se definen en función del ángulo que se desea encontrar, recordando que el ángulo de un vector se mide en su raíz. Así queda la ecuación de lazo cerrado:

$$R2 + R3 - R4 - R1 = 0 \qquad [1.31]$$

Para sustituir en el siguiente paso por la notación de números complejos para cada vector de posición, sustituyendo sus magnitudes por los parámetros a, b, c, y d tal y como se aprecian en la figura 1.2, resultando:

$$ae^{j\theta 2} + be^{j\theta 3} - ce^{j\theta 4} - de^{j\theta 1} \qquad [1.32]$$

Las anteriores son dos formas de la misma ecuación vectorial, las cuales se pueden resolver para dos incógnitas. De éstas ecuaciones se desprende el resto de la teoría de mecanismos guiándonos a las diversas soluciones dependiendo de los parámetros que son conocidos y lo que se busca. En el caso particular, se conoce el valor de las cuatro barras, por ser dos de ellas constantes y dos variables conocidas en el instante, y dos de los ángulos, dejando como incógnitas dos variables más, los ángulos de posición.

Resulta claro, que estos dos ángulos serán funciones de los demás parámetros:

$$\theta_3 = f\{a, b, c, d, \theta_2\}$$
$$\theta_4 = f\{a, b, c, d, \theta_2\} \qquad [1.33]$$

Nótese que se ha omitido θ_1 debido a que ésta es siempre 0°, por ser colineal al eje de las abcisas. Esto claro por conveniencia.

Ahora, para resolver la ecuación polar, se deben de sustituir los términos $e^{j\theta}$ por los equivalentes de Euler, y después separar la ecuación vectorial en dos ecuaciones escalares que se pueden resolver simultáneamente para θ3 y θ4:

$$a(\cos\theta_2 + j\,\mathrm{sen}\theta_2) + b(\cos\theta_3 + j\,\mathrm{sen}\theta_3) - c(\cos\theta_4 + j\,\mathrm{sen}\theta_4) - d(\cos\theta_1 + j\,\mathrm{sen}\theta_1) = 0 \qquad [1.34]$$

Se separa ahora en parte real e imaginaria, y eliminando j por división

$$a(\cos\theta_2) + b(\cos\theta_3) - c(\cos\theta_4) - d(\cos\theta_1) = 0 \qquad [1.35a]$$

$$a(\text{sen}\theta_2) + b(\text{sen}\theta_3) - c(\text{sen}\theta_4) - d(\text{sen}\theta_1) = 0 \qquad [1.35b]$$

Con $\theta_1 = 0$

$$a(\cos\theta_2) + b(\cos\theta_3) - c(\cos\theta_4) - d = 0 \qquad [1.35a]$$

$$a(\text{sen}\theta_2) + b(\text{sen}\theta_3) - c(\text{sen}\theta_4) = 0 \qquad [1.35b]$$

El resultado son dos ecuaciones escalares que pueden ser resueltas simultáneamente para θ_3 y θ_4, para lo cual, primero se rescribe para despejar θ_3.

$$b\cos\theta_3 = -a\cos\theta_2 + c\cos\theta_4 + d \qquad [1.36a]$$

$$b\,\text{sen}\theta_3 = -a\,\text{sen}\theta_2 + c\,\text{sen}\theta_4 \qquad [1.36b]$$

A continuación se elevan al cuadrado los dos términos de la ecuación 1.36 (a) y (b) se suman

$$b^2(\text{sen}^2\theta_3 + \cos^2\theta_3) = (-a\,\text{sen}\theta_2 + c\,\text{sen}\theta_4)^2 + (-a\cos\theta_2 + c\cos\theta_4 + d)^2 \qquad [1.37]$$

La expresión entre paréntesis del lado izquierdo es por definición 1, quedando de esta forma la ecuación en términos de θ_4, la cual puede ser resuelta.

$$b^2 = (-a\,\text{sen}\theta_2 + c\,\text{sen}\,\theta_4)^2 + (-a\cos\theta_2 + c\cos\theta_4 + d)^2 \qquad [1.38]$$

Ahora se desarrolla el lado derecho de la ecuación.

$$b^2 = a^2 + c^2 + d^2 - 2ad\cos\theta_2 + 2cd\cos\theta_4 - 2ac(\text{sen}\theta_2\,\text{sen}\theta_4 + \cos\theta_2\cos\theta_4) \qquad [1.39]$$

Para simplificar la expresión se definen las constantes K_1, K_2 y K_3 en términos de las longitudes de eslabón constantes.

$$K_1 = d / a \qquad K_2 = d / c \qquad K_3 = (a^2 - b^2 + c^2 + d^2) / 2ac \qquad [\ 1.40\ ]$$

Y así queda

$$K_1 \cos\theta_4 - K_2 \cos\theta_2 + K_3 = \cos\theta_2 \cos\theta_4 + \operatorname{sen}\theta_2 \operatorname{sen}\theta_4 \qquad [\ 1.41\ ]$$

"Se introduce la identidad $\cos(\theta_2 - \theta_4) = \cos\theta_2 \cos\theta_4 + \operatorname{sen}\theta_2 \operatorname{sen}\theta_4$ quedando la ecuación en la forma conocida como *Ecuación de Freudenstein*"[1]

$$K_1 \cos\theta_4 - K_2 \cos\theta_2 + K_3 = \cos(\theta_2 - \theta_4) \qquad [\ 1.42\ ]$$

Con el fin de simplificar la ecuación para una resolución más simple, se introducen las identidades trigonométricas del *Angulo mitad* quedando como sigue.

$$\operatorname{sen}\theta_4 = \{2 \tan\ (\theta_4 / 2)\} / \{\ 1 + \tan^2 (\theta_4 / 2)\} \qquad [\ 1.43a]$$

$$\cos\theta_4 = \{1 - \tan^2(\theta_4 / 2)\} / \{1 + \tan^2(\theta_4 / 2)\} \qquad [\ 1.43b]$$

Se obtiene como resultado la siguiente ecuación simplificada donde las longitudes del eslabón y el termino conocido de entrada θ_2, se han agrupado como las constantes A, B, C.

$$A \tan^2 (\theta_4 / 2) + B \tan(\theta_4 / 2) + C = 0 \qquad [\ 1.44\ ]$$

Donde: $A = \cos\theta_2 - K_1 - K_2 \cos\theta_2 + K_3$

$B = -2 \operatorname{sen}\theta_2$

$C = K_1 - (K_2 + 1) \cos\theta_2 + K_3$

[1] Robert L. Norton, Diseño de Máquinas, Mc Graw Hill, pp. 134 Hace referencia a Freudestein

Se puede apreciar a simple vista que la ecuación 1.45 es de la forma cuadrática, la cual se puede resolver con el método tradicional.

$$\theta_4 = 2 \arctan \left(\frac{-B \pm \sqrt{B^2 - 4AC}}{2A} \right) \qquad [1.45]$$

Como es bien conocido, la ecuación 1.2.19 tiene dos soluciones, una de ellas representará el mecanismo real o abierto, mientras la otra nos dará la configuración imaginaria o cerrada. La decisión de cual de las dos posiciones se debe tomar, correrá a cargo del software, que decidirá la ruta más corta para el posicionamiento.

Análogamente se resuelven las ecuaciones para θ_3, con lo cual nos queda

$$K_1 \cos\theta_3 + K_4 \cos\theta_2 + K_5 = \cos\theta_2 \cos\theta_3 + \text{sen}\theta_2 \, \text{sen}\theta_3 \qquad [1.46]$$

Donde se define K_4 y K_5 como

$$K_4 = d / b \qquad K_5 = (c^2 - d^2 - a^2 - b^2) / 2ab \qquad [1.47]$$

Que también se reduce a la forma cuadrática

$$D \tan^2 (\theta_3 / 2) + E \tan(\theta_3 / 2) + F = 0 \qquad [1.48]$$

Donde: $D = \cos \theta_2 - K_1 - K_4 \cos\theta_2 + K_5$

$E = -2 \, \text{sen}\theta_2$

$F = K_1 - (K_4 - 1) \cos\theta_2 + K_5$

Y cuya solución esta dada por

$$\theta_3 = 2 \arctan \left(\frac{-E \pm \sqrt{E^2 - 4DF}}{2D} \right) \qquad [1.49]$$

que de igual forma posee dos soluciones, nuevamente una para el mecanismo real o abierto, y otra para el imaginario o cerrado.

De esta forma se concluye el análisis cinemático para un mecanismo de cuatro barras, en nuestro caso lazo abierto. Ahora que se conoce la posición en todo momento de nuestros eslabonamientos, se puede proceder al análisis de velocidad y aceleración.

Análisis de Velocidad

Primeramente, se definirá lo que es la velocidad, la cual se conoce como la derivada de la posición con respecto al tiempo, o en palabras más explícitas, la tasa de variación de la posición (**R**) respecto del tiempo. Como la posición es un vector, nuestra velocidad será vectorial también, dividiéndose en angular y lineal. La angular se representará con ω, mientras que la velocidad lineal será representada por **V.** De esta forma

$$\omega = d\theta / dt \qquad \mathbf{V} = d\mathbf{R} / dt \qquad [1.50]$$

En la figura 1.3 se puede apreciar un eslabón en rotación.

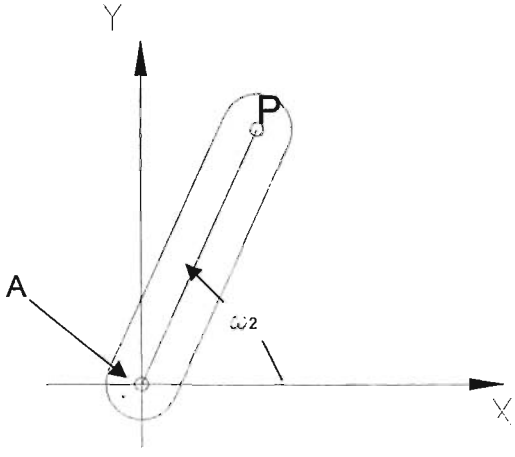

Figura 1.3. Eslabón en rotación pura

Nos interesa la velocidad del punto P, cuando la velocidad es ω, así que se puede representar el vector de posición como $\mathbf{R}_{PA}$ como numero complejo en su forma polar

$$\mathbf{R}_{PA} = pe^{j\theta} \qquad [1.51]$$

Donde p es la longitud del vector, ecuación que se puede derivar fácilmente para obtener

$$\mathbf{V}_{PA} = d\,\mathbf{R}_{PA} / dt = pje^{j\theta}\, d\theta / dt = p\,\omega\, je^{j\theta} \qquad [\,1.52\,]$$

Se puede apreciar que a causa de la derivación, la expresión de velocidad ha sido multiplicada por el operador j, lo cual representa una rotación de 90° con respecto al vector de posición, afirmando matemáticamente que la velocidad será perpendicular al vector de posición.

Ahora, se incluirá la identidad de Euler en la ecuación [1.52] para obtener las componentes real e imaginaria de la velocidad.

$$\mathbf{V}_{PA} = p\omega\, j(\cos\theta + j\,\mathrm{sen}\theta) + p\omega(-\mathrm{sen}\theta + j\cos\theta) \qquad [\,1.53\,]$$

La velocidad $\mathbf{V}_{PA}$ puede considerarse velocidad absoluta cuando el pivote A es fijo.

Ahora se recurrirá a la ecuación de lazo cerrado deducida en un principio.

$$ae^{j\theta 2} + be^{j\theta 3} - ce^{j\theta 4} - de^{j\theta 1} \qquad [\,1.54]$$

y se hará un cambio en un vector con el fin de obtener el mismo lazo cerrado con diferentes vectores, para lo cual bastará cambiar la dirección del vector b, y se tendrá

$$ae^{j\theta 2} - be^{j\theta 3} - ce^{j\theta 4} - de^{j\theta 1} \qquad [\,1.55\,]$$

ahora se derivará la ecuación con a, b, θ_1 y θ_4 constantes, se obtendrá

$$ja\omega_2 e^{j\theta 2} - jb\omega_3 e^{j\theta 3} - \mathbf{c} - \mathbf{d} = 0 \qquad [\,1.56\,]$$

donde el término **c** y **d** representan el movimiento en el plano XY del punto final de la cadena cinemática abierta, el efector final. Normalmente este tipo de

ecuación es empleada en la resolución del mecanismo Biela – Manivela – Corredera, donde **c** es constante y el desplazamiento se realiza a lo largo del eje paralelo a **d**. Se puede apreciar el mecanismo en la figura 1.4.

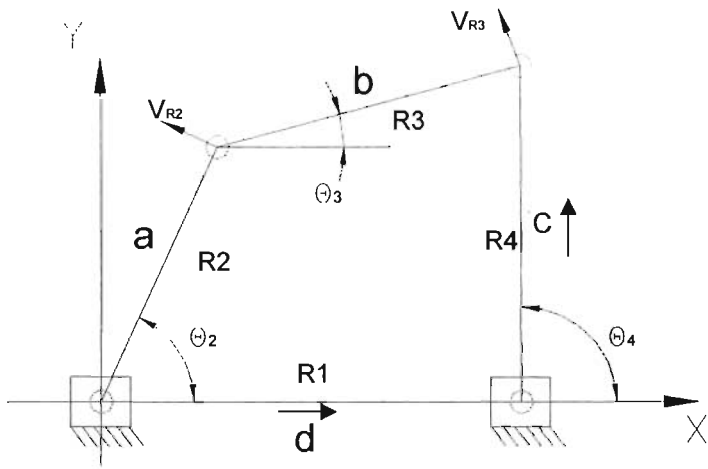

Figura 1.4. Velocidades en Mecanismo de 4 Barras

Ahora se introducirán los equivalentes de Euler en la ecuación 1.56

$$ja\omega_2 (\cos\theta_2 + j\,\mathrm{sen}\theta_2) - jb\omega_3 (\cos\theta_3 + j\,\mathrm{sen}\theta_3) - \mathbf{c} - \mathbf{d} = 0 \qquad [1.57]$$

y simplificamos

$$a\omega_2 (-\mathrm{sen}\theta_2 + j\cos\theta_2) - b\omega_3 (-\mathrm{sen}\theta_3 + j\cos\theta_3) - \mathbf{c} - \mathbf{d} = 0 \qquad [1.58]$$

Se separan las componentes real e imaginarias

Real $\quad -a\omega_2\,\mathrm{sen}\theta_2 + b\omega_3\,\mathrm{sen}\theta_3 - \mathbf{d} = 0 \qquad [1.59]$

Imaginaria $\quad a\omega_2\cos\theta_2 - b\omega_3\cos\theta_3 - \mathbf{c} = 0 \qquad [1.60]$

Estas dos ecuaciones pueden ser resueltas simultáneamente por cualquiera de los métodos tradicionales para conocer ω_2 y ω_3, en función de las velocidades deseadas en **c** y **d**, es decir, las componentes del vector de velocidad en el Plano XY.

Finalmente, las velocidad absoluta del punto A y B, y la velocidad de A con respecto a B, será:

$$V_A = a\omega 2\,(-\text{sen}\theta 2 + j\cos\theta 2)$$
$$V_{AB} = b\omega 3\,(-\text{sen}\theta 3 + j\cos\theta 3)$$
$$V_{BA} = -V_{AB} \qquad [\,1.61\,]$$

Resulta obvio, que la velocidad del punto B, o punto final resultará de la suma vectorial de las velocidades V_A y V_{AB}, que será equivalente a la velocidad del punto final en el Plano XY, como se había mencionado antes. De esta forma se cumple que

$$V_A + V_{AB} = \mathbf{c} + \mathbf{d} \qquad [\,1.62\,]$$

Análisis de Aceleración

Para concluir el análisis por medio de la teoría de mecanismos, se finalizará con el análisis de aceleración que debería de sufrir cada uno de los grados de libertad para conservar una velocidad determinada en su punto final.

El conocimiento de las aceleraciones es necesario para conocer las fuerzas dinámicas que estarán presentes en los eslabones, con $F = ma$.

Se definirá ahora el concepto de Aceleración. La aceleración, al ser derivada de la velocidad, es la tasa de variación de ésta, con respecto al tiempo. Asimismo, la aceleración será una cantidad vectorial, y puede ser lineal o angular. La aceleración angular se definirá por α y la lineal por **A.**

$$\alpha = d\omega / dt \qquad \mathbf{A} = d\mathbf{V} / dt \qquad [1.63]$$

En la figura 1.5, se muestra un eslabón en rotación pura, con pivote en el punto A del plano XY, e interesa la aceleración en el punto P, cuando el eslabón está sujeto a una velocidad angular ω y una aceleración angular α.

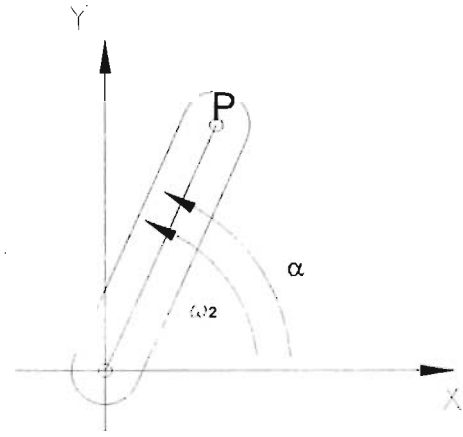

Figura 1.5. Eslabón en Rotación Pura

Nuevamente el vector de posición del eslabón esta definido por **R** y la velocidad de P es V_{PA}.

$$\mathbf{R}_{PA} = pe^{j\theta} \qquad [1.64]$$

$$\mathbf{V}_{PA} = d\,\mathbf{R}_{PA} / dt = pje^{j\theta}\, d\theta / dt = p\,\omega\, je^{j\theta} \qquad [1.65]$$

Donde p es la magnitud escalar del vector $\mathbf{R}_{PA}$. La ecuación 1.2.26 puede derivarse para encontrar la aceleración del punto P.

$$\mathbf{A}_{PA} = d\,\mathbf{V}_{PA} / dt = d\,pje^{j\theta} / dt \qquad [1.66]$$

$$\mathbf{A}_{PA} = jp\,\{\, e^{j\theta}\, d\omega/dt + j\omega\, e^{j\theta}\, d\theta/dt \,\}$$

$$\mathbf{A}_{PA} = p\alpha\, j\, e^{j\theta} - p\omega^2\, e^{j\theta}$$

$$\mathbf{A}_{PA} = \mathbf{A}^{t}_{PA} + \mathbf{A}^{n}_{PA}$$

Ahora se puede apreciar que existen dos términos en la ecuación que varían respecto al tiempo, que son θ y ω, por lo que el resultado es una ecuación con dos componentes de aceleración, normal y tangencial, como era de esperarse, debido al movimiento circular. Ahora, introduciendo los equivalentes de Euler, se obtienen las componentes del vector aceleración.

$$\mathbf{A}_{PA} = p\alpha\,(-\mathrm{sen}\theta + j\cos\theta) - p\omega^2\,(\cos\theta + j\,\mathrm{sen}\theta)$$

La aceleración $\mathbf{A}_{PA}$ puede denominarse aceleración absoluta de P, si está referida al punto A fijo en el plano.

Se retomarán las ecuaciones de lazo vectorial cerrado para nuestra cadena cinemática

$$ae^{j\theta 2} - be^{j03} - ce^{j\theta 4} - de^{j01} \qquad [1.67]$$

posteriormente se deriva con respecto al tiempo para obtener la ecuación de velocidad

$$ja\omega_2 e^{j\theta 2} - jb\omega_3 e^{j\theta 3} - \mathbf{c} - \mathbf{d} = 0 \qquad [1.68]$$

Y nuevamente se deriva para obtener la expresión correspondiente a la aceleración. Recordamos que θ1 y θ4 así como a y b son constantes.

$$(ja\alpha_2 e^{j\theta 2} + j^2 a\omega_2^2 e^{j\theta 2}) - (jb\alpha_3 e^{j\theta 3} + j^2 b\omega_3^2 e^{j\theta 3}) - \underline{\mathbf{c}} - \underline{\mathbf{d}} = 0 \qquad [1.69]$$

y simplificando.

$$(a\alpha_2 j e^{j\theta 2} - a\omega_2^2 e^{j\theta 2}) - (b\alpha_3 j e^{j\theta 3} - b\omega_3^2 e^{j\theta 3}) - \underline{\mathbf{c}} - \underline{\mathbf{d}} = 0 \qquad [1.70]$$

que es la ecuación de diferencia de aceleración

$$A_A - A_{AB} - A_B = 0 \qquad [1.71]$$

$$A_B = A_A + A_{AB}$$

Se introducen ahora los equivalentes de Euler para buscar resolver las incógnitas que se presenten

$$a\alpha_2(-\text{sen}\theta_2 + j\cos\theta_2) - a\omega_2^2(\cos\theta_2 + j\,\text{sen}\theta_2)$$
$$- b\alpha_3(-\text{sen}\theta_3 + j\cos\theta_3) + b\omega_3^2(\cos\theta_3 + j\,\text{sen}\theta_3) - \underline{\mathbf{c}} - \underline{\mathbf{d}} = 0 \qquad [1.72]$$

se separa en parte real e imaginaria

Real $-a\alpha_2\,\text{sen}\theta_2 - a\omega_2^2\cos\theta_2 + b\alpha_3\,\text{sen}\theta_3 + b\omega_3^2\cos\theta_3 - \underline{\mathbf{d}} = 0$

Imaginaria $a\alpha_2\cos\theta_2 - a\omega_2^2\,\text{sen}\theta_2 - b\alpha_3\cos\theta_3 + b\omega_3^2\,\text{sen}\theta_3 - \underline{\mathbf{c}} = 0 \qquad [1.73]$

Nuevamente se tienen dos ecuaciones simultaneas, de las cuales se conoce a, b, θ_2, θ_3, ω_2, ω_3, $\underline{\mathbf{d}}$ y $\underline{\mathbf{c}}$, y nos quedan de incógnitas α_2 y α_3.

Al conocer las aceleraciones angulares de cada eslabón resulta sencillo el calcular las demás fuerzas que se presentarían, tales como momentos, y cargas inerciales, teniendo como datos adicionales, el peso de los eslabones, su centro de masa y la carga en el final de la cadena.

1.3 Motores a Pasos.

Introducción

La mayoría de los motores convencionales de corriente continua tienen como características básicas la rotación a un determinado número de revoluciones por minuto, dependiendo del voltaje aplicado a ellos. Los motores a pasos son totalmente diferentes, puesto que no dependen de un voltaje y corriente determinados para regular su velocidad, por el contrario tienen la característica de poder ser enclavados en una posición determinada o variar su velocidad a través de pulsos digitales y poder alcanzar la posición deseada, ésta en un número determinado de grados, dependiendo del diseño del mismo.

Los motores a pasos tienen la desventaja, respecto de los servomotores, que su potencia es limitada, pero a diferencia de éstos, los motores de pasos pueden ser controlados por medio de señales digitales, como se mencionó antes, contrario a lo que sucede con los servomotores, que requieren de un control análogo.

Cabe señalar que en la aplicación se debe de estimar los pros y contras que ofrece cada uno de los tipos de motores, mientras que los motores de pasos pueden ser empleados en lazos de control abiertos, esto solo es conveniente cuando las cargas son invariables y las aceleraciones bajas, ya que si se excede el par de un motor de pasos, se pierde el posicionamiento y el sistema debe de ser reinicializado. Con un servomotor no sucede esto, ya que en todo momento se conoce la posición que ocupa el eje, pero la estabilidad de los servomotores depende del potenciómetro de retroalimentación y del circuito de control.

También puede emplearse un control de medio paso, como se describirá más adelante, y que de hecho fue empleado en este trabajo con el objeto de obtener la mejor resolución posible y suavidad de movimiento de los motores obtenidos. Existe además el llamado control de micropasos, pero éste suele complicar el diseño electrónico y la mayoría de las veces resulta mejor emplear servomotores que motores de pasos con éste tipo de control.

Aunque la teoría de este tipo de motores existe desde hace ya algún tiempo, no se habían utilizado por falta de dispositivos de interrupción adecuados, como lo son ahora los transistores de potencia o relevadores de estado sólido. Ha sido solamente con la aparición de los modernos semiconductores que se ha comenzado a popularizar su uso.

Fundamentos Básicos

Las resoluciones que pueden tener estos motores, varían según el diseño y van desde los 90° por paso, lo cual resultaría en 4 pasos por revolución, hasta resoluciones de 0.72° por paso, es decir 500 pasos por revolución.

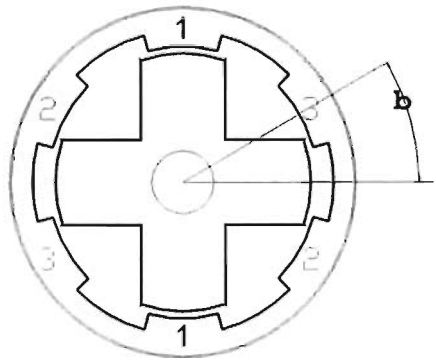

Figura 1.6 Angulo de variación en un motor de Pasos

De lo anterior se puede deducir las siguientes fórmulas, referentes a la resolución de un motor de pasos

$$\textbf{resolución} = \frac{\textbf{pasos}}{\textbf{revolución}} = \frac{360^\circ}{\beta} \qquad [1.74]$$

$$\theta = \beta \textbf{ x pasos} \qquad [1.75]$$

Donde :

β = ángulo de paso(grados/pulso)

θ = Ángulo total recorrido por el rotor (grados)

y su velocidad, de esta manera :

$$n = \frac{\beta \times \oint_p}{360},$$

donde:

$$\mathbf{n = velocidad\ del\ eje}\left(\frac{\mathbf{radianes}}{\mathbf{segundo}}\right)$$

$$\oint_p = \mathbf{frecuencia\ de\ pasos}\left(\frac{\mathbf{pulsos}}{\mathbf{segundo}}\right) \qquad [1.76]$$

Clasificación de los Motores Paso a Paso.

Los motores paso a paso, en adelante motores PAP, se clasifican principalmente en base a su funcionamiento en motores bipolares y unipolares. Los primeros sólo disponen de dos grupos de devanados, mientras que los segundos, disponen de cuatro grupos de devanados. En lo motores unipolares basta con energizar los diferentes devanados en la secuencia correcta, todos con una tierra común, mientras que los bipolares, requieren que la corriente en sus bobinas cambie de polaridad constantemente, pero tienen la ventaja de una mejor relación potencia-dimensiones ya que en los unipolares el devanado debe ser dividido en dos, mientras que en el bipolar se emplea el devanado en su totalidad.

En la figura 1.7 se puede apreciar la configuración esquemática de los motores bipolares y los unipolares.

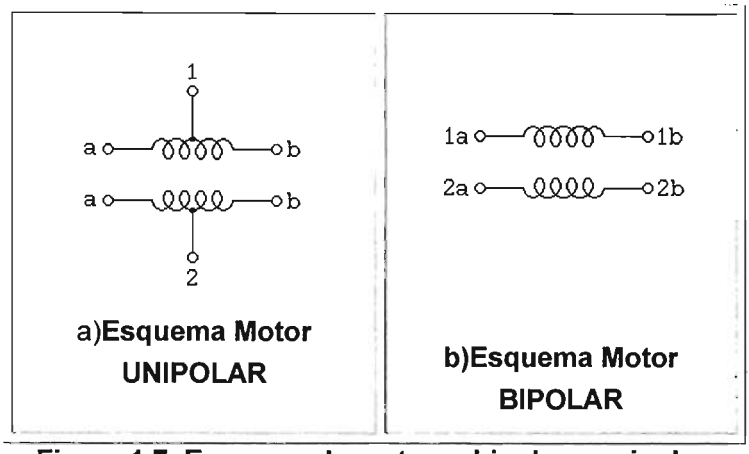

Figura 1.7. Esquema de motores bipolar y unipolar.

Se puede apreciar que la diferencia es que básicamente los unipolares poseen un tap central entre sus devanados, partiendo la bobina a la mitad.

Por otra parte, también se clasifican también en relación a la construcción de su rotor. Existen, de imán permanente, de reluctancia variable e híbridos. Los de imán permanente disponen de un rotor formado por un juego de imanes permanentes tal y como su nombre lo indica, mientras que los de reluctancia variable cuentan con un rotor dentado de hierro dulce, ocasionando que cuando una bobina se energiza, el rotor se mueva a la posición de menor reluctancia del circuito, cuando cambia esta bobina, el rotor asumirá la nueva posición, así permite que las pérdidas por magnetismo residual, que en algunos casos puede llegar a ser de un 10% de la potencia nominal, sean mínimas. Los motores de pasos híbridos son una combinación de ambos para tener las ventajas de cada uno de ellos. Desde el punto de vista de control, no existe diferencia entre un motor híbrido y uno de imán permanente.

En la figura 1.8 se puede apreciar el diagrama de un motor de reluctancia variable.

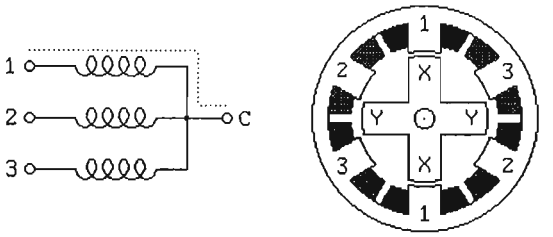

Figura 1.8. Motor de reluctancia variable.

Principios de funcionamiento

El principio de funcionamiento de los motores de paso, en adelante motores PAP, es en realidad muy sencillo, se basa en la repulsión y atracción entre los polos magnéticos del motor PAP. A esta fuerza de atracción entre el polo del estator y su correspondiente en el rotor se le conoce como torque dinámico, mientras que en el momento de estar alineados los respectivos polos, se genera lo que se conoce como torque retentivo. En la figura 1.9 se muestra la configuración típica de un motor de pasos y la forma en la que se encuentran sus polos.

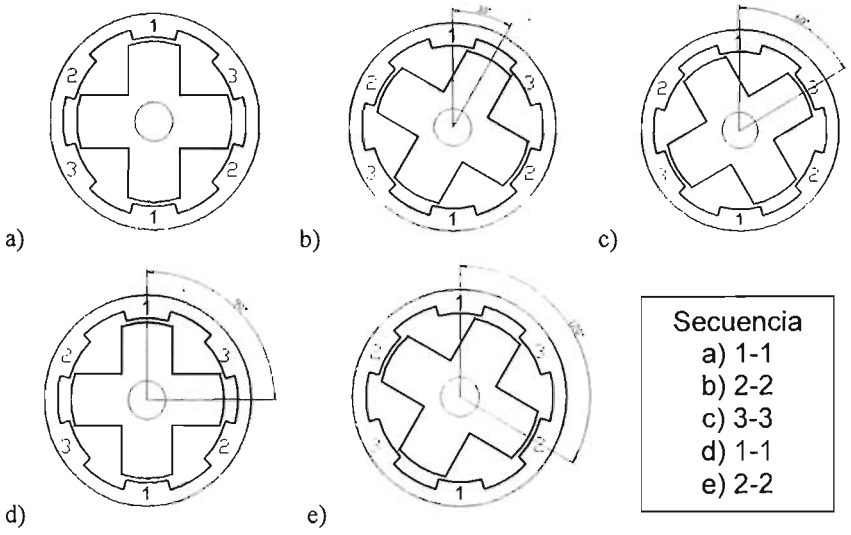

Figura 1.9 Configuración típica de los polos de un motor de pasos.

Se puede apreciar una secuencia de pasos, en la cual se puede ver que cada que se activa una bobina, el polo más cercano a ésta será atraído por la fuerza magnética hasta alinearse con ella, y dependiendo del sentido de rotación deseado, será el orden de la secuencia que se aplicará. En la figura se puede apreciar una secuencia 1,2,3,1,2 lo cual genera un movimiento de 120° en sentido horario en relación a la primera posición de enclavamiento. En este caso se considera un motor de pasos con una resolución de 30° por paso.

Además de esta forma de control, de paso completo, existen variantes, lo que se llama activación doble y funcionamiento a medio paso. En la activación doble, en vez de energizar una bobina, se energizan dos por cada paso, resultando la secuencia como sigue: AB, BC, CD, DA, AB, obteniendo el mismo resultado en cuanto a movimiento angular que la configuración de paso completo, pero en la mayoría de los motores de pasos resulta más conveniente utilizar la doble activación para mejorar el torque retentivo. De hecho la mayoría de los fabricantes dan como datos técnicos el par retentivo cuando dos de las bobinas son energizadas.

Adicionalmente, existe la configuración de medio paso, la cual consiste en combinar las secuencias de activación simple con la activación doble. Este es el sistema que se empleo en este proyecto, tratando de obtener una mayor resolución y suavidad de movimiento con los motores de que se disponen. La secuencia que se debe de seguir para obtener este tipo de activación es la siguiente: A, AB, B, BC, C, CD, D, DA, A. Con esta secuencia se obtiene el mismo desplazamiento angular que con las anteriores con la diferencia de que se dispone de una resolución del doble del nominal del motor PAP.

En la figura 1.10 se puede apreciar las secuencias adecuadas para el funcionamiento a medio paso, recordando que es una combinación de paso a simple activación y a doble activación.

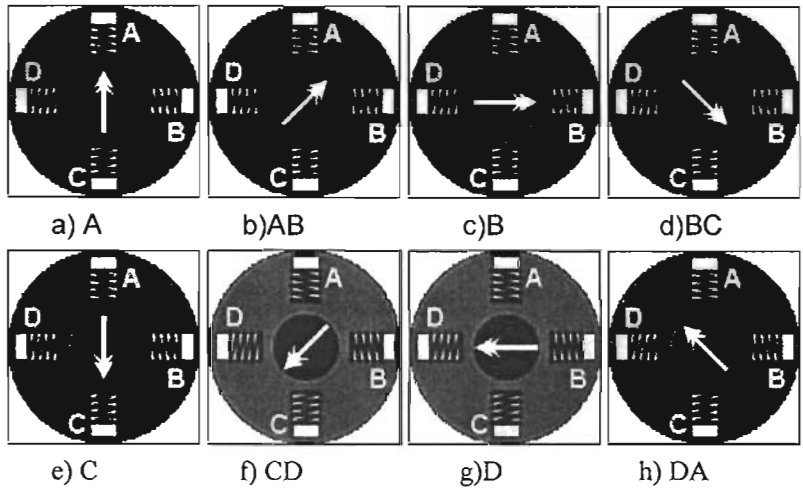

Figura 1.10. Secuencia de activación para medio paso

En ésta figura se puede apreciar la secuencia en sentido horario, para invertir el sentido de giro, simplemente se invierte la secuencia quedando A, AD, D, DC, C, BC, B, AB, A.

Para este proyecto se dispuso de 4 motores de pasos, uno por cada grado de libertad, el primero con una resolución de 200 pasos por revolución, empleado a medio paso, se obtiene una resolución de 400 pasos por revolución, y los tres restantes con una resolución de 48 pasos por revolución, que de igual manera duplican esta obteniendo 96 pasos por revolución.

Por último existe el funcionamiento de micropasos o "Microstepin'" que consiste en energizar las bobinas elevando la corriente de forma gradual, pero éste método requiere de un control análogo de cada uno de los campos, lo cual complica el diseño electrónico, haciendo más conveniente el uso de servomotores.

Control

Hay aspectos que se deben de considerar cuando se intenta controlar un motor PAP. Uno de los más importantes es la forma en que serán acelerados y frenados. No se puede enviar la frecuencia de pasos de una velocidad elevada directamente la motor de pasos, debido a que primero debe de vencer la inercia generada por los rodamientos del mismo motor, como de las masas que actúa. Un motor de pasos debe de ser acelerado gradualmente, esto se conoce como "ramping" o efecto rampa. El momento que puede generar un motor PAP cuando frena es mucho menor que el par retentivo, tal y como se aprecia en la figura 1.11.

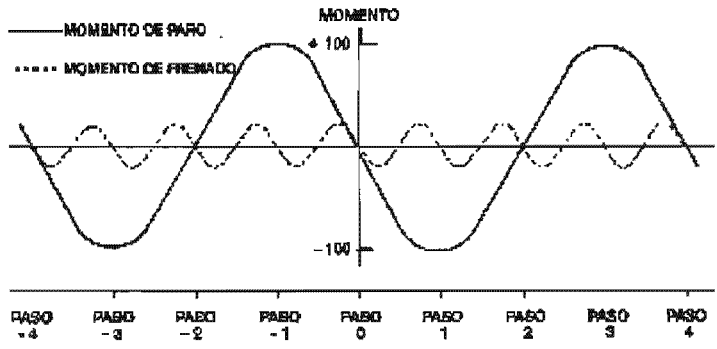

Figura 1.11. Par retentivo y par de frenado en un motor de pasos.

De esta forma se arrancará nuestro motor en una velocidad que el torque pueda vencer la inercia para después incrementar gradualmente la velocidad, posteriormente se disminuirá esta hasta el punto en que el torque de paro pueda frenar por completo el sistema.

Una forma de tener un control preciso y en tiempo real de la aceleración es por medio de una retroalimentación adecuada. Los sistemas de lazo cerrado hacen que las desventajas que presentan los motores PAP con respecto a otros se vean drásticamente disminuidas.

Hay que tener en cuenta también el comportamiento del torque con relación a la velocidad del motor de pasos, que se puede apreciar en la figura 1.12.

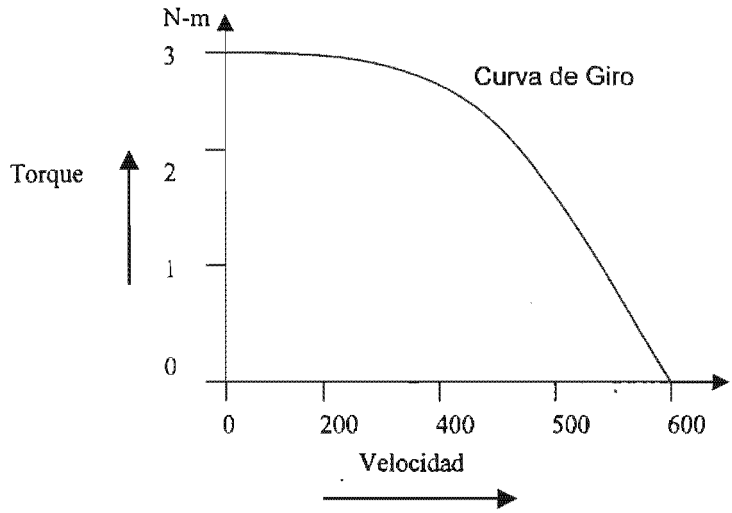

Figura 1.12. Torque contra velocidad de un motor de pasos.

Como puede apreciarse, el torque de un motor de paso disminuye conforme la velocidad aumenta y el efecto de cargas adicionales acentúa este fenómeno, por lo que se debe de tener como velocidad máxima aquella donde el torque sea capaz de mover con seguridad la masa actuada para evitar que se pierdan pasos y por lo tanto precisión. Aunque no es posible en la mayoría de los casos que el motor de pasos trabaje a su velocidad máxima cuando se le aplica una carga, para proyectos de robótica se pueden obtener velocidades muy aceptables, con un engranaje adecuado, que disminuya la velocidad y multiplique la fuerza.

En cuanto a torque retentivo se refiere, este variará de forma casi senoidal mientras el motor se encuentra en funcionamiento, como se muestra en la figura 1.13.

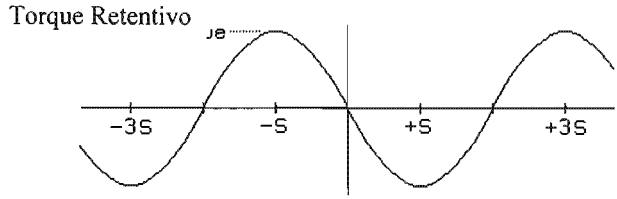

Figura 1.13. Variación del torque retentivo por paso.

Por otra parte, el torque dinámico se encuentra proporcionado por la relación matemática siguiente:

$$T = -h \sin(((\pi/2) / S) \Theta) \qquad [1.77]$$

Donde:

T = torque dinámico
h = torque retentivo
S = ángulo de paso
Θ= ángulo del eje

Cuando se utiliza la configuración de doble activación, el torque tiende a ser más elevado, debido a que son dos bobinas las que se energizan para atraer al polo del rotor, pero no llega a ser el doble de la activación simple debido a que el polo no se alinea con ninguno de ellos. Para determinar el torque con relación al de activación sencilla, se puede recurrir a la siguiente fórmula:

$$h_2 = 2^{0.5} h_1 \qquad [1.78]$$

donde:

h_1 = Torque retentivo de simple activación.
h_2 = Torque retentivo a doble activación.

Con nuestra configuración de medio paso, es claro que nuestro torque variará a cada medio paso dado, por lo que se deberá tomar en cuenta el de activación simple para futuros cálculos.

Existen para el control soluciones por medio de hardware, algunos fabricantes de semiconductores proveen con controladores para motores de paso bipolares y unipolares, aunque en la actualidad la mayoría de los sistemas que emplean motores de pasos se basan en las soluciones por software.

Al momento de diseñar el circuito se debe de tener en cuenta algunos aspectos como la inclusión de un diodo que desenergice el devanado una vez que se le retira el voltaje, para que cuando energizamos el devanado siguiente, éste no tenga que vencer fuerzas magnéticas residuales, para tener un mejor rendimiento del motor.

1.4 Microcontrolador Pic 16F84A de Microchip®

En la actualidad existen diversos controladores que integran dentro de un mismo encapsulado, un CPU[1], memoria de programa, memoria para trabajo o RAM[2] e incluso memoria EEPROM[3] para guardado de valores. También disponen de puertos, es decir, entradas y salidas de datos para interactuar con el entorno. Dentro de este gran mercado se encuentran fabricantes como Motorola® y Microchip®.

A nivel mundial son ampliamente utilizados para proyectos de robótica este tipo de controladores, ya que nos dan una gran opción de control y respuesta en tiempo real, además de ser algunos de ellos muy accesibles.

Por otra parte hay que mencionar, que su lenguaje de programación es del tipo ensamblador, es un lenguaje de bajo nivel en el cual puede llegar a ser complicada la programación de ciertas operaciones aritméticas. Por este motivo se debe considerar que tipo de programa necesitamos cargar a nuestros controladores para que escojamos uno que cubra nuestras necesidades de capacidad de procesado, velocidad y memoria.

El microcontrolador que se uso en éste proyecto fue el Pic 16F84A de la compañía Microchip®, debido a sus características que se verán a lo largo de las subsecuentes páginas de este trabajo y también a su bajo costo, lo cual permite que la experimentación con este tipo de dispositivos este al alcance de nuestro bolsillo. En la figura 1.14 podemos apreciar la configuración de las patillas del controlador.

[1] CPU. Unidad central de Proceso
[2] RAM. Memoria de acceso aleatorio que permite ser leída en forma no secuencial.
[3] EEPROM. Memoria de solo lectura escribible y borrable eléctricamente.

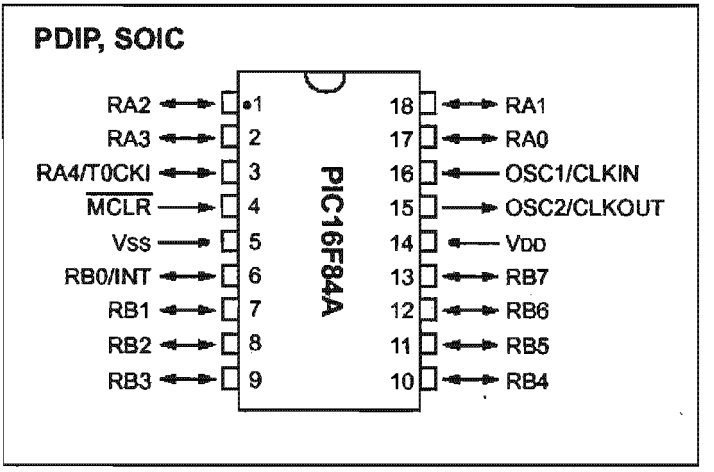

Figura 1.14. Configuración del PIC16F84A

En ella se muestra la configuración de las terminales de nuestro controlador. Sus características principales, tal y como lo describe su hoja de datos proporcionada por Microchip®, son:

1. **Características del CPU de alto desempeño.**
   - Solamente maneja 35 instrucciones fáciles de aprender.
   - Todas las instrucciones corren en un solo ciclo a excepción de los saltos y bucles, que corren en dos ciclos de instrucción.
   - La velocidad de operación puede ser de hasta 20 MHz.
   - 1024 palabras en memoria de programa, es decir, 1024 líneas de programación.
   - 68 bytes[4] de memoria RAM de datos.
   - 64 bytes de memoria EEPROM de datos.
   - Instrucciones de 14-bits.
   - Registros de 8 bits de amplitud (1 byte).
   - 15 registros de hardware para registros especiales.
   - Ocho niveles de profundidad de ruta o stack[5].
   - Direccionamiento directo, indirecto y relativo.

[4] Byte. Unidad de información constituida por 8 bits, es decir una cadena de unos y ceros de 8 dígitos
[5] Pila de almacenamiento de direcciones para retorno después de una llamada CALL.

- Cuatro Fuentes de interrupción:
  a) Pin externo RB0/INT.
  b) Sobre flujo del Temporizador TMR0.
  c) Interrupción en cambio del puerto PORTB<7:4>.
  d) Escritura por completo de la EEPROM.

2. **Características de los periféricos**
   - 13 pins de entrada / salida con direccionamiento individual.
   - Alta corriente de entrada / salida para manejo directo de LED's[6]:
     - 25 mA[7] entrada máxima por pin.
     - 25 mA salida máxima por pin.
   - TMR0: Temporizador / contador de 8-bits con preescalador programable.

3. **Características Especiales:**
   - Programación en Circuito a través de dos pines.
   - Retención de memoria hasta 40 años sin alimentación.
   - Selección de diferentes tipos de osciladores.
   - Memoria para programa tipo Flash[8]/EEPROM.

A continuación, se explica brevemente la función y configuración de los pines para entender como serán utilizados en el prototipo de la tarjeta de control para el robot.

[6] Diodo emisor de Luz.
[7] Milésimos de Ampere
[8] Flash. Nombre común con el que se conoce actualmente a la EEPROM de alta densidad utilizada como memoria de programa.

OSC1 y OSC2 (Pines 15 y 16).

Estos dos pines son entrada de señal de reloj y salida respectivamente, entre ellos se colocará un cristal de cuarzo pudiendo ser este de hasta 20 MHz[9], el cual nos dará la velocidad de trabajo del microcontrolador. En nuestro caso trabajaremos con un cristal de 4 MHz, ya que el Pic tomará 4 ciclos de reloj para cada ciclo de instrucción, lo cual nos dará una velocidad de 1 MHz. Real. En dicha velocidad nos basaremos para programar el Temporizador **TMR0** y demás retardos en el programa interno.

MCLR/RESET (PIN 4).

Por medio de este pin se conecta el voltaje de programación. Una vez que se ha cargado al PIC con la instrucción, se alimentará con 13.2 Volts para quemarla en la memoria Flash. Dicho pin también es el Reset del PIC en funcionamiento, ya que cuando la tensión en él se eleva a 5 Volts, comienza a trabajar y ejecutar el programa almacenado en memoria.

RA0, RA1, RA2, RA3, RA4, RA5/T0CKI (Pines 1,2,3,17,18)

Estos pines corresponden al Puerto A, y cada uno puede ser direccionado de forma individual como entrada o salida, soportando hasta 25 mA de corriente a 5 Volts, Siendo el RA5 una posible entrada de reloj externa para el **Timer0**/Contador en vez del cristal de cuarzo, si así se designa internamente en el PIC.

RB0/INT ,RB1,RB2,RB3,RB4,RB5,RB6,RB7 (Pines 6 al 13)

Estos pines por su parte, corresponden al Puerto B los cuales de igual manera pueden ser asignados individualmente como entradas o salidas de las mismas características que el Puerto A, solo que en adición el Pin RB0 puede ser utilizado como una interrupción externa, los pines RB4 al RB7 también pueden servir como interrupciones externas si cambian de estado y los Pines RB6 y RB7 se utilizan además para la programación serial.

[9] Mhz. Millones de Ciclos por Segundo.

VSS (PIN 5)

Este pin es tierra, es decir 0 Volts, y será también la referencia de tierra para los demás pines. Si un pin esta configurado como salida, la diferencia de potencial con VSS cuando esta desactivado será 0, mientras que si esta activado será de 5 volts. Si alguno de los pines en los puertos A o B, esta configurado como entrada, se considerará baja cuando su diferencia de potencial con respecto a VSS sea 0 y alta si es de 5 Volts.

VDD (PIN 14)

Este es el pin de alimentación, en él conectaremos la fuente de 5 volts para el funcionamiento del PIC.

Memoria y registros especiales

Dentro de la memoria del PIC podemos diferenciar tres tipos: La memoria de programa, Flash/EEPROM, la memoria de trabajo o memoria RAM y memoria para el guardado de valores, tipo EEPROM, la cual se puede escribir durante un ciclo de trabajo normal del microcontrolador, a diferencia de la memoria de programa, la cual requiere de un voltaje de 13.2 volts aproximadamente para su escritura.

La memoria RAM dentro del PIC esta dividida en dos bancos, el banco 0 y el banco 1. Dentro de la memoria, se encuentran algunos registros especiales que son los que dan la configuración al microcontrolador, veremos algunos de los registros más importantes para el proyecto.

Registros de Funciones Especiales.

Los registros de Funciones Especiales (SFR's) son utilizados por el CPU y sus periféricos para el control de la operación de nuestro dispositivo, los cuales debemos de conocer para poder configurar apropiadamente el microcontrolador para llevar a cabo nuestros propósitos. Estos registros se alojan en direcciones de memoria RAM estáticas.

En la tabla 1.2 podemos ver los registros especiales y su descripción, así como el banco de memoria correspondiente.

Tabla 1.2.- Sumario de registros especiales. Cortesía Microchip®

Dirección	Nombre	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Banco 0									
00h	INDF	Usa el contenido de FSR para almacenar datos							
01h	TMR0	Contador – Reloj en tiempo real de 8 bits							
02h	PCL	8 Bits menos significativos del contador de programa PC							
03h	STATUS	IRP	RP1	RP0	T0	PD	Z	DC	C
04h	FSR	Apuntador para almacenamiento de datos indirectamente							
05h	PORT A	-	-	-	RA4T/T0Ckl	RA3	RA2	RA1	RA0
06h	PORT B	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT
07h	-	No implementado							
08h	EEDATA	Registros para datos EEPROM							
09h	EEADR	Dirección de registros EEPROM							
0Ah	PCLATH	-	-	-	Bufer de escritura para los 5 bits altos del PC				
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
Banco 1									
80h	INDF	Usa el contenido de FSR para almacenar datos							
81h	OPTION	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
82h	PCL	8 Bits menos significativos del contador de programa PC							
83h	STATUS	IRP	RP1	RP0	T0	PD	Z	DC	C
84h	FSR	Apuntador para almacenamiento de datos indirectamente							
85h	-	-	-	Registro de dirección de datos de PORT A					
86h	Registro de dirección de datos de PORT B								
87h	-	No implementado							
88h	EECON1	-	-	-	EEIF	WRERR	WREN	WR	RD
89h	EECON2	Registro de control de la EEPROM							
8Ah	PCLATH	-	-	-	Buffer de escritura para los 5 bits altos de PC				
8Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF

El registro STATUS

Este registro contiene el banco de memoria en el que se estas trabajando y otros datos de utilidad al momento de hacer operaciones aritméticas y lógicas con registros.

Como cualquier registro, puede ser el destino de cualquier instrucción, pero si la instrucción afecta los bits Z (cero), DC (digito de acarreo), o C (Acarreo), estos tres bits no son afectados, ya que dependen de la lógica del dispositivo, mientras los Bits T0 y PD no son escribibles. Se sobreentiende entonces que los resultados de una instrucción que tiene como destino el registro STATUS podrán no ser como se espera.

Por ejemplo la instrucción CLRF STATUS limpiará los tres bits altos del registro mientras que el bit Z, se activara, debido a la presencia de un cero en la operación. Los demás bits permanecerán sin cambio.

Solo las instrucciones BCF, BSF SWAPF y MOVWF deberían de usarse para no alterar al mismo tiempo algún otro bit del registro. Más adelante cuando se traten más acerca las instrucciones que emplea el microcontrolador, se verá con más claridad como se afectan los bits DC, y C.

Retomemos la Tabla 1.4.1. para dar una descripción breve de cada bit del registro STATUS.

bit 7-6 No implementados

bit 5 **RP0**, Este bit nos designa el banco de memoria en el que estamos trabajando.

01 = Banco 1 (80h - FFh)

00 = Banco 0 (00h - 7Fh)

bit 4 **TO**: Bit de tiempo fuera

1 = después del encendido, una instrucción CLRWDT o SLEEP.

0 = cuando se acaba el tiempo del WDT. (Temporizador de Perro guardián.)

bit 3 **PD**: Power-down bit o bit de apagado

1 = después del encendido o la instrucción CLRWDT.

0 = cuando se ejecuta la instrucción SLEEP

bit 2 **Z**: Zero bit

1 = cuando el resultado de una operación lógica o aritmética es cero

0 = cuando el resultado de una operación lógica o aritmética es diferente de cero

bit 1 **DC**: Bit de digito de acarreo o préstamo. (Instrucciones ADDWF, ADDLW,SUBLW,SUBWF)

1 = una operación a causado que el bit menos significativo haya salido del registro

0 = no ha salido el bit del registro a causa de la operación.

bit 0 **C**: bit de préstamo o acarreo (Instrucciones ADDWF, ADDLW,SUBLW,SUBWF)

1 = una operación a causado que el bit más significativo haya salido del registro

0 = no ha salido el bit del registro a causa de operación alguna.

El registro OPTION.

Este registro contiene algunos bits de control que se verán a continuación.

bit 7 **RBPU:** bit de activación de los **PORTB Pull-ups**

1 = **PORTB pull-ups** desactivados

0 = **PORTB pull-ups** activados

bit 6 **INTEDG**: bit para activar la interrupción externa en RB0

1 = interrupción por elevación en RB0

0 = interrupción por caída en RB0

bit 5 **T0CS**: Selección del reloj para el TMR0

1 = señal externa en RA4/T0CKI.

0 = reloj interno del microcontrolador.

bit 4 **T0SE**: selección de la fuente para el TMR0

1 = incrementa en transición alta-baja en RA4/T0CKI.

0 = incrementa en transición baja-alta en RA4/T0CKI.

bit 3 **PSA**: asignación del preescalador.

1 = el preescalador se asigna al WDT

0 = el preescaldor se asigna al Timer0.

bit 2-0 **PS2:PS0**: Bits para la selección del rango del preescalador.

Bits	TMR0	WDT
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

El registro INTCON

Este registro contiene los bits de configuración para todas las fuentes de interrupción.

bit 7 **GIE:** Bit para la activación global de interrupciones.
1 = Activa todas las interrupciones.
0 = Desactiva todas las interrupciones.

bit 6 **EEIE**: Bit para activar interrupción si se completa la escritura de la memoria.
1 = Activa la interrupción si se completa la escritura de la memoria
0 = Desactiva la interrupción por escritura de memoria

bit 5 **T0IE**: Activa el bit de interrupción para sobre flujo de TMR0
1 = Activa la interrupción del TMR0
0 = Desactiva la interrupción del TMR0

bit 4 **INTE**: Bit de activación de interrupción por RB0/INT
1 = Activa interrupción externa en RB0/INT
0 = Desactiva la interrupción externa en RB0/INT

bit 3 **RBIE**: Bit para activar la interrupción si hay cambios en el puerto RB
1 = Activa la interrupción si hay cambio en el puerto RB
0 = Desactiva la interrupción por cambio en el puerto RB

bit 2 **T0IF**: Bandera de sobre flujo en el TMR0.
1 = TMR0 ha tenido sobre flujo
0 = TMR0 no ha tenido sobre flujo

bit 1 **INTF**: Bandera de interrupción externa en RB0/INT
1 = Ha ocurrido una interrupción debido a RB0/INT
0 = No ha ocurrido ninguna interrupción por RB0/INT

bit 0 **RBIF**: Bandera de interrupción por cambio en el puerto RB
1 = Al menos uno de los pines RB7:RB4 han cambiado de estado
0 = ninguno de los pines RB7:RB4 han cambiado de estado.

Internamente el microcontrolador posee un contador de ciclo de programa, que indica que instrucción esta siendo ejecutada, a este contador se le llamará **Program Cicle** (PC) y es un registro de 13 bits, los ocho menos significativos se denominan PCL y los 5 más significativos PCH. Solamente PCL es directamente escribible. Por otra parte disponemos de un "apilado" o "stack" de instrucciones que permite llamar, con una profundidad de ruta de hasta 8 subrutinas o interrupciones. De esta forma es como trabaja conjuntamente con el PC: cuando se llama una subrutina u ocurre una interrupción, el programa se mueve hacia la nueva dirección que le asigna el cursor, pero el PC original no es modificado, de esta forma cuando termina la subrutina, el programa regresa a la posición original.

Si intentamos una profundidad de ruta mayor a 8 niveles, el noveno nivel se sobrescribirá al primero, y así sucesivamente, lo cual podría causar que en determinado momento no logre regresar al punto deseado.

Existe también un cursor que se denomina FSR y un numero de registro para cada ubicación de memoria denominado INDF, si nosotros cargamos un valor en el cursor (FSR) por ejemplo 02, y la dirección 02 contiene el valor FFh, entonces el valor que se cargará en el registro INDF será FFh. Estos registros no son útiles para el direccionamiento indirecto de valores en las memorias.

Puertos de Entrada / salida de datos (RA y RB) y registros TRISA y TRISB.

Estos son registros de vital importancia cuando se trata de interactuar con el exterior. Los registros PORTA y PORTB en el banco 0 nos indican la condición de los pines de salida, cada uno consta de 8 bits, pero en el caso del PORTA no importa que pase con los 3 pines más significativos <5:8>, ya que no tienen uso. Si el bit se lee como 1, quiere decir que el pin tiene un estado alto, en cambio si se lee como 0 indicará un estado bajo.

Mientras tanto, los registros TRISA y TRISB determinan si los pines del puerto son salidas o entradas; si el bit correspondiente se lee como 0, indicará que el pin es empleado como salida, y si se lee como 1 indicará que el pin es empleado como entrada.

Todos estos registros son escribibles y leíbles directamente, pero por ejemplo si se trata de escribir un registro como PORTA y contiene entradas en su respectiva configuración, los bits que corresponden a dichas entradas siempre estarán regidos por la lógica del dispositivo.

Para inicializar los puertos es necesario activar el bit RP0 del registro STATUS, con esto pasamos al banco de memoria 1 que contiene a TRISA y TRISB, procedemos a cargar un valor en nuestro acumulador W y lo descargamos en el registro que queramos. El valor puede estar en decimal, hexadecimal o binario. Para valores decimales se escribirán precedidos de la letra d, y entre comillas. Para valores binarios del mismo modo pero precedidos de una b, y los valores hexadecimales se escribirán con el siguiente formato: 0xhh, donde hh será el numero hexadecimal de dos cifras.

Posterior a la inicialización de cualquiera de los dos puertos, debemos regresar al banco 0 de memoria, limpiando el bit RP0 del registro STATUS, ya que es en el banco 0 donde realizamos la mayor parte de operaciones.

Una alternativa es la instrucción TRIS, como se verá más adelante, y que de hecho fue la que se uso en este proyecto.

También hay que considerar que algunos pines como el RA4 están multiplexados con otros periféricos, en este caso el RA4 puede ser utilizado como entrada externa de señal de reloj, si así se configura.

En el puerto B o PORTB existe algo que llamamos **Pull-ups**, se pueden activar desde el bit7 del registro OPTION, y nos sirven para utilizar los pines

como entradas en circuito abierto. Es decir, cuando nosotros configuramos una entrada en el PIC16F84, esta debe tener un estado alto o bajo, no puede estar "suelta", su estado debe ser 0 volts conectado a tierra lógica ó 5 volts, tensión de Vss. Para ayudar con este detalle se implementaron estos Pull-ups, que en realidad mantienen un estado alto en el pin, aunque este no este conectado a voltaje. Esta característica no esta disponible en el puerto A o PORTA. Y por eso se debe de escoger que entradas serán designadas como entradas y cuales como salidas, y en base a eso diseñar el circuito donde trabajará el microcontrolador.

Por otra parte, cuatro pines en el puerto B, RB7:RB4 tienen una característica de cuasar interrupción si existe cambio de estado, esto solamente en aquellos pins que estén configurados como entradas, ya que los que tengan configuración de salida serán excluidos de esta característica por obvias razones. El proceso se lleva a cabo de la siguiente manera: cuando esta activada la función de interrupción por cambio en puerto B, el CPU comparará los valores del puerto B nuevos con los antes leídos a través de una instrucción OR, si existe algún cambio se activara el bit bandera INCON,0.

El objetivo de esta como otras interrupciones es poder tener el PIC trabajando en una rutina o "despertarlo" de un modo de ahorro de energía (SLEEP), para que comience a ejecutar una subrutina. Para regresar al estado de no interrupción por cambio de estado, podemos leer o escribir en el puerto B, registro PORTB y después simplemente limpiar el bit INTCON,0.

Si no leemos o escribimos el registro PORTB se seguirá activando el registro INTCON,0.

TIMER0

El TIMER0 puede funcionar como un temporizador o como un contador, y tiene las siguientes características.

- temporizador / contador de 8 bits
- es leíble y escribible
- puede seleccionar reloj interno o externo
- cuenta con un preescalador programable de 8 bits
- activa interrupción cuando sobre fluye de FFh a 00h

este funcionará en modo de Temporizador cuando el bit 5 del registro OPTION_REG este en 0. En este modo el modulo del Timer0 se incrementará con cada ciclo de instrucción. Si escribimos directamente sobre el registro TIMER0 el incremento se inhibirá los dos siguientes ciclos de instrucción. Así que en aplicaciones de lata velocidad es algo que se debe tener en cuenta para cargar un valor que se ajuste en el TIMER0, en nuestro caso muy concreto, la velocidad de los eventos externos no es afectada por ajustes tan finos del TIMER0.

Por otra parte, si activamos el bit 5 del registro OPTION_REG entraremos en el modo de contador, en este modo de funcionamiento el modulo incrementara en cada cambio de estado del pin RA4/T0CKI, pero también hay que considerar que cuando se utiliza una entrada de reloj externa, se deben de cumplir ciertos requerimientos. Se debe sincronizar la fuente de oscilación externa con la fase de reloj interna TOSC, lo cual puede causar un pequeño retardo.

Para nuestro proyecto utilizaremos solamente el modulo como temporizador, ya que los pines del puerto A tendrán otro uso.

El preescalador

El preescalador es un registro de 8 bits el cual puede ser utilizado por el Temporizador o por el "Temporizador de Perro Guardián". Este registro es no escribible ni leíble, ya que por ser un temporizador no debe ser afectado mas que por el ciclo de reloj.

El bit PSA del registro OPTION_REG sirve para determinar si el preescalador se asignará al TIMER0 o al WDT. Por defecto es 0 y esto asigna el preescalador al TIMER0.

Los bits PS2:PS0 nos dan el valor del preescalador como se vio en la sección del registro OPTION.

La asignación del preescalador esta totalmente controlada por software, y en determinado caso puede ser cambiada en "el vuelo", es decir, durante la ejecución de un programa. Una vez que el módulo del TIMER0 sobre fluye genera una interrupción en el registro INTCON, bit 2, el cual debe ser limpiado en el mismo software antes de su reutilización.

La interrupción que genera el Timer0 no puede regresar al Microcontrolador de un estado de ahorro de energía (SLEEP), ya que el Temporizador es apagado durante el ahorro de energía.

Características especiales del CPU

Lo que diferencia a un microcontrolador de otros procesadores son los circuitos especiales que se han implementado para las necesidades de aplicaciones en tiempo real. El PIC16F8A dispone de varias características previstas para maximizar confiabilidad del sistema, para reducir al mínimo coste con la eliminación de componentes externos, para proporcionar modos de funcionamiento del ahorro de energía y para ofrecer la protección del código. Estas características son:

- Selección del oscilador.
- Reset
  - Reset por encendido (POR)

- Contador de tiempo de ciclo inicial (PWRT)
- Contador de tiempo de inicio del Oscilador (OST)

- Interrupciones
- Contador de tiempo del perro guardián (WDT)
- Ahorro de energía
- Protección del Código
- Localizaciones de las ID
- Programación Serial En-Circuito (ICSP)

El PIC16F8A tiene un contador de tiempo del "perro guardián" que pueda ser desactivado solamente a través de los bits de la configuración inicial. Tiene integrado su propio oscilador de RC para mayor confiabilidad. Hay dos contadores de tiempo que ofrecen el retraso necesario para el ciclo inicial. Uno es el contador de tiempo del inicio del oscilador (OST), previsto para mantener el dispositivo en un estado de RESET mientras que la fuente de Oscilación se estabiliza. El otro es el contador de estabilización de energía que mantiene al controlador en RESET por 72 ms[10] con el objetivo de que la fuente de alimentación se estabilice. Con la adición de esos dos temporizadores no es necesaria la adición de una circuitería para el arranque del dispositivo.

El modo SLEEP ofrece un muy bajo consumo de energía. Podemos "despertarlo" a través de un Reset, por la interrupción de un temporizador de perro guardián o por una interrupción externa.

Existen varias opciones de oscilador, para permitir que la pieza se ajuste a nuestras necesidades. La opción del oscilador de RC[11] ahorra costo mientras que la opción de cristal LP[12] ahorra energía. Un sistema de bits de configuración se utiliza para seleccionar estas opciones.

[10] Milisegundos, o milésimos de segundo.
[11] Resistencia - Capacitor
[12] Low Power, bajo consumo de energía

Los bits de configuración se pueden programar o dejar sin cambios. Estos bits se encuentran en la dirección 2007h de la posición de memoria del programa y están más allá de la memoria del programa de usuario y pertenece a la memoria especial de test / configuration (2000h - 3FFFh). Este espacio se puede alcanzar solamente durante la programación.

Para configurar estos bits cuando se programa la memoria del PIC, se escribe con palabras la configuración deseada, o se puede recurrir a una de las plantillas que proporciona Microchip® con su software MPLAB®, dependiendo de la configuración y controlador que trabajemos.

Configuración de los Osciladores

Tipos de osciladores

El PIC16F84A puede trabajar con cuatro diferentes tipos de osciladores, y se describirán a continuación.

- Cristal de bajo consumo LP
- Cristal / resonador XT
- Cristal / resonador de alta velocidad HS[13]
- Resistor Capacitor RC

Cristal Oscilador / resonador

En modos LP, XT y HS se puede conectar un cristal oscilador en los pines OSC1 y OSC2 para que efectúen el ciclo de oscilación. En éste proyecto se emplea un cristal de cuarzo de 4 MHz, debido a que los PICs adquiridos son para una velocidad máxima de 4 MHz precisamente, aunque como se verá más adelante es posible adquirir PICs que trabajan con un cristal HS a 20 MHz.

En la figura tal se puede apreciar como es la conexión del cristal y su arquitectura interna. La resistencia Rs(2) es opcional y solo se emplea con ciertos tipos de cristales de corte transversal.

[13] High Speed, u oscilador de alta velocidad

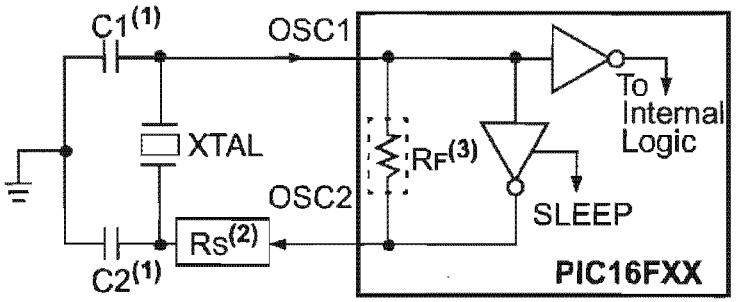

Figura 1.15, diagrama de conexión del cristal oscilador

El diseño del oscilador requiere el uso de un cristal de corte paralelo, ya que existen cristales de corte serie que podrían salir del rango de frecuencia especificado

Para seleccionar el capacitor cerámico adecuado recurriremos a la siguiente tabla.

Modo	Frecuencia	Osc1	Osc2
XT	455 kHz	47 – 100 pF	47 – 100 pF
	2.0 MHz	15 – 33 pF	15 – 33 pF
	4.0 MHz	15 – 33 pF	15 – 33 pF
HS	8.0 MHz	15 – 33 pF	15 – 33 pF
	10.0 MHz	15 – 33 pF	15 – 33 pF

Tabla 1.3. Selección de Cristal Oscilador

Lista de instrucciones para el PIC16F84A

A continuación se listan las instrucciones que emplea el microcontrolador, a partir de las cuales debemos desarrollar el algoritmo de control para nuestro manipulador. También se dará una breve explicación de algunas reglas que se deben seguir cuando escribimos el programa en lenguaje ensamblador.
Se listarán en orden alfabético y conforme las podemos encontrar en el sitio de MICRCHIP®.

Se pueden leer o escribir cualquier registro, incluyendo los registros de funciones especiales, pero se debe tener cuidado porque son estos los que rigen el funcionamiento del controlador.

Las instrucciones orientadas a bits leerán primero todo el registro, posteriormente operarán sobre el bit y al final escribirán el valor en el puerto, se debe tener en cuenta esto cuando se trabaja con registro especiales como los puertos.

W es el acumulador del microcontrolador, todos los valores de las operaciones pasan por dicho acumulador.

Es muy conveniente interpretar las literales empleadas e incluso las instrucciones y registro por su nombre en inglés. Resulta fácil si consideramos que por ejemplo el registro STATUS se refiere al estado del controlador. Asimismo, en las siguientes instrucciones se emplea 'd' por su significado destination = destino, que nos indica donde se guardarán los datos de la operación, 'f' por file = registro, archivo, que nos especifica una locación de memoria. W que es nuestro Work_Register = registro de trabajo, por el cual pasarán la mayoría de los valores.

La etiqueta es opcional y solamente la escribimos donde queremos tener una marca para llamar una subrutina o ejecutar un salto.

Básicamente el ensamblador, MPLAB, reconoce cuatro columnas de texto, la primera se referirá siempre a las etiquetas, la segunda contendrá la instrucción, la tercera será el valor de la literal o el nombre o locación del registro a usar, y la cuarta, precedida de una comilla, puede ser utilizada como comentario dentro del programa.

1.5 Etapas de potencia para el control de motor de pasos.

Como se vio anteriormente, el PIC16F84A será nuestro controlador el cual se encargará de generar el tren de pulsos digitales para el control de la rotación, velocidad y aceleración, pero sus salidas, de apenas 25mA, son incapaces de manejar consumos mayores a los de un LED, por lo tanto se debe de acoplar una etapa de potencia adecuada que nos permita manejar las diversas corrientes y voltajes que requieren los motores paso a paso, las cuales varían desde 300mA hasta 2 A.

Se seleccionará un transistor de potencia tipo Darlington que pueda manejar la cantidad de corriente requerida y tenga un margen de tolerancia bastante amplio para evitar el sobrecalentamiento con el uso continuo. El circuito lineal seleccionado para este proyecto fue el TIP122, lo se puede apreciar en la figura 1.5.1.

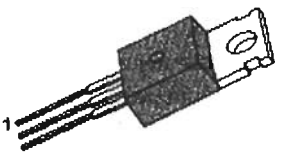

Figura 1.16, configuración del TIP122, 1.Base, 2.Colector, 3.Emisor.

Es recomendable acoplar un disipador de calor, para aplicaciones en las que las exigencias del sistema pueden llevar a sus límites de funcionamiento este transistor, o simplemente para tener un mayor margen de seguridad, pero de debe tener en cuenta que el disipador y el colector son comunes, por lo que no deberá emplearse el mismo elemento disipador de calor para más de uno de ellos, y si se hace, deberá de contar con el aislamiento adecuado. Estos por lo regular se aíslan con micas.

En la tabla 1.5. se pueden ver sus características eléctricas principales:

Tabla 1.5. Características principales del TIP122

Tipo	NPN	Transistor Darlington
Voltaje máximo Base-Colector	V_{CBO}	100 Volts
Voltaje máximo Emisor-Colector	V_{CEO}	100 Volts
Voltaje máximo Emisor-Base	V_{EBO}	5 Volts
Voltaje de saturación Base-Colector	$V_{CE(Sat)}$	2 Volts
Corriente en colector (CD)	I_C	5 A
Corriente en colector (Pulsos)	I_c	8 A
Corriente en la Base	I_B	120 mA
Corriente de Saturación de la Base	$I_{B(Sat)}$	12 mA
Disipación del Colector (Tc = 25°C)	Pc	65 W
Temperatura máxima de la junta	T_j	150 °C

Como se puede observar la corriente máxima que puede circular en un momento de retención, circulación continua, puede llegar a ser de hasta 5 amperes, por lo que si se emplea un motor que consuma 2 amperes a 6 Volts la corriente estará utilizando apenas el 40% de la capacidad del TIP122. Nuestra potencia disipada será de 12 VA, aproximadamente 10.32 Watts. Este transistor nos deja un amplio margen de seguridad para evitar que el semiconductor se colapse.

En la figura 1.17 se puede apreciar la gráfica de potencia disipada contra temperatura característica de nuestro TIP122.

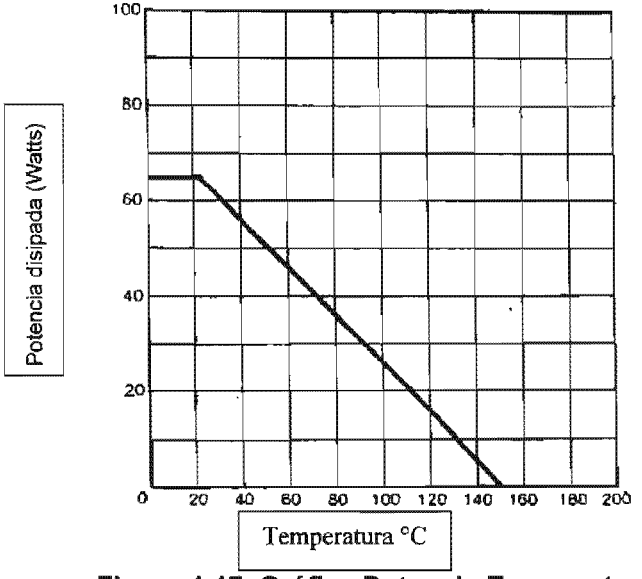

Figura 1.17. Gráfica Potencia-Temperatura

Se necesita conocer adicionalmente el circuito equivalente del TIP122 para poder determinar la manera en que será conectado el microcontrolador, hay que recordar que la corriente máxima de salida de los puertos del PIC es de 25 mA, y la salida de 5 Volts. Por lo cual se recurrirá a la ley de Ohm para determinar la resistencia mínima necesaria en el circuito de baja corriente.

De la siguiente manera, haciendo el despeje de la resistencia nos queda:

$$R = V / I$$

Tomando los valores de 5 Volts y .025 A, se obtiene una resistencia necesaria de 200 Ohms.

En la figura 1.18 se puede apreciar el circuito equivalente del TIP122.

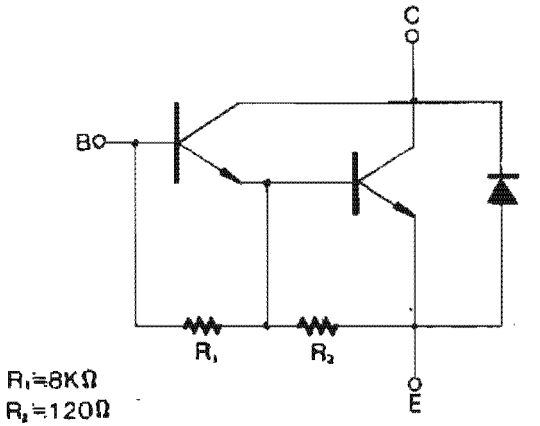

Figura 1.18. Circuito equivalente del TIP122

La resistencia interna que nos puede ofrecer el TIP122 entre su Base y emisor es de apenas 120 Ohms, por lo tanto será necesario agregar otra resistencia la circuito. Una resistencia de 100 Ohms a ¼ Watt será más que suficiente. La resistencia tampoco debe de ser demasiado grande, ya que la corriente de saturación de la Base es de 12 mA a 2 Volts, si se toma una resistencia mayor que disminuya ésta corriente por debajo del limite, se tendrán problemas de activación en el transistor.

En el circuito equivalente se puede apreciar que existe también un diodo, el cual permitirá el desahogo de la corriente residual cuando el TIP detiene su funcionamiento, pero adicionalmente se agregará otro diodo externo como protección. Uno de la familia 1N400x será adecuado.

Habiéndose hecho la selección y tomando en consideración los datos de nuestro controlador, transistor de potencia y motor de pasos, resulta sencillo diseñar el acoplamiento para el circuito de control.

Capitulo 2

Construcción del Perfil de Posición, Velocidad y Aceleración de un Robot de Cuatro Grados de Libertad

2.1 Perfil de posición

Para definir el perfil de posición del manipulador, se recurrirá a las ecuaciones previamente obtenidas en el capítulo 1, ya sea por medio del álgebra de cuaterniones, basándonos en los parámetros que podemos apreciar en la figura 2.1 , o por medio de la teoría de mecanismos, como se verá más adelante. Por medio de cuaterniones, se tendrá como resultado:

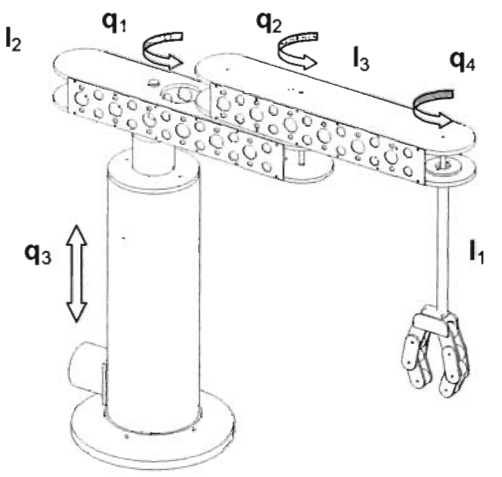

Figura 2.1, Parámetros del Robot SCARA

$$(0,\mathbf{a}_0) = (0, l_3C_{1122} + l_2C_{11}, l_3S_{1122} + l_2S_{11}, q_3 - l_1) \qquad [2.1]$$

$$\mathbf{R}_0 = (C_{124}, 0, 0, S_{124}) \qquad [2.2]$$

O mejor planteado en términos geométricos, se expresa de la siguiente manera:

$$X = a_{0x} = l_3 \cos(q_1+q_2) + l_2 \cos q_1 \qquad [2.3]$$
$$Y = a_{0y} = l_3 \operatorname{sen}(q_1+q_2) + l_2 \operatorname{sen} q_1$$
$$Z = a_{0z} = q_3 - l_1$$

Y la orientación del efector queda definida como

$$\mathbf{Rot}\,(\mathbf{z}, q_1+q_2+q_4) \quad \text{ó} \quad q_1 + q_2 + q_4 \text{ en torno al eje } \mathbf{z} \qquad [2.4]$$

En este caso nos interesa, conocer la orientación a través de la teoría de mecanismos, pues será con esta con la cual elaboraremos los perfiles de velocidad y aceleración.

Se comenzará con las ecuaciones para los eslabones principales. Primeramente, la ecuación vectorial de cierre, cuyos parámetros podemos apreciar en la figura 2.2:

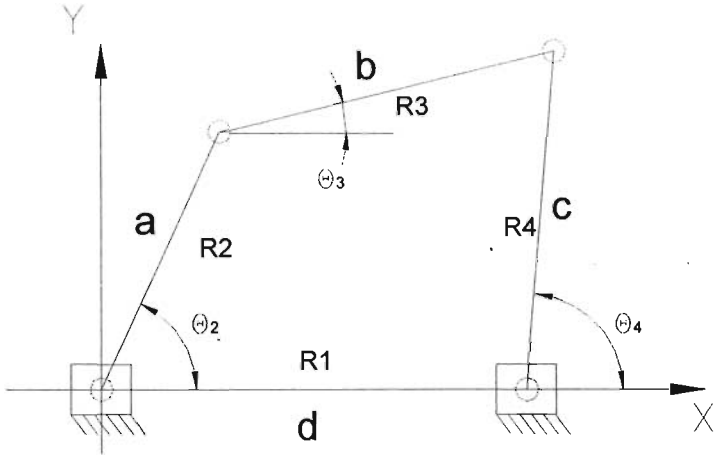

Figura 2.2. Mecanismo de 4 barras y parámetros correspondientes

$$R2 + R3 - R4 - R1 = 0 \qquad [1.31]$$

Que como se dedujo anteriormente, se convierte en

$$a(\cos\theta_2 + j\text{sen}\theta_2) + b(\cos\theta_3 + j\text{sen}\theta_3) - c(\cos\theta_4 + j\text{sen}\theta_4) - d(\cos\theta_1 + j\text{sen}\theta_1) = 0 \qquad [1.34]$$

Esta ecuación claro, solo resuelve el movimiento en el plano XY, debemos agregar, un vector adicional, que nos dará la elevación en Z, pudiéndolo llamar R5 que sería nuestra componente en el plano Z. O de otra manera, adicionar directamente a la ecuación el valor de **Z** en el eje *k*.

$$a(\cos\theta_2 + j\text{sen}\theta_2) + b(\cos\theta_3 + j\text{sen}\theta_3) - c(\cos\theta_4 + j\text{sen}\theta_4) - d(\cos\theta_1 + j\text{sen}\theta_1) + \mathbf{Z}k = 0 \qquad [2.1]$$

En la figura 2.3 se puede apreciar la adición del vector R5 a las proyecciones de los vectores R2 y R3 para completar el espacio en los tres ejes coordenados.

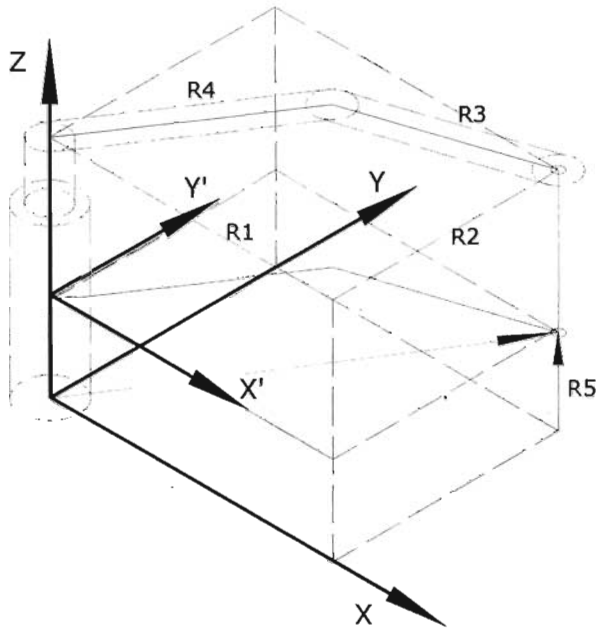

Figura 2.3 Complemento del sistema coordenado con el vector R5

Con lo anterior, se deduce que se pueden trabajar por separado los parámetros que se proyectan en el plano X'Y', el cual se tomará simplemente como XY en las ecuaciones para mayor comodidad, y adicionar por separado los correspondientes al eje Z.

Así pues, resolviendo la ecuación para obtener los ángulos de cada eslabón con respecto al origen del sistema coordenado X'Y', y tomando la cadena cinematica en un orden más conveniente, nos queda para θ_4:

$$A \tan^2(\theta_4 / 2) + B \tan(\theta_4 / 2) + C = 0 \qquad [1.44]$$

Donde: $A = \cos\theta_2 - K_1 - K_2 \cos\theta_2 + K_3$

$B = -2 \operatorname{sen}\theta_2$

$C = K_1 - (K_2 + 1) \cos\theta_2 + K_3$

Y para θ_3

$$D \tan^2(\theta_3 / 2) + E \tan(\theta_3 / 2) + F = 0 \qquad [1.48]$$

Donde: $D = \cos\theta_2 - K_1 - K_4 \cos\theta_2 + K_5$

$E = -2 \operatorname{sen}\theta_2$

$F = K_1 - (K_4 - 1) \cos\theta_2 + K_5$

De las cuales obtendremos los ángulos para cada eslabón y la ubicación en el plano XY estará dada por:

$$X = l_3 \cos(\theta_2 + \theta_3) + l_2 \cos\theta_2$$
$$Y = l_3 \operatorname{sen}(\theta_2 + \theta_3) + l_2 \operatorname{sen}\theta_2 \qquad [2.2]$$

Conservando como l_2 y l_3 las longitudes de los eslabones, conocidas en la teoría de mecanismos como *a* y *b*. Podemos notar que la respuesta a la que se llega para el perfil de posición con cuaterniones y teoría de mecanismos es la misma. El valor correspondiente a Z quedará definido de igual manera como:

$$Z = \mathbf{R5} = q3 - l_1 \qquad [2.3]$$

Y la orientación del efector final quedará definida como:

$$\theta_4 + \theta_3 + \theta_{Ef} \text{ en torno al eje } \mathbf{z} \qquad [2.4]$$

En este caso θ_4 y θ_3 equivalen a $\mathbf{q}_1$ y $\mathbf{q}_2$ mientras θ_{Ef} equivale a $\mathbf{q}_4$ y comprobamos la equivalencia con el método de cuaternios.

2.2 Perfil de velocidad

Una vez comprobada la validez de este método para la resolución cinemática inversa de este modelo, procederemos a implementar la teoría de velocidad y aceleración que se obtuvo con la teoría de mecanismos.

Se tomará como base la ecuación 1.2.29 deducida en el capitulo 1, la cual expresa lo siguiente:

$$ja\omega_2 e^{j\theta 2} - jb\omega_3 e^{j\theta 3} - \mathbf{c} - \mathbf{d} = 0 \qquad [1.57]$$

que al introducir los equivalentes de Euler, queda de la siguiente forma

$$a\omega_2 (-\text{sen}\theta_2 + j\cos\theta_2) - b\omega_3 (-\text{sen}\theta_3 + j\cos\theta_3) - \mathbf{c} - \mathbf{d} = 0 \qquad [1.58]$$

Pero se debe recordar el agregar la velocidad en el eje Z, para lo cual podemos hacer la adición en la anterior ecuación de la siguiente manera:

$$a\omega_2 (-\text{sen}\theta_2 + j\cos\theta_2) - b\omega_3 (-\text{sen}\theta_3 + j\cos\theta_3) + \text{Vel}_z - \mathbf{c} - \mathbf{d} + \mathbf{Z}k = 0 \qquad [2.5]$$

Es notable, que si las ecuaciones obtenidas en la teoría de mecanismos resuelven los dos eslabones principales rotativos que se encuentran en el plano XY, eslabones 2 y 3, y el primer grado de libertad es únicamente para la variación en el eje Z, se puede ir agregando este termino en cada ecuación sin mayor problema, puesto que las ecuaciones de cierre, velocidad y aceleración de la teoría de mecanismos darán como resultado siempre un vector con componentes X e Y, al cual solo se tendrá que agregar el último vector con su componente única en Z, haciendo que el problema sea una simple suma vectorial en tres dimensiones para la obtención del modelo dinámico completo, esto claro considerando el efector final como un punto en el espacio.

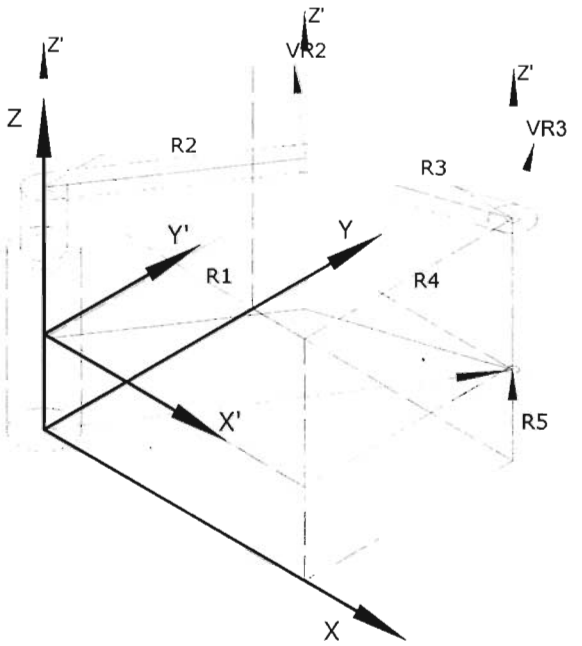

Figura 2.4, Velocidad de los eslabones

En la figura 2.4 puede apreciarse la colocación de los vectores de velocidad en los eslabones principales.

En base a lo anterior podemos hacer la siguiente consideración, cambiando los términos de parte real e imaginaria por componentes en X,Y,Z:

Componente X $-a\omega_2 \operatorname{sen}\theta_2 + b\omega_3 \operatorname{sen}\theta_3 - \mathbf{d} = 0$ [2.6]

Componente Y $a\omega_2 \cos\theta_2 - b\omega_3 \cos\theta_3 - \mathbf{c} = 0$ [2.7]

Componente Z $Vel_z - \mathbf{Z'} = 0$ [2.8]

La componente en Z, en comparación con las componentes en X e Y, es extremadamente sencilla, como se menciono antes, por ser afectada únicamente por uno de los grados de libertad. Además en la ecuación podemos notar que la

velocidad en el eje Z será igual a la que se imprima en el actuador de dicho grado de libertad.

De igual manera, las aceleraciones locales de los punto A y B, serían

$V_A = a\omega 2\ (-i\ sen\theta 2 + j\ cos\theta 2) + k\ \mathbf{Z'}$

$V_{AB} = b\omega 3\ (-I\ sen\theta 3 + j\ cos\theta 3) + k\ \mathbf{Z'}$

$V_{BA} = -V_{AB}$ [2.9]

En la figura 2.5 se puede apreciar como quedan los vectores de velocidad de cada una de las juntas rotativas.

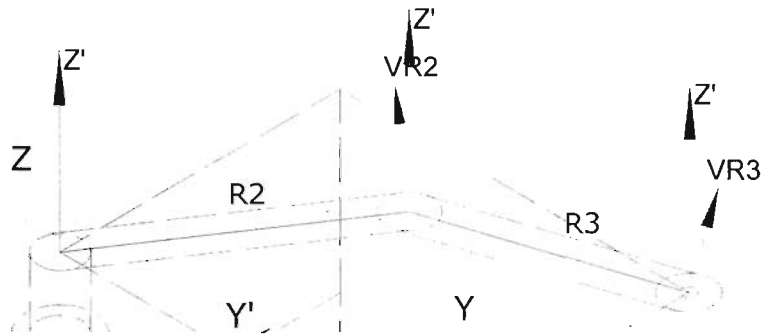

Figura 2.5. Suma vectorial de velocidades en juntas rotativas

2.3 Perfil de Aceleración

En la figura 2.6 se pueden apreciar las aceleraciones contenidas en plano que forman los eslabones rotativos.

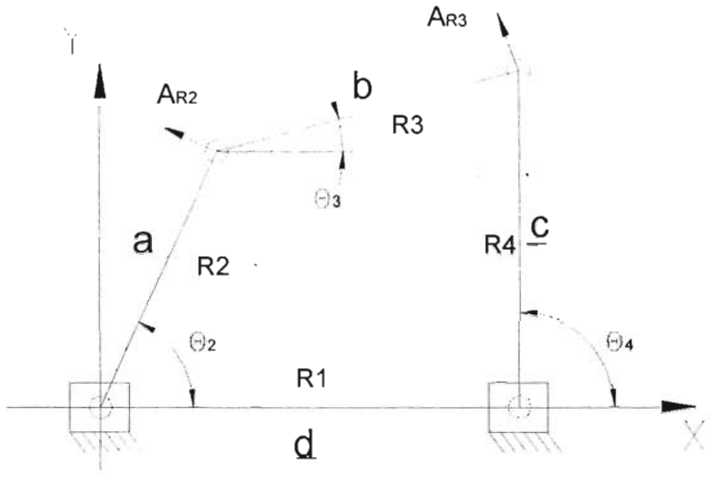

Figura 2.6. Aceleraciones de los eslabones principales

Nuevamente, solo bastara con tomar el perfil de velocidad y derivarlo, para obtener el perfil de aceleración:

$$(ja\alpha_2 e^{j\theta 2} + j^2 a\omega_2^2 e^{j\theta 2}) - (jb\alpha_3 e^{j\theta 3} + j^2 b\omega_3^2 e^{j\theta 3}) + Acel_z - \underline{c} - \underline{d} - \mathbf{Z''} = 0 \qquad [2.10]$$

esta vez **d** será la velocidad en el eje X, **c** en el eje Y, y **Z''** en el eje Z precisamente.

Se puede apreciar que nuestro termino de velocidad, se cambio por el de aceleración $Acel_z$, mientras que a nuestro termino **Z'** se opto por únicamente convertirlo a **Z''**, para tener una mejor comprensión de la operación realizada.

Nuevamente, al introducir lo equivalentes de Euler, tendremos:

$$a\alpha_2 (-\text{sen}\theta_2 + j\cos\theta_2) - a\omega_2^2(\cos\theta_2 + j\,\text{sen}\theta_2)$$
$$- b\alpha_3(-\text{sen}\theta_3 + j\cos\theta_3) + b\omega_3^2(\cos\theta_3 + j\,\text{sen}\theta_3) + \text{Acel}_z - \underline{\mathbf{c}} - \underline{\mathbf{d}} - \mathbf{Z''} = 0 \qquad [2.11]$$

Separando esta vez en components X, Y, y Z

Componente X $\quad -a\alpha_2\,\text{sen}\theta_2 - a\omega_2^2\cos\theta_2 + b\alpha_3\,\text{sen}\theta_3 + b\omega_3^2\cos\theta_3 - \underline{\mathbf{d}} = 0$

Componente Y $\quad a\alpha_2\cos\theta_2 - a\omega_2^2\,\text{sen}\theta_2 - b\alpha_3\cos\theta_3 + b\omega_3^2\,\text{sen}\theta_3 - \underline{\mathbf{c}} = 0$

Componente Z $\quad \text{Acel}_z - \mathbf{Z''} = 0 \qquad [2.12]$

Para obtener las aceleraciones que se desee conocer, bastará con sumar vectorialmente cada una de las componentes para conocer el vector de aceleración correspondiente.

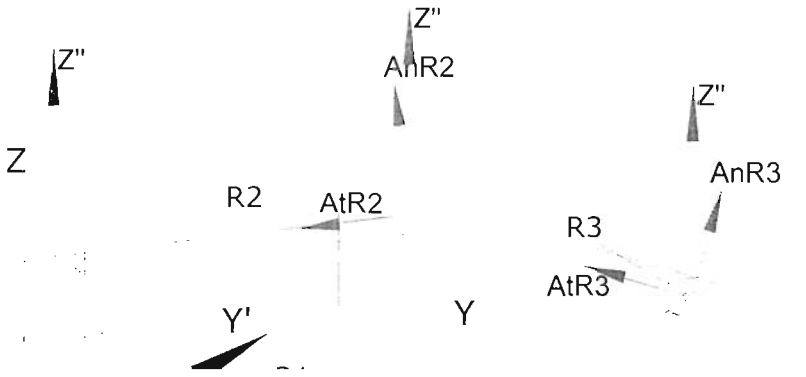

Figura 2.7. Aceleraciones en las juntas del Robot Scara

En la figura 2.7 puede apreciarse como quedan los vectores de aceleración de cada una de las juntas del manipulador. Se puede apreciar que en las juntas rotativas existen tres aceleraciones, que son, la aceleración Normal, la aceleración Tangencial y la aceleración de ascenso Z'', la cual es la única que se presenta en la junta rotativa primera, es decir, en el par inferior del eslabón 2, y también a lo largo de toda la junta prismática.

Capitulo 3

Software

3.1 Programación Básica en Visual Basic®

3.1.1 Introducción

Visual Basic® es uno de los lenguajes de programación con mayores ventajas para los programadores de PCs. En el caso de los programadores expertos por la facilidad con la que desarrollan aplicaciones de gran complejidad en comparación con otros lenguajes de programación, como Visual C++®, por ejemplo. El inconveniente por utilizar Visual Basic® es una menor velocidad o eficiencia en las aplicaciones cuando estas llegan a ser en grado complejas, como seria el caso de un software CAD u hojas de calculo poderosas.

La estructura básica y principal interfaz de este programa la podemos apreciar en la figura 3.1.

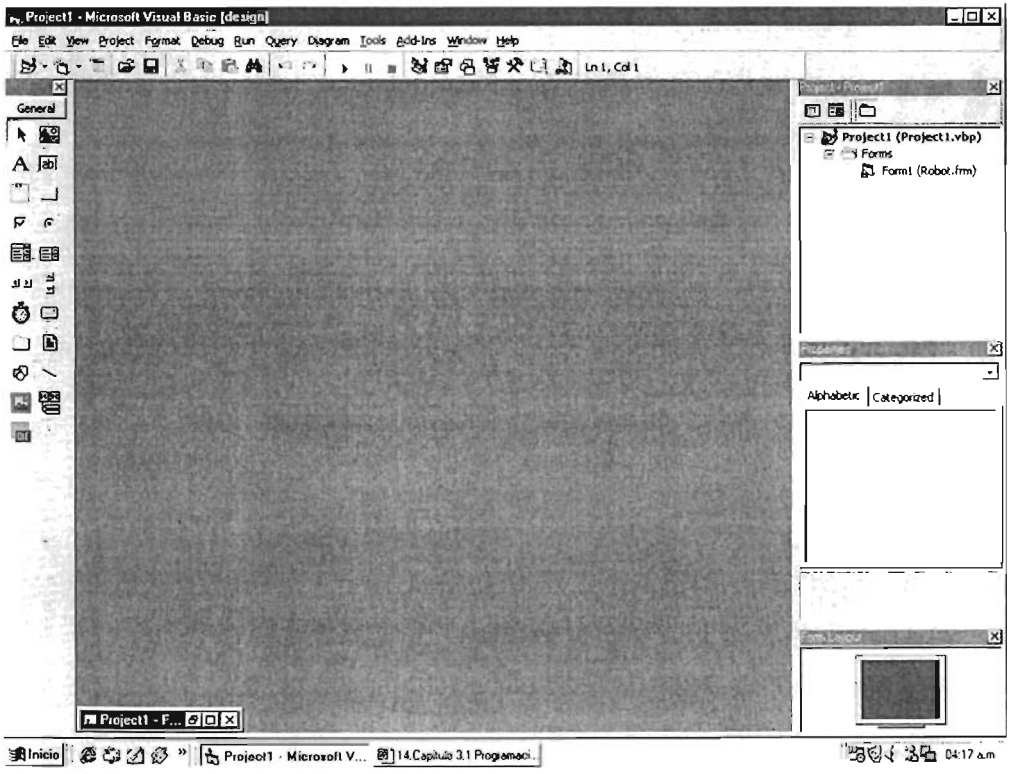

Figura 3.1. Interfaz principal de Visual Basic 6.07

Visual Basic® es un lenguaje de programación visual, también llamado lenguaje de 4ª generación. Esto quiere decir que un gran número de tareas se realizan sin escribir código, simplemente con operaciones gráficas realizadas con el ratón sobre la pantalla. Es también un programa basado en objetos, aunque no orientado a objetos, la diferencia está en que Visual Basic® utiliza objetos con propiedades y métodos, pero carece de los mecanismos propios de los verdaderos lenguajes orientados a objetos como Java y C++. Es decir, Visual Basic® es un lenguaje de programación que emplea objetos sobre los cuales ocurren eventos, tales como un clic y ejecuta una tarea, a esto le llamaremos *Procedimiento de Evento*, para estructurar los códigos, los cuales se escriben en lenguaje Basic.

Hay que destacar que Visual Basic® crea aplicaciones de Windows, con una interfaz gráfica que es la base de la programación, pudiendo integrar todos los elementos de Windows tales como ventanas, botones, cajas de diálogo y de texto, botones de opción y de selección, barras de desplazamiento, gráficos, menús, etc., es decir, prácticamente todos los elementos de que dispone Windows desde Visual Basic®, basta con agregarlos a nuestra interfase gráfica, a la cual llamaremos formulario en lo sucesivo, dar doble clic y escribir el código correspondiente.

La principal razón por la que se escogió este lenguaje de programación es el conocimiento que se tiene de Qbasic y el lenguaje Basic en general, el cual se maneja en las calculadoras programables actuales, por lo que solamente se tiene que familiarizar el programador que ya conoce Qbasic o Basic con las interfaces gráficas, lo demás resulta muy intuitivo, e incluso para programadores novatos, como se mencionó antes, resulta fácil el aprendizaje.

3.1.2. Tipos de Programas

Existen distintos tipos de programas. En los primeros tiempos de los ordenadores los programas eran de tipo secuencial. Un programa secuencial es un programa que se arranca, lee los datos que necesita, realiza los cálculos e imprime o guarda en el disco los resultados. De ordinario, mientras un programa secuencial está ejecutándose no necesita ninguna intervención del usuario. A este tipo de programas se les llama también *programas basados u orientados a*

procedimientos o *a algoritmos* (*procedural languages*). Este tipo de programas siguen utilizándose ampliamente en la actualidad, pero la difusión de los PCs ha puesto de actualidad otros tipos de programación.

Los programas interactivos exigen la intervención del usuario en tiempo de ejecución, bien para suministrar datos, bien para indicar al programa lo que debe hacer por medio de menús. Los programas interactivos limitan y orientan la acción del usuario. Un ejemplo de programa interactivo podría ser Matlab®.

Por su parte los programas orientados a eventos son los programas típicos de Windows, tales como Netscape, Word, Excel y PowerPoint. Cuando uno de estos programas ha arrancado, lo único que hace es quedarse a la espera de las acciones del usuario, que en este caso son llamadas eventos.

El usuario dice si quiere abrir y modificar un fichero existente, o bien comenzar a crear un archivo desde el principio. Estos programas pasan la mayor parte de su tiempo esperando las acciones del usuario (eventos) y respondiendo a ellas. Las acciones que el usuario puede realizar en un momento determinado son variadas, y exigen un tipo especial de programación: la programación orientada a eventos. Este tipo de programación es sensiblemente más complicada que la secuencial y la iterativa, pero Visual Basic® la hace fácil para el usuario tanto experto como principiante. Este es el tipo de programa que nosotros creamos para el control, simulación y programación de nuestro Robot, y es la razón más importante por la cual también se escogió este lenguaje, en el cual nuestros archivos creados serán, los códigos fuente que deberán ser cargados a los microcontroladores para la ejecución de la secuencia simulada.

3.1.3 Partes de un Programa Creado en Visual Basic®.

Principalmente hablaremos de Formularios, controles, objetos y propiedades. Los formularios (Forms) son las ventanas en las cuales colocaremos los demás objetos, controles, como lo son botones, cajas de texto, menús, etc., a los cuales se les asigna un nombre por defecto para hacer referencia a ellos dentro del código, pero estos nombre pueden ser cambiados, por ejemplo, en vez de llamar a un botón "Command_1", que es como lo designa Visual Basic®, lo podemos llamar "Abrir", "Cerrar", o simplemente "Botón", dependiendo del uso que

nosotros le demos al objeto podemos llamarlo, como en nuestro caso, "coordenadaX". En la figura 3.2 podemos apreciar la estructura básica de nuestro Form o fichero principal, ventanas gráficas, botones, ventanas de texto y barras de desplazamiento.

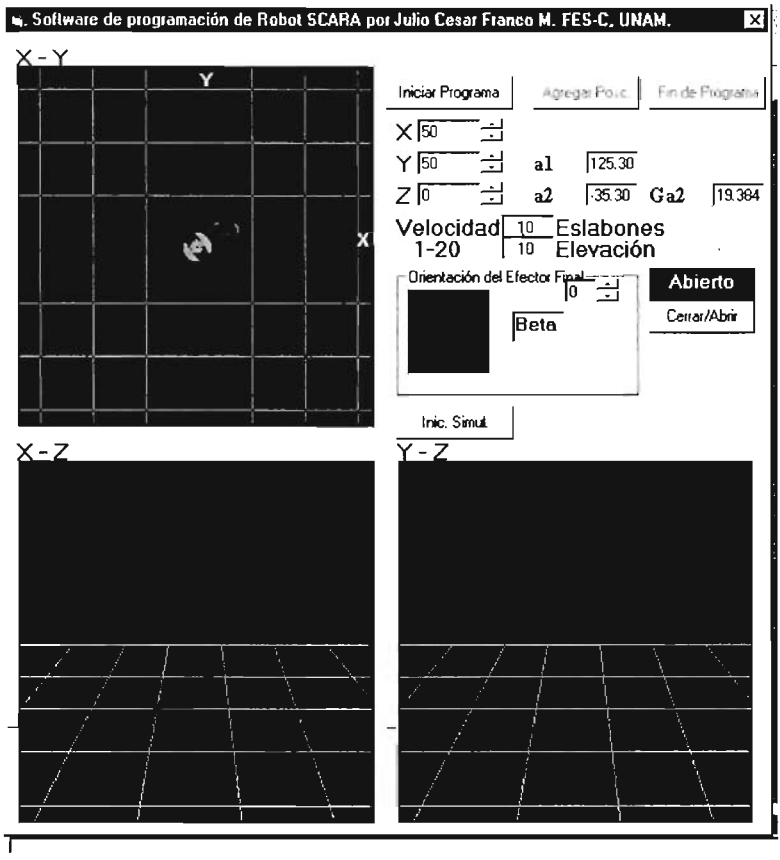

Figura 3.2. Form o fichero básico del Software Scara.

Los objetos se dividen a su vez en clases, de esta forma, todos los botones de comando pertenecen a una clase, mientras que la ventanas gráficas a otra, las barras de desplazamiento a una diferente y así con los subsecuentes objetos.

Cada formulario y cada tipo de control tienen un conjunto de propiedades que definen su aspecto gráfico como el tamaño, color, posición en la ventana, tipo y tamaño de letra, etc., y su forma de responder a las acciones del usuario. Cada propiedad tiene un nombre que viene ya definido por el lenguaje.

Por lo general, las propiedades de un objeto son datos que tienen valores lógicos o numéricos concretos, propios de ese objeto y distintos de las de otros objetos de su clase. Así pues, cada clase, tipo de objeto o control tiene su conjunto de propiedades, y cada objeto o control concreto tiene unos valores determinados para las propiedades de su clase. En la figura 3.3 podemos apreciar la ventana *Properties*.

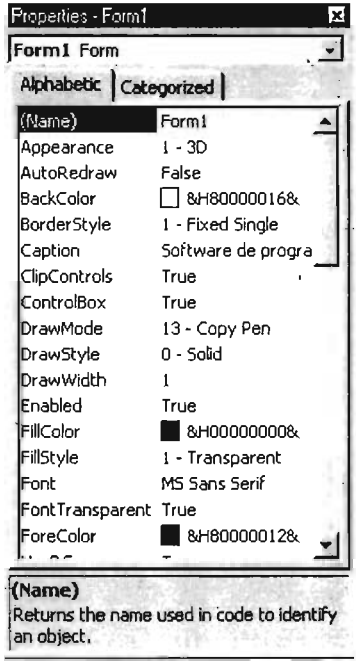

Figura 3.3. Ventana Properties.

Casi todas las propiedades de los objetos pueden establecerse en tiempo de diseño y también, casi siempre, en tiempo de ejecución. Esto quiere decir que es posible cambiar por ejemplo el tamaño, color o texto de un botón mientras se ejecuta el programa. En este segundo caso se accede a sus valores por medio de las sentencias del programa, en forma análoga a como se accede a cualquier variable en un lenguaje de programación. Para ciertas propiedades ésta es la única forma de acceder a ellas. Por supuesto Visual Basic®, al igual que Qbasic, permite crear distintos tipos de variables.

Se puede acceder a una propiedad de un objeto por medio del nombre del objeto a que pertenece, seguido de un punto y el nombre de la propiedad, como por ejemplo Texto1.Text para acceder al texto contenido de Texto1.

3.1.2.1 Eventos

Ya se ha dicho que las acciones del usuario sobre el programa se llaman eventos. Son eventos típicos el dar clic sobre un botón, el hacer doble clic sobre el nombre de un fichero para abrirlo, el arrastrar un icono, el pulsar una tecla o combinación de teclas, el elegir una opción de un menú, el escribir en una caja de texto, o simplemente mover el ratón.

Cada vez que se produce un evento sobre un determinado tipo de control, Visual Basic® ejecuta la acción programada por el usuario para ese evento concreto, si no existe nada programado para tal evento, el programa no ejecutara nada y seguirá en espera. Estos procedimientos se llaman con un nombre que se forma a partir del nombre del objeto y el nombre del evento, separados por el carácter guión bajo (_), como por ejemplo Boton1_click, que es el nombre del procedimiento que se ocupará de responder al evento *click* en el objeto *Boton1*.

3.1.2.2 Métodos

Los métodos son funciones que también son llamadas desde programa, pero a diferencia de los procedimientos no son programadas por el usuario, sino que vienen ya pre-programadas con el lenguaje. Los métodos realizan tareas típicas, comunes para todas las aplicaciones. De ahí que vengan con el lenguaje y que se libere al usuario de la tarea de programarlos. Cada tipo de objeto o de control tiene sus propios métodos, como por ejemplo no hay que programar que cuando demos clic sobre una barra de desplazamiento, ésta cambie y recorra su indicador central, esto es un método que ya viene preprogramado, o el sombrear un texto que escojamos arrastrando el mouse. También, los controles gráficos tienen un método llamado *Line* que se encarga de dibujar líneas rectas. De la misma forma existe un método llamado *Circle* que dibuja circunferencias y arcos de circunferencia Es obvio que el dibujar líneas rectas o circunferencias es una tarea común para todos los programadores y Visual Basic®, de hecho los métodos son tan comunes en los programas actuales y en el entorno Windows que pasan desapercibidos la mayoría de las veces.

3.1.2.3 Proyectos y ficheros

Cada aplicación que se empieza a desarrollar en Visual Basic® es un nuevo proyecto. Un proyecto comprende otras componentes más sencillas, como por ejemplo los formularios más comúnmente conocidas como ventanas, y los módulos que son la parte del programa que contiene el código fuente con las instrucciones a ejecutar. Un proyecto se compone siempre de varios ficheros, en nuestro caso cuatro, aunque dos es lo mínimo, y hay que preocuparse de guardar cada uno de ellos en el directorio adecuado y con el nombre adecuado. En la figura 3.4 se puede apreciar los ficheros que componen el proyecto. Existe siempre un fichero con extensión *.vbp que es el nombre principal del proyecto y se crea con el comando *File/Save Project As*. El fichero del proyecto contiene toda la información de conjunto. Además hay que crear un fichero por cada formulario y por cada módulo que tenga el proyecto. Los ficheros de los formularios se crean con *File/Save Nombredelarchivo As* teniendo como extensión *.frm. Los ficheros de código o módulos se guardan también con el comando *File/Save Nombredelarchivo As* y tienen como extensión *.bas si se trata de un módulo estándar o *.cls si se trata de un módulo de clase (class module). Dando click en el botón *Save* en la barra de herramientas se actualizan todos los ficheros del proyecto. Si no se habían guardado todavía en el disco, Visual Basic® abre cajas de diálogo *Save As* por cada uno de los ficheros que hay que guardar.

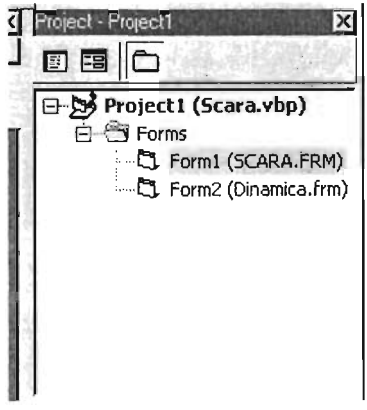

Figura 3.4. Ventana Proyect

3.2 Entorno de programación visual basic®

Visual Basic® es una herramienta de programación para crear aplicaciones que corran sobre los sistemas operativos de Microsoft®, tales como Windows 95/98/ME o Windows NT/XP. Con ella se puede crear desde una simple calculadora hasta una hoja de cálculo de la talla de Exel, procesador de textos como Word o cualquier otra aplicación, lo cual nos permite crear nuestro software propio para nuestro proyecto que correrá sobre Windows®. Sus aplicaciones en Ingeniería son casi ilimitadas: representación de movimientos mecánicos o de funciones matemáticas, gráficas termodinámicas, simulación de circuitos, etc. De hecho es común encontrar en Internet una gran cantidad de aplicaciones desarrolladas en Visual Basic®, aunque las aplicaciones comerciales más potentes siempre son escritas en lenguaje C. Crear el entorno de programa tal como ventanas ó botones es tan sencillo como escoger el objeto y ajustar su tamaño y posición arrastrando el mouse. El lenguaje de programación que se utilizará, como se ha dicho antes, será el Basic, ampliamente conocido por su sencillez de aprendizaje.

Visual Basic® tiene todos los elementos que caracterizan a los programas de Windows® e incluso algunos no tan habituales. En cualquier caso, el entorno de Visual Basic® es muy intuitivo para los familiarizados con Windows®, y además se puede obtener una descripción de la mayoría de los elementos dando clic en ellos para seleccionarlos y pulsando luego la tecla <F1>, si se tiene instalada la ayuda.

En la figura 3.5 se puede apreciar la barra de herramientas y menús de Visual Basic®.

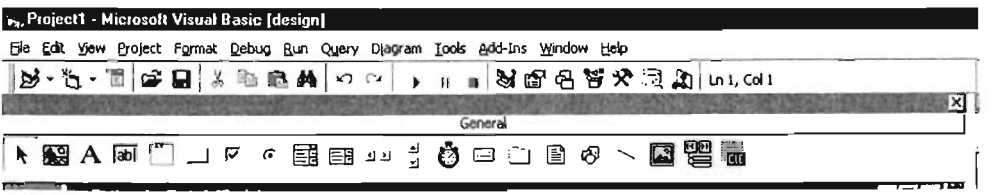

Figura 3.5. Barras de Menú y Herramientas Visual Basic®

3.2.1 La barra de menús y las barras de herramientas

La barra de menús de Visual Basic® resulta similar a la de cualquier otra aplicación de Windows. Bajo dicha barra aparecen las barras de herramientas, con una serie de botones que permiten acceder fácilmente a las opciones más importantes de los menús. En Visual Basic® existen cuatro barras de herramientas: *Debug, Edit, Form Editor y Standard.* Por defecto sólo aparece la barra *Standard.* Dando clic con el botón derecho sobre cualquiera de las barras de herramientas aparece un menú contextual con el que se puede hacer aparecer y ocultar cualquiera de las barras. Al igual que en otras aplicaciones de Windows 95/98/NT como por ejemplo Autocad o Word, también pueden modificarse las barras añadiendo o eliminando botones. En la barra de herramientas Standard también se pueden ver a la derecha dos recuadros con números, que representan cuatro propiedades del formulario referentes a su posición y: Top y Left, que indican la posición de la esquina superior izquierda del formulario, y también Height y Width, que describen el tamaño del mismo en unas unidades llamadas twips, que se corresponden con la vigésima parte de un punto (una pulgada tiene 72 puntos y 1440 twips). Los botones de la barra de herramientas *Standard* responden a las funciones más importantes: abrir y/o guardar nuevos proyectos, añadir formularios, hacer visibles las distintas ventanas del entorno de desarrollo, etc. Todos los botones tienen su correspondiente comando. Son importantes los botones que permiten arrancar y/o parar la ejecución de un proyecto, pasando de modo diseño a modo de ejecución y viceversa. El menú File tiene pocas novedades. Lo más importante es la distinción entre proyectos y todos los demás ficheros. Como ya se ha dicho, un proyecto reúne y organiza todos los ficheros que componen el programa o aplicación. Estos ficheros pueden ser *formularios*, *módulos, clases, recursos*, etc, pero la extensión de este trabajo de tesis, solo contemplará proyectos y formularios. En este menú está el comando *Make ProjectName.exe*, en nuestro caso *Make SCARA.EXE,* que permite crear ejecutables de los proyectos. Tampoco el menú *Edit* aporta cambios importantes sobre lo que es habitual. Por el contrario el menú *View*, generalmente de poca utilidad, es bastante propio de Visual Basic®. Este menú permite hacer aparecer en pantalla las distintas ventanas del entorno de desarrollo, algo muy similar a la interfase de desarrollo MPLAB que veremos más adelante, así como acceder a un formulario o al código relacionado con un control, y manejar funciones y procedimientos. El

menú *Project* permite añadir distintos tipos de elementos a un proyecto. Con *Project/Properties* se puede elegir el tipo de proyecto, en nuestro caso el tipo de proyecto será un ejecutable estándar, y determinar el formulario con el que se arrancará la aplicación (*Startup Object*). Con el comando *Components* se pueden añadir nuevos controles a la *Toolbox* que aparece a la izquierda de la pantalla.

El menú *Format* contiene opciones para controlar el aspecto de la aplicación (alinear controles, espaciarlos uniformemente, etc.). Los menús *Debug* y *Run* permiten controlar la ejecución de las aplicaciones. Con *Debug* se puede ver en detalle cómo funcionan, ejecutando paso a paso, yendo hasta una línea de código determinada, etc. Esto es especialmente útil cuando haya que encontrar algunos errores, que normalmente no se puede detectar en que parte del código se generan, ejecutando paso a paso, o viendo resultados intermedios.

En el menú *Tools* se encuentran los comandos para arrancar el *Menu Editor* y para establecer las opciones del programa. En *Tools/Options* se encuentran las opciones relativas al proyecto en el que se trabaja. La lengüeta *Environment* determina las propiedades del entorno del proyecto, como las opciones para actualizar o no los ficheros antes de cada ejecución; en General se establece lo referente a la rejilla o *grid* que aparece en el formulario; *Editor* permite establecer la necesidad de declarar todas las variables junto con otras opciones de edición, como si se quieren ver o no todos los procedimientos juntos en la misma ventana, y si se quiere ver una línea separadora entre procedimientos. *Editor Format* permite seleccionar el tipo de letra y los códigos de color con los que aparecen los distintos elementos del código. La opción *Advanced* hace referencia entre otras cosas a la opción de utilizar Visual Basic® en dos formatos SDI (Single Document Interface) y MDI (Multiple Document Interface). En nuestro caso, utilizaremos el formato MDI para programa de interfase sencilla.

Por último, la ayuda que se encuentra en el menú Help, se basa fundamentalmente en una clasificación temática ordenada de la información disponible , en una clasificación alfabética de la información y en la búsqueda de información sobre algún tema por el nombre.

3.2.1.1 Las herramientas (toolbox)

Incluye los controles con los que se puede diseñar la pantalla de la aplicación. Estos controles son por ejemplo botones, etiquetas, cajas de texto, zonas gráficas, etc. Para introducir un control en el formulario simplemente hay que pulsar en el icono adecuado del *toolbox* y colocarlo en el formulario con la posición y el tamaño deseado, presionando y arrastrando con el ratón. Pulsando dos veces sobre el icono de un control aparece éste en el centro del formulario y se puede modificar su tamaño y/o trasladar con el ratón como se desee. El número de controles que pueden aparecer en esta ventana varía con la configuración del sistema. Para introducir nuevos componentes se utiliza el comando *Components* en el menú *Project*, con lo cual se abre el cuadro de diálogo correspondiente.

3.2.1.2 Formularios (forms) y módulos

Los formularios son las zonas de la pantalla sobre las que se diseña el programa y sobre las que se sitúan los controles o herramientas del *toolbox*. Al ejecutar el programa, el *form* se convertirá en la ventana de la aplicación, donde aparecerán los botones, el texto, los gráficos, etc. Para lograr una mejor presentación existe una malla que permite alinear los controles manualmente de una forma precisa. Esta malla sólo será visible en el proceso de diseño del programa; al ejecutarlo no se verá. De cualquier forma, se puede desactivar la malla o cambiar sus características en el menú *Tools/Options/General*, cambiando la opción *Align Controls to Grid*. Exteriormente, los formularios tienen una estructura similar a la de cualquier ventana. Sin embargo, también poseen un código de programación que estará escrito en Basic, y que controlará algunos aspectos del formulario, sobre todo en la forma de reaccionar ante las acciones del usuario ó eventos. El formulario y los controles en él situados serán la base del programa. Una aplicación puede tener varios formularios, pero siempre habrá uno con el que arrancará la aplicación; este formulario se determina a partir del menú *Project/Properties*, en *Startup Objects*.

Cuando se crea un programa en Visual Basic® habrá que dar básicamente dos pasos:

1. Diseñar y preparar la parte gráfica en la forma descrita.
2. Realizar la programación ante los distintos eventos.

3.2.1.3 La ventana de proyecto

Esta ventana, la que se pudo apreciar en la figura 3.4, permite acceder a los distintos formularios y módulos que componen el proyecto. Desde ella se puede ver el diseño gráfico de dichos formularios con el botón *View Object* , y también permite editar el código que contienen con el botón *View Code*. Estos botones están situados en la parte superior de la ventana, debajo de la barra de títulos. Los módulos estándar ó ficheros **.bas* que contienen sólo código que, en general, puede ser utilizado por distintos formularios y/o controles del proyecto e incluso por varios proyectos.

3.2.1.4 La ventana de propiedades

Todos los objetos tienen unas propiedades que los definen: su nombre (*Name*), su etiqueta o título (*Caption*), el texto que contiene (*Text*), su tamaño y posición, su color, si está activo o no (*Enabled*), etc. La podemos apreciar en la figura 3.6.

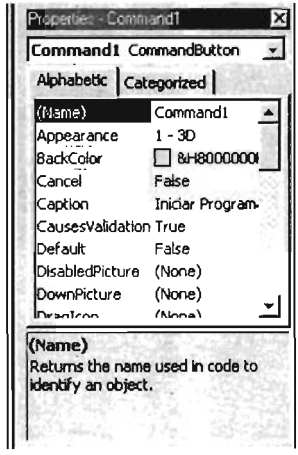

Figura 3.6. Ventana Properties.

Todas estas propiedades se almacenan dentro de cada control o formulario en forma de estructura. Por tanto, si por ejemplo en algún momento se quiere modificar el nombre de un botón basta con hacerlo en la ventana de propiedades al diseñar el programa o en el código en Basic para que tal modificación ocurra durante la ejecución, mediante el operador punto (.), en la forma:

```
Boton1.Name = "NuevoNombre"
```

Para realizar una modificación de las propiedades de un objeto durante el diseño del programa, se activará la ventana de propiedades. Esta ventana tiene dos lengüetas, que permiten ordenar las propiedades alfabéticamente o por categorías. Utilizando la forma que sea más cómoda se localizará con ayuda de la barra de desplazamiento la propiedad que se quiera modificar. Al pulsar sobre ella puede activarse un menú desplegable con las distintas opciones, o bien puede modificarse directamente el valor de la propiedad. Si esta propiedad tiene sólo unos valores fijos, como por ejemplo los colores, puede abrirse un cuadro de diálogo para elegir un color, o el tamaño y tipo de letra que se desee si se trata de una propiedad *Font*.

La ventana *FormLayout*, que permite determinar la posición en la que el formulario aparecerá sobre la pantalla cuando se haga visible al ejecutar la aplicación. La podemos apreciar en la figura 3.7, se encuentra típicamente debajo de la ventana *Properties*.

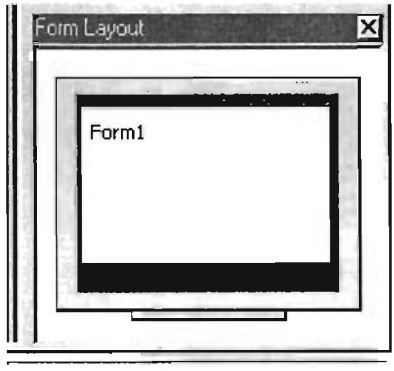

Figura 3.7. Ventana Form Layout

3.2.2. Creación de programas ejecutables

Una vez finalizada la programación de la nueva aplicación, la siguiente tarea es la creación de un programa ejecutable para su distribución e instalación en cuantos ordenadores se desee, incluso aunque en ellos no esté instalado Visual Basic®. Para crear un programa ejecutable se utiliza el comando *Make SCARA.exe* en el menú *File*. De esta manera se generará un fichero cuya extensión será **.exe*.

Para que este programa funcione en un ordenador solamente se necesita que el fichero MSVBVM60.DLL esté instalado en el directorio c:\Windows\System.

En el caso de el programa se vaya a utilizar en un ordenador en el que no esté instalado Visual Basic® puede resultar interesante construir unos disquetes de instalación que simplifiquen la tarea de instalar el programa en cualquier ordenador sin tener que ver en cada caso cuáles son los ficheros que faltan. Visual Basic® dispone de un Asistente que, interactivamente, simplifica enormemente la tarea de creación de disquetes de instalación.

3.2.3 Utilización del code editor

El editor de código o *Code Editor* de Visual Basic® es la ventana en la cual se escriben las sentencias del programa. Para abrir la ventana del editor de código se elige *Code* en el menú *View*. Cada formulario, cada módulo de clase y cada módulo estándar tienen su propia ventana de código. Aunque el aspecto de dicha ventana no tiene nada de particular, el Code Editor de Visual Basic® ofrece muchas ayudas al usuario que requieren una explicación más detenida. En primer lugar, el *Code Editor* utiliza un código de colores para destacar cada elemento del programa. Así, el código escrito por el usuario aparece en negro, las palabras clave de Basic en azul, los comentarios en verde, los errores en rojo. En la parte superior de esta ventana aparecen dos listas desplegables. La de la izquierda corresponde a los distintos elementos del formulario, la parte General es común a todo el formulario, el propio formulario y los distintos controles que están incluidos en él. La lista desplegable de la derecha muestra los distintos procedimientos que se corresponden con el elemento seleccionado en la lista de la izquierda. Lo podemos apreciar en la figura 3.8.

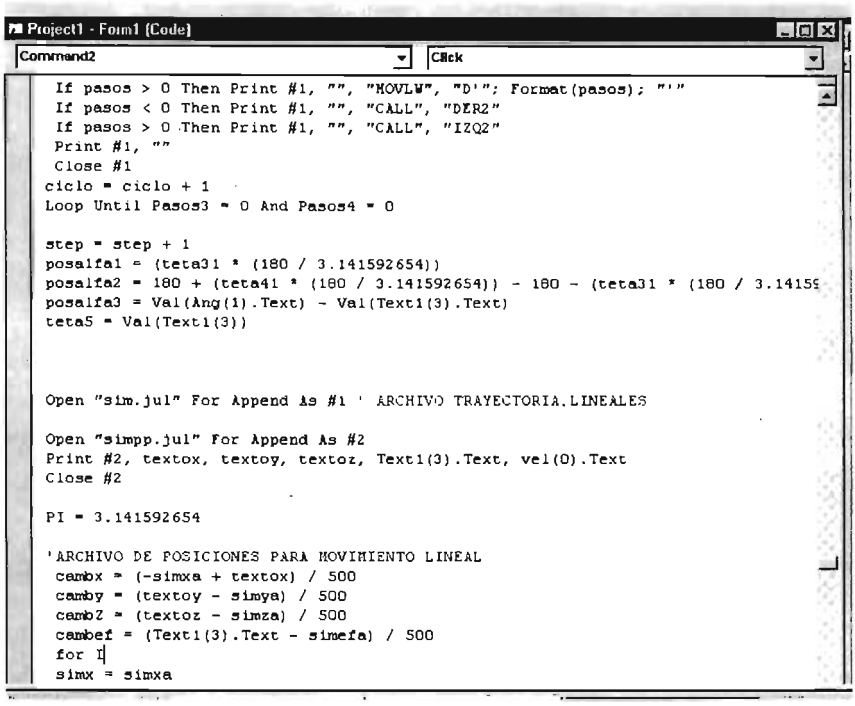

Figura 3.8. Ventada del Editor de Código ó Code Editor

Estas dos listas permiten localizar fácilmente el código que se desee programar o modificar. Otra opción muy interesante es la de completar automáticamente el código (*Automatic Completion Code*). Al teclear el punto detrás del nombre de un objeto, automáticamente se abre una lista con las propiedades de ese objeto. Pulsando la tecla *Tab* se introduce el nombre completo de la propiedad seleccionada. A esta característica se le conoce como *AutoListMembers*. Por otra parte, la opción *AutoQuickInfo* hace que al comenzar a teclear el nombre de una función aparezca información sobre esa función: nombre, argumentos y valor de retorno.

3.2.4 Lenguaje Basic

Al igual que con nuestro microcontrolador, en este apartado veremos de forma general el lenguaje Basic así como los conceptos básicos para su uso.

Un programa está constituido en un sentido general por variables que contienen los datos con los que se trabaja y por algoritmos que son las sentencias que operan sobre estos datos. Estos datos y algoritmos suelen estar incluidos dentro de funciones o procedimientos. Un procesador digital únicamente es capaz de entender aquello que está constituido por conjuntos de unos y ceros, nuevamente, la analogía con nuestro microcontrolador. A esto se le llama lenguaje de máquina o binario, y es muy difícil de manejar. Por ello, desde casi los primeros años de los ordenadores, se comenzaron a desarrollar los llamados lenguajes de alto nivel que están mucho más cerca del lenguaje natural. Estos lenguajes están basados en el uso de *identificadores*, tanto para los *datos* como para las componentes elementales del programa, que en algunos lenguajes se llaman *rutinas*, *procedimientos*, o *funciones*. Además, cada lenguaje dispone de una sintaxis o conjunto de reglas con las que se indica de modo inequívoco las operaciones que se quiere realizar. Los lenguajes de alto nivel son más o menos comprensibles para el usuario, pero no para el procesador. Para que éste pueda ejecutarlos es necesario traducirlos a su propio lenguaje de máquina. Al paso del lenguaje de alto nivel al lenguaje de máquina se le denomina compilación. En *Visual Basic*® esta etapa no se aprecia tanto como en otros lenguajes donde el programador tiene que indicar al ordenador explícitamente que realice dicha compilación. Los programas de Visual Basic® se dice que son interpretados y no compilados ya que el código no se convierte a código máquina sino que hay otro programa que durante la ejecución "interpreta" las líneas de código que ha escrito el programador. En general durante la ejecución de cualquier programa, el código es cargado por el sistema operativo en la memoria RAM. Esta sin duda es una característica fundamental de Visual Basic®

Para ayudar al programador a no perderse en un programa muy amplio, Visual Basic®, al igual que otros software de programación, dispone de la opción de comentario, que funciona exactamente igual que en el lenguaje ensamblador, solo que aquí no importa que este en la cuarta columna, de hecho cualquier texto después de la comilla (') será considerado un comentario.

```
B = C*x+4 ' esto es un comentario
```

Los comentarios son tremendamente útiles para futuras revisiones y correcciones, cuando se trata de proyectos realmente complejos, o desarrollados por varias personas. Otro aspecto práctico en la programación es la posibilidad de escribir una sentencia en más de una línea para poder visualizarlo en pantalla. En otro caso la lectura del código se hace mucho más pesada. Para ello es necesario dejar un *espacio en blanco* al final de la línea y escribir el *carácter* (_) tal y como se muestra en el siguiente ejemplo:

```
Private Sub Command4_Click() ' iniciar simulacion
Timer1.Enabled = True
```

Una limitación a los comentarios en el código es que no se pueden introducir en una línea en la que se ha introducido el carácter de continuación (_). La sintaxis de Visual Basic® permite también incluir varias sentencias en una misma línea. Para ello las sentencias deben ir separadas por el carácter dos puntos (:)

Ámbito de las variables y los procedimientos

Se entiende por *ámbito* de una variable la parte de la aplicación donde la variable es válida y por lo tanto puede ser utilizada en cualquier expresión.

Un módulo puede contener variables y procedimientos o funciones públicos y privados. Los públicos son aquellos a los que se puede acceder libremente desde cualquier punto del proyecto. Para definir una variable, un procedimiento o una función como público es necesario preceder a la definición de la palabra Public, como por ejemplo:

```
Public step, flag, acel10, acel11, acel20, acel21 As Integer
```

Al declarar una variable como Public, esta se vuelve accesible desde cualquier punto del programa que se encuentre en la misma ventana de código, pero si se requiere una variable de otro modulo, se debe preceder con el nombre del modulo y separarlo con un punto.

Una variable Private, por el contrario, no es accesible desde ningún otro módulo distinto de aquél en el que se haya declarado.

Se llama variable local a una variable definida dentro de un procedimiento o función. Las variables locales no son accesibles más que en el procedimiento o función en que están definidas. Una variable local es reinicializada cada vez que se entra en el procedimiento. Es decir, una variable local no conserva su valor entre una llamada al procedimiento y la siguiente.

Variables y funciones de ámbito global

Se puede acceder a una variable o función global desde cualquier parte de la aplicación. Para hacer que una variable sea global, hay que declararla en la parte general (General) de un módulo *.bas o de un formulario de la aplicación. Para declarar una variable global se utiliza la palabra Public. Por ejemplo:

```
Public t1ant, t2ant, REL1, REL2, REL3 As Double
```

De esta forma se podrá acceder a las variables var1_global, var2_global desde todos los formularios, las variables globales se pueden acceder desde cualquier punto del programa como si las hubiéramos declarado en ese módulo.

Identificadores

Debido a que los datos en los sistemas informáticos son almacenados en direcciones de memoria en disco o memoria RAM, estas direcciones suelen cambiar constantemente, así que para identificar un dato en específico nos es conveniente darlo un nombre concreto, en este caso denominado Identificador, algo muy similar a lo que hacemos en el principio de cada programa en lenguaje ensamblador, con la diferencia de que en este caso, es el programa el que designa la dirección de memoria que le convenga. En Visual Basic® hay gran libertad para escoger los nombres de nuestros identificadores, de programas, variables y constantes, de esta forma para referirnos a un dato, no tenemos que especificar su dirección de memoria que consiste en un largo número hexadecimal, sino por su nombre, por ejemplo, CordenadaX.

Variables y constantes

Una variable es un nombre que designa a una zona de memoria, que contiene un valor de un tipo de información. Tal y como su nombre indica, las variables pueden cambiar su valor a lo largo de la ejecución de un programa. Completando a las variables existe lo que se denomina constantes las cuales son identificadores pero con la particularidad de que el valor que se encuentra en ese lugar de la memoria sólo puede ser asignado una única vez. El tratamiento y tipos de datos es igual al de las variables. Para declarar un dato como constante únicamente es necesario utilizar la palabra Const en la declaración de la variable. Si durante la ejecución se intenta variar su valor se producirá un error. Esto nos es útil para no variar el valor de una constante por error.

Ejemplos:

```
Const k = 500 ' Las constantes son privadas por defecto.
Public Const Pal = "HELP" ' Declaración de una constante pública.
Private Const Entero As Integer = 5 ' Declaración de un entero constante.
Const Hola = "Hi", PI As Double = 3.14 ' Múltiples constantes en una línea.
```

Visual Basic® tiene sus propias constantes, muy útiles por cierto. Algunas ya se han visto al hablar de los colores. En general estas constantes empiezan por ciertos caracteres como vb y van seguidas de una o más palabras que indican su significado. Para ver las constantes disponibles se puede utilizar el comando *View/Object Browser*, y se pueden apreciar en la imagen 3.9

Nombres de variables

El nombre de una variable o de una constante tiene que comenzar siempre por una letra y puede tener una longitud hasta 255 caracteres. No se admiten espacios o caracteres en blanco, ni puntos (.), ni otros caracteres especiales. Los caracteres pueden ser letras, dígitos, el carácter de subrayado (_) y los caracteres de declaración del tipo de la variable (%, &, #, !, @, y $). El nombre de una variable no puede ser una palabra reservada del lenguaje (*For*, *If*, *Loop*, *Next*, *Val*, *Hide*, *Caption*, *And*, etc.). Usualmente las palabras reservadas del lenguaje aparecen de color azul en el editor de código, lo que hace más fácil saber si una palabra es reservada o no.

En Visual Basic® es habitual utilizar las letras mayúsculas para separar las distintas palabras que están unidas en el nombre de una variable, aunque en general, el programador puede denominarlas como mejor le convenga. La declaración de una variable o la primera vez que se utiliza determinan cómo se escribe en el resto del programa.

Al igual que C y otros lenguajes de programación, Visual Basic® dispone de distintos tipos de datos, aplicables tanto para constantes como para variables. En la tabla 3.1 se enumeran los distintos tipos de datos que pueden contener las variables.

Tipo	Descripción	Carácter de declaración	Rango
Boolean	Binario		True o False
Byte	Entero		corto 0 a 255
Integer	Entero (2 bytes)	%	-32768 a 32767
Long	Entero largo (4 bytes)	&	-2147483648 a 2147483647
Single	Real simple precisión (4 bytes)	!	-3.40E+38 a 3.40E+38
Double	Real doble precisión (8 bytes)	#	-1.79D+308 a 1.79D+308
Currency	Número con punto decimal fijo (8 bytes)	@	-9.22E+14 a 9.22E+14
String	Cadena de caracteres (4 bytes + 1byte/car hasta 64 K)	$	0 a 65500 caracteres
Date	Fecha (8 bytes)		1 de enero de 100 a 31 de diciembre de 9999. Indica también la hora, desde 0:00:00 a 23:59:59.
Variant	Fecha/hora números enteros, reales, o caracteres (16 bytes + 1 byte/car. en cadenas de caracteres) ninguno		F/h: como Date números: mismo rango que el tipo de valor almacenado
User-defined	Cualquier tipo de dato o estructura de datos. Se crean utilizando la sentencia Type		

Tabla 3.1. Tipos de variables empleadas en Visual Basic®

En el lenguaje *Visual Basic®* existen dos formas de agrupar varios valores bajo un mismo nombre. La primera de ellas son los *arrays*, que agrupan datos de tipo homogéneo. La segunda son las *estructuras*, que agrupan información heterogénea o de distinto tipo. En *Visual Basic®* las estructuras son verdaderos *tipos de datos definibles por el usuario.* Para declarar las variables se utiliza la sentencia siguiente:

```
Dim Radio As Double, Superficie as Single
Dim Nombre As String
Dim Etiqueta As String * 10
Dim Francos As Currency
Dim Longitud As Long, X As Currency
```

En *Visual Basic®* no es estrictamente necesario declarar todas las variables que se van a utilizar, y hay otra forma de declarar las variables anteriores, utilizando los caracteres especiales vistos anteriormente. Así por ejemplo, el tipo de las variables del ejemplo anterior se puede declarar al utilizarlas en las distintas expresiones, poniéndoles a continuación el carácter que ya se indicó, en la forma:

```
Radio# doble precisión
Nombre$ cadena de caracteres
Francos@ unidades monetarias
Longitud& entero largo
```

Esta forma de indicar el tipo de dato no es la más conveniente. Se mantiene en las sucesivas versiones de *Visual Basic®* por la compatibilidad con códigos anteriores. Es preferible utilizar la notación donde se escribe directamente el tipo de dato.

Elección del tipo de una variable

Si en el código del programa se utiliza una variable que no ha sido declarada, se considera que esta variable es de tipo *Variant*. Las variables de este tipo se adaptan al tipo de información o dato que se les asigna en cada momento. Son pues variables muy flexibles, pero su uso debe restringirse porque ocupan *más memoria* (almacenan el tipo de dato que contienen, además del propio valor de dicho dato) y requieren más tiempo de CPU que los restantes tipos de variables.

En general es el tipo de dato es lo que determina qué tipo de variable se debe utilizar.

Integer para numerar las filas y columnas de una matriz no muy grande
. *Long* para numerar los habitantes de una ciudad o los números de teléfonos
. *Boolean* para una variable con sólo dos posibles valores (sí o no)
. *Single* para variables físicas con decimales que no exijan precisión
. *Double* para variables físicas con decimales que exijan precisión
. *Currency* para cantidades grandes de dinero

Declaración explícita de variables

Una variable que se utiliza sin haber sido declarada toma por defecto el tipo *Variant*. Puede ocurrir que durante la programación, se cometa un error y se escriba mal el nombre de una variable. *Visual Basic*® supondría que ésta es una nueva variable de tipo *Variant*. Para evitar este tipo de errores, se puede indicar a *Visual Basic*® que genere un mensaje de error siempre que encuentre una variable no declarada previamente. Para ello lo más práctico es establecer una opción por defecto, utilizando el comando *Environment* del menú *Tools/Options*; en el cuadro que se abre se debe poner *Yes* en la opción *Require Variable Declaration*. También se puede hacer esto escribiendo la sentencia siguiente en la sección de declaraciones de cada formulario y de cada módulo: Option Explicit

3.2.4.1 Operadores

La Tabla 3.2 presenta el conjunto de operadores que soporta *Visual Basic®*.

Tipo	Operación	Operador
Aritmético	Exponenciación	^
	Cambio de signo (operador unario)	-
	Multiplicación, división	*, /
	División entera	\
	Resto de una división entera	Mod
	Suma y resta	+, -
Concatenación	Concatenar o enlazar	& +
	Relacional Igual a	=
	Distinto	<>
	Menor que / menor o igual que	< <=
	Mayor que / mayor o igual que	> >=
Otros	Comparar dos expresiones de caracteres	Like
	Comparar dos referencias a objetos	Is
Lógico	Negación	Not
	And	And
	Or inclusivo	Or
	Or exclusivo	Xor
	Equivalencia (opuesto a Xor)	Eqv
	Implicación (*False* si el primer operando es *True* y el segundo operando es *False*)	Imp

Tabla 3.2 Conjunto de operadores soportados por Visual Basic.

3.2.4.2 Sentencias de control

Las *sentencias de control*, denominadas también *estructuras de control*, permiten tomar decisiones y realizar un proceso repetidas veces. Son los denominados bifurcaciones y bucles. Este tipo de estructuras son comunes en cuanto a

concepto en la mayoría de los lenguajes de programación, aunque su sintaxis puede variar de un lenguaje de programación a otro. Se trata de unas estructuras muy importantes ya que son las encargadas de controlar el flujo de un programa según los requerimientos del mismo. Visual Basic® dispone de las siguientes estructuras de control:

If ... Then ... Else
Select Case
For ... Next
Do ... Loop
While ... Wend
For Each ... Next

Sentencia IF ... THEN ... ELSE ...

Esta estructura permite ejecutar condicionalmente una o más sentencias y puede escribirse de dos formas. La primera ocupa sólo una línea y tiene la forma siguiente:

```
If condicion Then sentencia1 [Else sentencia2]
```

La segunda es más general y se muestra a continuación:

```
If condicion Then
sentencia(s)
[Else
sentencia(s)]
End If
```

Si *condicion* es *True (verdadera)*, se ejecutan las sentencias que están a continuación de *Then*, y si *condicion* es *False (falsa)*, se ejecutan las sentencias que están a continuación de *Else*, si esta cláusula ha sido especificada (pues es opcional). Para indicar que se quiere ejecutar uno de varios bloques de sentencias dependientes cada uno de ellos de una condición, la estructura adecuada es la siguiente:

```
If condicion1 Then
sentencias1
ElseIf condicion2 Then
sentencias2
Else
sentencia-n
End If
```

Si se cumple la *condicion1* se ejecutan las *sentencias1*, y si no se cumple, se examinan secuencialmente las condiciones siguientes hasta *Else*, ejecutándose las sentencias correspondientes al primer *ElseIf* cuya condición se cumpla. Si todas las condiciones son falsas, se ejecutan las sentencias-n correspondientes a *Else*, que es la opción por defecto.

Sentencia SELECT CASE

Esta sentencia permite ejecutar una de entre varias acciones en función del valor de una expresión. Es una alternativa a *If ... Then ... ElseIf* cuando se compara la misma expresión con diferentes valores. Su forma general es la siguiente:

```
Select Case expresion
Case etiq1
[ sentencias1]
Case etiq2
[ sentencias2]
Case Else
sentenciasn
End Select
```

donde *expresion* es una expresión numérica o alfanumérica, y *etiq1*, *etiq2*, ... pueden adoptar las formas siguientes:

1. *expresion*
2. *expresion* To *expresion*
3. Is *operador-de-relación expresion*
4. *combinación de las anteriores separadas por comas*

Por ejemplo, cuando se utiliza la forma *expresion To expresion*, el valor más pequeño debe aparecer en primer lugar. Cuando se ejecuta una sentencia *Select Case*, Visual Basic® evalúa la *expresion* y el control del programa se transfiere a la sentencia cuya etiqueta tenga el mismo valor que la expresión evaluada, ejecutando a continuación el correspondiente bloque de sentencias. Si no existe un valor igual a la *expresion* entonces se ejecutan las sentencias a continuación de *Case Else*.

Sentencia FOR ... NEXT

La sentencia *For* da lugar a un lazo o bucle, y permite ejecutar un conjunto de sentencias cierto número de veces. Su forma general es:

```
For variable = expresion1 To expresion2 [Step expresion3]
[sentencias]
Exit For
[sentencias]
Next [variable]
```

Cuando se ejecuta una sentencia *For*, primero se asigna el valor de la *expresion1* a la variable y se comprueba si su valor es mayor o menor que la *expresion2*. En caso de ser menor se ejecutan las sentencias, y en caso de ser mayor el control del programa salta a las líneas a continuación de *Next*. Todo esto sucede en caso de ser la *expresion3* positiva. En caso contrario se ejecutarán las sentencias cuando la variable sea mayor que *expresion2*. Una vez ejecutadas las sentencias, la variable se incrementa en el valor de la *expresion3*, o en 1 si *Step* no se especifica, volviéndose a efectuar la comparación entre la variable y la *expresion2*, y así sucesivamente. La sentencia *Exit For* es opcional y permite salir de un bucle *For ... Next* antes de que éste finalice. Por ejemplo,

```
MyString="Ingenieríaa "
For Words = 4 To 1 Step -1 ' 4 veces decrementando de 1 en 1.
For Chars = Words To Words+4 '5 veces.
MyString = MyString & Chars ' Se añade el número Chars al string.
Next Chars ' Se incrementa el contador
```

```
MyString = MyString & " " ' Se añade un espacio.
Next Words
'El valor de MyString es: Ingeniería 34567 23456 12345
```

Sentencia DO ... LOOP

Un *Loop (bucle)* repite la ejecución de un conjunto de sentencias mientras una condición dada sea cierta, o hasta que una condición dada sea cierta. La condición puede ser verificada antes o después de ejecutarse el conjunto de sentencias. Sus posibles formas son las siguientes:

```
Formato 1:
Do [{While/Until} condicion]
[ sentencias]
[Exit Do]
[ sentencias]
Loop

Formato 2:
Do
[ sentencias]
[Exit Do]
[ sentencias]
Loop [{While/Until} condicion]
```

La sentencia opcional *Exit Do* permite salir de una bucle *Do ... Loop* antes de que finalice éste. Por ejemplo,

```
Check = True ' Se inicializan las variables.
Counts = 0
Do ' Empieza sin comprobar ninguna condición.
Do While Counts < 20 ' Bucle que acaba si Counts>=20 o con Exit Do.
Counts = Counts + 1 ' Se incrementa Counts.
If Counts = 10 Then ' Si Counts es 10.
Check = False ' Se asigna a Check el valor False.
Exit Do ' Se acaba el segundo Do.
```

```
End If
Loop
Loop Until Check = False ' Salir del "loop" si Check es False.
```

En el ejemplo mostrado, se sale de los bucles siempre con *Counts = 10*. Es necesario fijarse que si se inicializa *Counts* con un número mayor o igual a 10 se entraría en un bucle infinito (el primer bucle acabaría con *Counts = 20* pero el segundo no finalizaría nunca, bloqueándose el programa y a veces el ordenador).

Sentencia WHILE ... WEND

Esta sentencia es otra forma de generar bucles que se recorren mientras se cumpla la condición inicial. Su estructura es la siguiente:

```
While condicion
[ sentencias]
Wend
Por ejemplo,
Counts = 0 ' Se inicializa la variable.
While Counts < 20 ' Se comprueba el valor de Counts.
Counts = Counts + 1 ' Se incrementa el valor de Counts.
Wend ' Se acaba el bucle cuando Counts > 19.
```

Sentencia FOR EACH ... NEXT

Esta construcción es similar al bucle *For*, con la diferencia de que la variable que controla la repetición del bucle no toma valores entre un mínimo y un máximo, sino a partir de los elementos de un array (o de una colección de objetos). La forma general es la siguiente:

```
For Each variable In grupo
[ sentencias]
Next variable
```

Con arrays *variable* tiene que ser de tipo *Variant*. Con colecciones *variable* puede ser *Variant* o una variable de tipo *Object*. Esta construcción es muy útil

cuando no se sabe el número de elementos que tiene el array o la colección de objetos.

3.3 Funciones y procedimientos Sub en Visual Basic®

En Visual Basic® se distingue entre *funciones* y *procedimientos Sub*. En ocasiones se utiliza la palabra genérica *procedimiento* para ambos. La fundamental diferencia entre un *procedimiento Sub* y una *función* es que ésta última puede ser utilizada en una expresión porque tiene un *valor de retorno*. El valor de retorno ocupa el lugar de la llamada a la función donde esta aparece. Por ejemplo, si en una expresión aparece *sin(x)* se calcula el seno de la variable *x* y el resultado es el valor de retorno que sustituye a *sin(x)* en la expresión en la que aparecía. Por tanto, las funciones devuelven valores, a diferencia de los procedimientos que no devuelven ningún valor, y por tanto no pueden ser utilizadas en expresiones. Un *procedimiento Sub* es un segmento de código independiente del resto, que una vez llamado por el programa, ejecuta un número determinado de instrucciones, sin necesidad de devolver ningún valor al mismo (puede dar resultados modificando los argumentos), mientras que una función siempre tendrá un valor de retorno.

Los nombres de los procedimientos tienen reglas de visibilidad parecidas a las de las variables. Para llamar desde un formulario a un procedimiento *Public* definido en otro formulario es necesario preceder su nombre por el del formulario en que está definido. Sin embargo, si se desea llamar a un procedimiento definido en un módulo estándar (**.bas*) no es necesario precederlo del nombre del módulo más que si hay coincidencia de nombre con otro procedimiento de otro módulo estándar.

3.3.1 Funciones

La sintaxis correspondiente a una función es la siguiente:

```
[Static] [Private] Function nombre ([ parámetros]) [As tipo]
[ sentencias]
[ nombre = expresion]
[Exit Function]
[ sentencias]
```

[nombre = expresion]
End Function

donde *nombre* es el nombre de la función. Será de un tipo u otro dependiendo del dato que devuelva. Para especificar el tipo se utiliza la cláusula *As Tipo* (*Integer*, *Long*, *Single*, *Double*, *Currency*, *String* o *Variant*). *parámetros* son los argumentos que son pasados cuando se llama a la función. *Visual Basic®* asigna el valor de cada argumento en la llamada al parámetro que ocupa su misma posición. Si no se indica un tipo determinado los argumentos son *Variant* por defecto. El *nombre de la función*, que es el valor de retorno, actúa como una variable dentro del cuerpo de la función. El valor de la variable *expresion* es almacenado en el propio nombre de la función. Si no se efectúa esta asignación, el resultado devuelto será 0 si la función es numérica, nulo ("") si la función es de caracteres, o *Empty* si la función es *Variant*. *Exit Function* permite salir de una función antes de que ésta finalice y devolver así el control del programa a la sentencia inmediatamente a continuación de la que efectuó la llamada a la función.

La sentencia *End Function* marca el final del código de la función y, al igual que la *Exit Function*, devuelve el control del programa a la sentencia siguiente a la que efectuó la llamada, pero lógicamente una vez finalizada la función. La *llamada a una función* se hace de diversas formas. Por ejemplo, una de las más usuales es la siguiente:

variable = nombre([argumentos])

donde *argumentos* son un lista de constantes, variables o expresiones separadas por comas que son pasadas a la función. En principio, el número de argumentos debe ser igual al número de parámetros de la función. Los *tipos* de los argumentos deben coincidir con los tipos de sus correspondientes parámetros, de lo contrario puede haber fallos importantes en la ejecución del programa. Esta regla no rige si los argumentos se pasan *por valor* (concepto que se verá más adelante). En cada llamada a una función hay que incluir los paréntesis, aunque ésta no tenga argumentos.

El siguiente ejemplo corresponde a una función que devuelve como resultado la raíz cuadrada de un número *N*:

```
Function Raiz (N As Double) As Double
If N < 0 Then
Exit Function
Else
Raiz = Sqr(N)
End Function
```

La llamada a esta función se hace de la forma siguiente:

```
Cuadrada = Raiz(Num)
```

A diferencia de C y C++ en Visual Basic® no es necesario devolver explícitamente el valor de retorno, pues el nombre de la función ya contiene el valor que se desea devolver. Tampoco es necesario declarar las funciones antes de llamarlas.

Procedimientos Sub

La sintaxis que define un *procedimiento Sub* es la siguiente:

```
[Static] [Private] Sub nombre [( parámetros)]
[ sentencias]
[Exit Sub]
[ sentencias]
End Sub
```

La explicación es análoga a la dada para funciones. La llamada a un *procedimiento Sub* puede ser de alguna de las dos formas siguientes:

```
Call nombre[(argumentos)]
```

o bien, sin pasar los argumentos entre paréntesis, sino poniéndolos a continuación del nombre simplemente separados por comas:

```
nombre [argumentos]
```

A diferencia de una *función*, un *procedimiento Sub* no puede ser utilizado en una expresión pues no devuelve ningún valor. Por supuesto una función puede ser llamada al modo de un *procedimiento Sub*, pero en esta caso no se hace nada con el valor devuelto por la función.

El siguiente ejemplo corresponde a un *procedimiento Sub* que devuelve una variable *F* que es la raíz cuadrada de un número *N*..

```
Sub Raiz (N As Double, F As Double)
If N < 0 Then
Exit Sub 'Se mandaría un mensaje de error
Else
F = Sqr(N)
End If
End Sub
```

La llamada a este *procedimiento Sub* puede ser de cualquiera de las dos formas siguientes:

```
Raiz N, F
Call Raiz(N, F)
```

En el ejemplo anterior, el resultado obtenido al extraer la raíz cuadrada al número *N* se devuelve en la variable *F* pasada como argumento, debido a que como se ha mencionado anteriormente, un *procedimiento Sub* no puede ser utilizado en una expresión.

Procedimientos recursivos

Se dice que una *función* (*Function*) es *recursiva* o que un *procedimiento Sub* es *recursivo* si se llaman a sí mismos.

A continuación se presenta una ejemplo de una función que calcula el factorial de un número programada de forma recursiva.

```
Function Factorial (N As Integer) As Long
If N = 0 Then
Factorial = 1 'Condición de final
Else
Factorial = N * Factorial (N - 1)
End If
End Function
```

En este ejemplo, si la variable N que *se le pasa* a la función vale 0, significará que se ha llegado al final del proceso, y por tanto *se le asigna* el valor 1 al valor del factorial (recordar que 0! = 1). Si es distinto de 0, la función se llama a ella misma, pero variando el argumento a (N-1), hasta llegar al punto en el que N-1=0, finalizándose el proceso.

3.3.2. Procedimientos con argumentos opcionales

Puede haber procedimientos en los que algunos de los argumentos incluidos en su definición sean opcionales, de forma que el programador pueda o no incluirlos en la llamada de dichos procedimientos. La forma de incluir un argumento opcional es incluir la palabra *Optional* antes de dicho argumento en la definición del procedimiento. Si un argumento es opcional, todos los argumentos que vienen a continuación deben también ser opcionales. Cuando un argumento es opcional y en la llamada es omitido, el valor que se le pasa es un *Variant* con valor *Empty*. A los argumentos opcionales se les puede dar en la definición del procedimiento un valor por defecto para el caso en que sean omitidos en la llamada, como por ejemplo:

```
Private Sub miProc(x as Double, Optional n=3 As Integer)
sentencias
End Sub
```

3.4. Funciones matemáticas

Visual Basic 6.0® dispone tambien de un número de funciones matemáticas, tal y como lo podemos apreciar en la tabla 3.3.

Función Matemática	Función VB.
Valor Absoluto	Abs(x)
Arco Tangente	Atn(x)
Exponencial	Exp(x)
Parte Entera	Int(x)
Logaritmo	Log(x)
Redondeo	Round(x,dec)
Aleatorio	Rnd
Seno y coseno	Sin(x), cos(x)
Tangente	Tan(x)
Raíz Cuadrada	Sqr(x)
Signo (1,0,-1)	Sgn(x)

Tabla 3.3. Funciones matemáticas de Visual Basic

Todas las funciones trigonométricas en Basic están configuradas en radianes, por lo que se debe tener en cuenta transformar los grados a ésta unidad antes de realizar los cálculos, y realizar la conversión inversa antes de dar los resultados, si es que se desea trabajar en grados, que es lo más común en la mayoría de los casos en robótica.

Por lo anterior, se puede ver que Visual Basic no incluye todas las funciones trigonométricas ni las trigonométricas parabólicas, dado que éstas son derivadas de las tres fundamentales arriba mencionadas, por lo que si se desea utilizar alguna función no incluida, se debe de desarrollar el algoritmo equivalente.

3.5. Controles Gráficos

Básicamente Visual Basic trabaja con los controles gráficos siguientes: *Line*, *shape, pset y circle*. Así como de los controles *Image* y *Picture Box* para el tratamiento de gráficos.

Por ejemplo, en este proyecto se utilizan las *Picture Box*, o cuadros de imagen para representar esquemáticamente el manipulador, mientras que la representación de este se lleva a cabo mediante los comandos *Line*. Finalmente en las gráficas se hace uso del *Pset*. A continuación se dará una breve

explicación de cómo se utilizan estos comandos, así como *Circle* y Shape, este ultimo no utilizado en el presente proyecto.

3.5.1. Control Line

Sin duda el control más elemental, pues es el que tiene menor número de propiedades, y no reconoce ningún evento, es solamente decorativo o ilustrativo, pero al ser creado en tiempo de ejecución puede tener una utilidad grande.

En tiempo de ejecución es posible dibujar líneas en un formulario o en un *Picture Box*, especificando sus coordenadas, XY iniciales y finales, el ancho de la misma, estilo y color, así como asignarle un nombre.

Ejemplo:

```
Picture1.Line (0, 0)-(-x1 * 130 / 210, y1 * 130 / 210), vbRed, BF
```

Donde *Picture1* es el nombre del objeto donde será dibujada la línea, siguen a continuación las coordenadas, x_1 y_1, y x_2 y_2. Hasta aquí son lo parámetros obligatorios, lo que sigue es opcional, y es el color, en este caso el rojo definido por Visual Basic®, vbRed, y las componentes que siguen son para crear una caja en vez de una linea, B de Box, y seleccionar que sea rellenada con el color, F de Fill.

3.5.2 Control Shape

Este control se diferencia de Line porque admite formas geométricas tales como cuadrado (***Square***), rectángulo (***Rectangle***), círculo (***Circle***), elipse (***Oval***), cuadrado redondeado (***Rounded Square***) y rectángulo redondeado (***Rounded Rectangle***).

Además de las propiedades correspondientes al tamaño y posición, las propiedades más interesantes del control ***Shape*** son las siguientes: *BackColor*, *BackStyle*, *BorderColor*, *BorderStyle*, *BorderWidth*, *FillColor*, *FillStyle*, *DrawMode*.

Un control ***Shape*** puede estar visible o no (***Visible***), y existe la propiedad ***Index***, que permite crear *arrays* de ***Shapes***.

3.5.3 Control Image

El control *Image* es un contenedor de gráficos *bitmap*, *iconos*, *metafile*, *enhanced metafile*, *GIF* y *JPEG*. Este control admite ya una amplia colección de eventos, por lo que es ya un control con un papel mucho más activo que los anteriores.

Las propiedades más propias e importantes de este control son las propiedades *Picture* y *Stretch*. La propiedad *Picture* sirve para relacionar este control con el fichero que contiene el gráfico que se desea representar, a través del cuadro de diálogo *Load Picture* que permite elegir el fichero a insertar. El fichero deberá ser de uno de los tipos admitidos. Según el fichero elegido, la propiedad *Picture* tendrá uno de los tres valores siguientes: *icon* (ficheros *cur*, *ico*), *bitmap* (*bmp*, *gif*, *jpg*) o *metafile* (*wmf*, *emf*).

La propiedad *Stretch* indica cómo se comporta el control *Image* al introducir en él el contenido del fichero gráfico. Por defecto, cuando se crea un control *Image* arrastrando en el formulario con el ratón esta propiedad tiene el valor *False*. Estando la propiedad *Stretch* en *False* el tamaño del control se ajusta al tamaño del *bitmap* o del *metafile* que se introduce en dicho control.

Por el contrario, si dicha propiedad está en *True* el gráfico que proviene del fichero se adapta al tamaño de control. Se puede tratar de modificar el tamaño del gráfico en modo de diseño (con el ratón o cambiando las propiedades de tamaño del control). Si el gráfico es un *bitmap* y la propiedad *Stretch* está en *False*, el tamaño de la imagen no cambia aunque cambie el del control (quedando en la esquina superior izquierda si el control se hace más grande, o quedando parcialmente oculta si alguna de las dimensiones del control se hace más pequeña que la del *bitmap*. Si la propiedad *Stretch* está en *True*, el *bitmap* se adapta al tamaño del control y su tamaño se cambia con el de éste. Los gráficos *metafile* siempre se pueden cambiar de tamaño en modo de diseño, tanto si *Stretch* está en *True* como si está en *False*.

Existen otras formas de cargar un gráfico en un control *Image*, además de utilizar la propiedad *Picture* en modo de diseño, como se ha visto anteriormente. Una segunda forma, utilizable también en modo de diseño, es hacer *Copy* y *Paste* a partir de un gráfico contenido en otra aplicación como *Paint Shop Pro* o *Excel.*

En modo de ejecución se puede copiar el contenido de un control *Image* en otro control del mismo tipo por medio de una sentencia de asignación en la forma:

```
imgCuadro.picture = imgCaja.picture
```

y se puede también cargar una imagen de un fichero utilizando el procedimiento *LoadPicture*, por ejemplo en la forma siguiente (habrá que estar seguro de que existe el fichero):

```
imgCuadro.picture = LoadPicture("G:\graficos\pc.wmf")
```

Aunque el control *Image* admite algunos eventos (*Click*, *DblClick*, *DragDrop*, *DragOver*, *MouseUp*, *MouseDown*, *MouseMove*), sus posibilidades son también limitadas. Por la forma en que se dibuja, el control *Image* no puede estar sobre otro control, como por ejemplo un botón.

Además sólo puede contener gráficos, y este control no puede obtener el *focus* y por tanto no puede responder a acciones desde el teclado. El control *PictureBox*, que se verá a continuación, resuelve estas limitaciones aunque presenta la desventaja de ser más lento en dibujar que el control *Image*.

3.5.4 Control PictureBox

Este es el control gráfico más potente y general de Visual Basic®. Se trata de una especie de *formulario reducido*, pues puede contener imágenes y otros tipos de controles tales como botones, shapes, labels, cajas de texto, etc.

Con respecto a los *bitmaps*, el control *PictureBox* se comporta de modo diferente que el control *Image*. El control *PictureBox* no tiene propiedad *Stretch*, con lo cual al cargar un icono o un bitmap siempre aparecen con su tamaño natural. Sin embargo el control *PictureBox* tiene la propiedad *AutoSize*, que por defecto está

en *False*. Cuando se carga un *bitmap* con *AutoSize* en *False* el gráfico aparece en la esquina superior izquierda del control; sin embargo, si *AutoSize* está en *True* el control *PictureBox* adapta su tamaño al del *bitmap* que es cargado. La Figura 6.5 muestra los resultados de introducir un icono en un control *Image* (*Stretch*: *False* y *True*) y en un control *PictureBox* (*AutoSize*: *False* y *True*).

En el control *Image* se cargan con su verdadero tamaño si la propiedad *Stretch* es *False*, mientras que se adaptan al tamaño del control si dicha propiedad es *True*. Con el control *PictureBox* se adaptan al tamaño del control si *AutoSize* es *False*, mientras que se cargan con su propio tamaño si es *AutoSize* es *True*.

En el control *PictureBox* son importantes las cuatro propiedades relacionadas con el color: *BackColor*, *ForeColor*, *FillColor* y *FillStyle*. La propiedad *BackColor* controla el color de fondo del control. La propiedad *ForeColor* controla el color del texto que se escribe en el control (con el método *Print*, por ejemplo, como luego se verá). Las propiedades *FillColor* y *FillStyle* no afectan directamente al control sino a los elementos gráficos que se dibujen sobre él con métodos tales como *Line* y *Circle*, que se verán a continuación. *FillStyle* determina el tipo de relleno o *pattern* (líneas horizontales, verticales, inclinadas, cruzadas, ...), mientras que *FillColor* determina el color de estas líneas del relleno.

3.6. Archivos y entrada/salida de datos

Ahora se van a describir varias formas de introducir información en el programa, así como de obtener resultados mediante escritura en un fichero. Se va a presentar una nueva forma interactiva de comunicarse con el usuario, como son las cajas de diálogo *MsgBox* e *InputBox*. Particular interés tiene la lectura y escritura de datos en el disco, lo cual es necesario tanto cuando el volumen de información es muy importante (la memoria RAM está siempre más limitada que el espacio en disco), como cuando se desea que los datos no desaparezcan al terminar la ejecución del programa, como simulaciones y ficheros de configuración en nuestro caso.

3.6.1 Cajas de diálogo inputbox y msgbox

Estas cajas de diálogo son similares a las que se utilizan en muchas aplicaciones de *Windows*. La caja de mensajes o *MsgBox* abre una ventana a través de la cual se envía un mensaje al usuario y se le pide una respuesta, por ejemplo en forma de pulsar un botón *O.K./Cancel*, o *Yes/No*. Este tipo de mensajes son lo utilizamos para enterar al usuario de que se ha cometido un error cuando intentaba programar una posición no válida en nuestro software.

La caja de diálogo *InputBox* pide al usuario que teclee una frase, pero esta función no fue utilizada en nuestra interfaz gráfica. La forma general de la función *MsgBox* es la siguiente:

```
respuesta = MsgBox("texto para el usuario", tiposBotones, "titulo")
```

donde *respuesta* es la variable donde se almacena el valor de retorno, que es un número indicativo del botón pulsado por el usuario. La constante simbólica que representa el valor de retorno indica claramente el botón pulsado.

El parámetro *tiposBotones* es un entero que indica la combinación de botones deseada por el usuario. También en este caso la constante simbólica correspondiente es suficientemente explícita. Si este argumento se omite se muestra sólo el botón *O.K.*

El parámetro *titulo* contiene un texto que aparece como título de la ventana; si se omite, se muestra en su lugar el nombre de la aplicación.

3.6.2. Método print

Este método permite escribir texto en *formularios*, cajas *pictureBox* y en un objeto llamado *Printer* o impresora

Características generales

1. El método *Print* recibe como datos una *lista de variables y/o cadenas de caracteres*. Las cadenas son impresas y las variables se sustituyen por su valor.
2. Hay dos tipos básicos de *separadores* para los elementos de la lista. El carácter *punto y coma* (;) hace que se escriba inmediatamente a continuación de lo anterior. La *coma* (,) hace que se vaya al comienzo de la siguiente *área de salida*. Con letra de paso constante como la *Courier* las áreas de salida empiezan cada 14 caracteres, es decir en las columnas 1, 15, 29, etc. Con letras de paso variable esto se hace sólo de modo aproximado.
3. Las constantes numéricas positivas van precedidas por un espacio en blanco y separadas entre sí por otro espacio en blanco. Si son negativas el segundo espacio es ocupado por el signo menos (-).
4. El tipo y tamaño de letra que se utiliza depende de la propiedad *Font* del formulario, objeto *PictureBox* u objeto *Printer* en que se esté escribiendo.

Existen otros separadores tales como *Tab(n)* y *Spc(n)*. El primero de ellos lleva el punto de inserción de texto a la columna *n*, mientras que el segundo deja *n* espacios en blanco antes de seguir escribiendo. *Tab* sin argumento equivale a la coma (,). Estos espaciadores se utilizan en combinación con el punto y coma (;), para separarlos de los demás argumentos.

Por defecto, la salida de cada método *Print* se escribe en una nueva línea, pero si se coloca un punto y coma al final de un método *Print*, el resultado del siguiente *Print* se escribe en la misma línea.

Puede controlarse el lugar del formulario o control donde se imprime la salida del método *Print*. Esta salida se imprime en el lugar indicado por las propiedades *CurrentX* y *CurrentY* del formulario o control donde se imprime. Cambiando estas propiedades se modifica el lugar de impresión, que por defecto es la esquina superior izquierda. Existen unas funciones llamadas *TextWidth(string)* y *TextHeight(string)* que devuelven la anchura y la altura de una cadena de

caracteres pasada como argumento. Estas funciones pueden ayudar a calcular los valores más adecuados para las propiedades *CurrentX* y *CurrentY*.

3.6.3. Tipos de ficheros

Tanto en *Windows* como en *Visual Basic 6.0* existen, principalmente, dos tipos de archivos:

1. *Ficheros ASCII* o ficheros de texto. Contienen caracteres codificados según el código ASCII y se pueden leer con cualquier editor de texto como *Notepad*. Suelen tener extensión **.txt* o **.bat,* pero también otras como **.m* para los programas de *Matlab*, **.c* para los ficheros fuente de C, **.cpp* para los ficheros fuente de C++ y **.java* para los de *Java*.
2. *Ficheros binarios*: Son ficheros imagen de los datos o programas tal como están en la memoria del ordenador. No son legibles directamente por el usuario. Tienen la ventaja de que ocupan menos espacio en disco y que no se pierde tiempo y precisión cambiándolos a formato ASCII al escribirlos y al leerlos en el disco.

Con Visual Basic 6.0 se pueden leer tanto ficheros ASCII como ficheros binarios. Además el acceso a un fichero puede ser de tres formas principales.

1. *Acceso secuencial.* Se leen y escriben los datos como si se tratara de un libro: siempre a continuación del anterior y sin posibilidad de volver atrás o saltar datos. Si se quiere acceder a un dato que está hacia la mitad de un fichero, habrá que pasar primero por todos los datos anteriores. Los ficheros de texto tienen acceso secuencial.
2. *Acceso aleatorio (random)*: Permiten acceder directamente a un dato sin tener que pasar por todos los demás, y pueden acceder a la información en cualquier orden. Tienen la limitación de que los datos están almacenados en unas unidades o bloques que se llaman *registros*, y que todos los registros que se almacenan en un fichero deben ser del mismo tamaño. Los ficheros de acceso aleatorio son ficheros binarios.
3. *Acceso binario.* Son como los de acceso aleatorio, pero el acceso no se hace por *registros* sino por *bytes*. Antes de poder leer o escribir en un fichero hay que abrirlo por medio de la sentencia *Open*. En esta sentencia

hay que especificar qué tipo de acceso se desea tener, distinguiendo también si es para lectura (*input*), escritura (*output*) o escritura añadida (*append*).

En nuestro proyecto solo nos interesa los ficheros del tipo ASCIII secuenciales, debido a que es una secuencia la que deseamos programar y para efectos de simulación usaremos ficheros secuenciales también. Daremos una explicación más detallada de los mismos.

3.6.7.1 Ficheros secuenciales

Para poder leer o escribir en un fichero antes debe ser abierto con la sentencia *Open*, cuya forma general es la siguiente:

Open filename For modo As # fileNo

donde:

- *filename* es el nombre del fichero a abrir. Será una variable *string* o un nombre entre dobles comillas (" ").
- *modo* Para acceso secuencial existen tres posibilidades: *Input* para leer, *Output* para escribir al comienzo de un fichero y *Append* para escribir al final de un fichero ya existente. Si se intenta abrir en modo *Input* un fichero que no existe, se produce un error. Si se abre para escritura en modo *Output* un fichero que no existe se crea, y si ya existía se borra su contenido y se comienza a escribir desde el principio. El modo *Append* es similar al modo *Output*, pero respeta siempre el contenido previo del fichero escribiendo a continuación de lo último que haya sido escrito anteriormente. *fileNo* es un número entero (o una variable con un valor entero) que se asigna a cada fichero que se abre.

En todas las operaciones sucesivas de lectura y/o escritura se hará referencia a este fichero por medio de este número. No puede haber dos ficheros abiertos con el mismo número.

Después de terminar de leer o escribir en un fichero hay que cerrarlo. Para ello, se utilizara el comando *Close*, que tiene la siguiente forma:

Close # fileNo

- *fileNo* es el número que se la había asignado al abrirlo con la instrucción *Open*.

3.6.3. Sentencia Input

Existen varias formas de leer en un fichero de acceso secuencial. Por ejemplo, para leer el valor de una o más variables se utiliza la sentencia *Input*:

Input # fileNo, varName1, varName2, varName3, ...

- *fileNo* es el número asignado al archivo al abrirlo y
- *varName1, varName2, ...* son los nombres de las variables donde se guardarán los valores leídos en el fichero.

Debe haber una correspondencia entre el orden y los tipos de las variables en la lista, con los datos almacenados en el fichero.

3.6.3.1 Función Print

Para escribir el valor de unas ciertas variables en un fichero previamente abierto en modo *Output* o *Append* se utiliza la instrucción *Print* #, que tiene la siguiente forma:

Print #fileNo, var1, var2, var2, ...

donde ***var1***, ***var2***,... pueden ser variables, expresiones que dan un resultado numérico o alfanumérico, o cadenas de caracteres entre dobles comillas, tales como *"El valor de x es..."*.

En base a estas funciones ya solo es cuestión de desarrollar en el editor de código el algoritmo que nos escriba los ficheros necesarios para la programación de nuestro manipulador. Para tal se debe de tomar en cuenta las características que debe contener los archivos, que se mencionaron en el capítulo del Pic 16f84.

3.2 Programa MPLAB

Introducción

Cundo se desea trabajar con un microcontrolador en específico, se deben tener en cuenta ciertas cuestiones, desde los costos del Hardware, hasta la forma en la que se desarrollará la lógica para éste. La forma más practica de programar un microcontrolador, consiste en cargar en la PC, un software de desarrollo donde probaremos el programa, y posteriormente grabarlo en la memoria del microcontrolador para ver que los resultados sean satisfactorios en la práctica.

La compañía Microchip® provee gratuitamente en Internet un programa de desarrollo para cualquiera de sus microcontroladores, en el cual pueden ser escritos los programas, depurados y simulados. Esta herramienta es llamada MPLAB IDE, es decir un Entorno de Desarrollo Integrado.

MPLAB es un programa diseñado para correr sobre Windows® desde la versión 3.11, al igual que el resto del software que empleamos en nuestro proyecto, con excepción de Mechanical Desktop®. Es cierto que el código fuente puede ser escrito en cualquier editor de texto, respetando las 4 columnas de parámetros para la creación y edición de este, pero para la correcta codificación, es indispensable el uso de una herramienta como MPLAB, sobre todo si es el mismo fabricante del chip quien la proporciona. El programa es en sí muy extenso debido a la gran cantidad de modelos que puede programar y simular, como se menciono antes, es un entorno que es capaz de manejar varios proyectos con diferentes modelos de controladores y archivos fuente, pero para nuestro proyecto sólo será necesario conocer las funciones indispensables como la creación de proyectos, de códigos y la codificación al lenguaje hexadecimal de los mismos.

MPLAB como un medio de desarrollo integral

MPLAB es un programa fácil de aprender a usar, y para los familiarizados con el ambiente Windows® será una tarea todavía más sencilla. Las principales funciones que nos brinda son las siguientes:

- Crear y editar códigos fuente
- Agrupar los archivos en proyectos
- Depurar el código fuente
- Depurar la ejecución de los programas a través del simulador o emulador.

Veremos más adelante cada una de las funciones y como se ejecutan dentro de MPLAB, pero por el momento valdría la pena mencionar algunas herramientas que integra MPLAB así como otras que se pueden adquirir por separado directamente de Microchip®.

- Project Manager: esta herramienta de MPLAB nos permite crear proyectos en los cuales se agrupen diferentes archivos de códigos fuente, en él se especifica cual será el archivo destino de cada código, una vez codificado lo carga en el simulador para que podamos probarlo en pantalla.
- Editor: que nos permite editar nuestros códigos fuente, es muy similar a un editor de texto sencillo como Wordpad®
- Depurador en circuito (ICD): Esta herramienta permite correr los programas ya montados los controladores en el circuito, pero desafortunadamente se debe adquirir por separado y tiene un costo, además de adquirir el hardware correspondiente..
- MPLAB SIM: Esté es nuestro simulador por software que nos permite ver la evolución del programa, registros y memoria.
- MPASM: Esta herramienta es el ensamblador, se le verá correr en segundo plano cada que reconstruimos un proyecto, es la parte de MPLAB que codifica el código fuente que tenemos como codigo ASCII, a Hexadecimal.

Aparte de las herramientas proporcionadas por Microchip® existen en Internet herramientas desarrolladas por terceros. En el siguiente apartado hablaremos de una de ellas de vital importancia en este proyecto.

Ejecución de MPLAB

La ejecución de MPLAB se lleva a cabo desde el menú Inicio, programas y MPLAB.EXE. La primera pantalla que veremos al iniciar será la mostrada en la figura 3.2.1

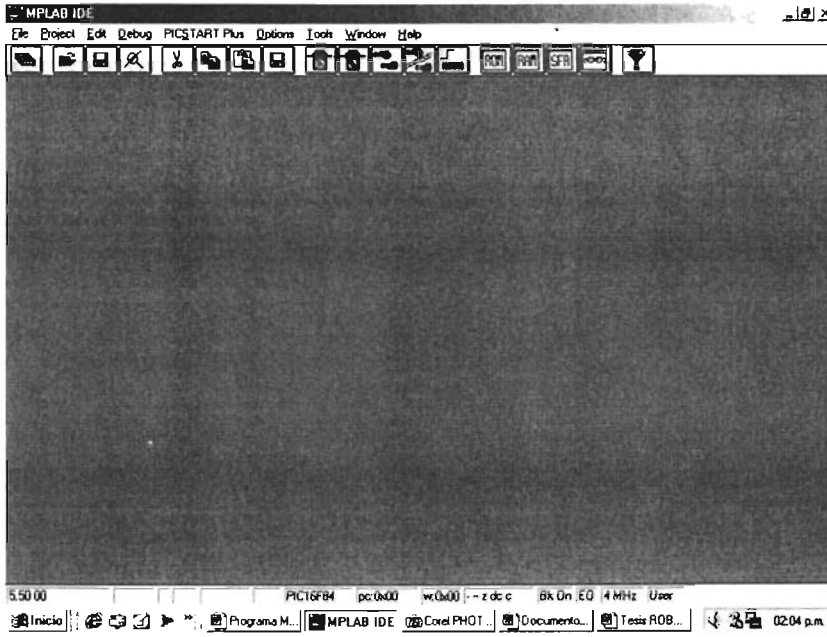

Figura 3.9. Interfase Principal de MPLAB®

En la figura 3.9 podemos apreciar las partes principales del programa que son: Barra de Menú, Barra de Herramienta, Espacio de Trabajo y Barra de Estado. Es en esta última donde aparecerá la información sobre nuestro proyecto, tipo de controlador, tipo de oscilador, etc.

Seleccionando Simulador y Procesador

Lo primero que haremos será seleccionar el modo de desarrollo en el menú *Options/Development Mode*. Por default se encuentra en modo "Editor Only", el cual no permite la simulación del código, así que conviene cambiar a modo "MPLAB SIM" si nos encontramos desarrollando una aplicación. En el caso de nuestro proyecto el código fuente proporcionado por el programa de Robot SCARA se encuentra libre de errores, por lo cual podemos dejar la opción en modo editor. En esta pestaña seleccionamos el Controlador a usar, en nuestro caso el PIC16F84A, como lo podemos apreciar en la figura 3.10

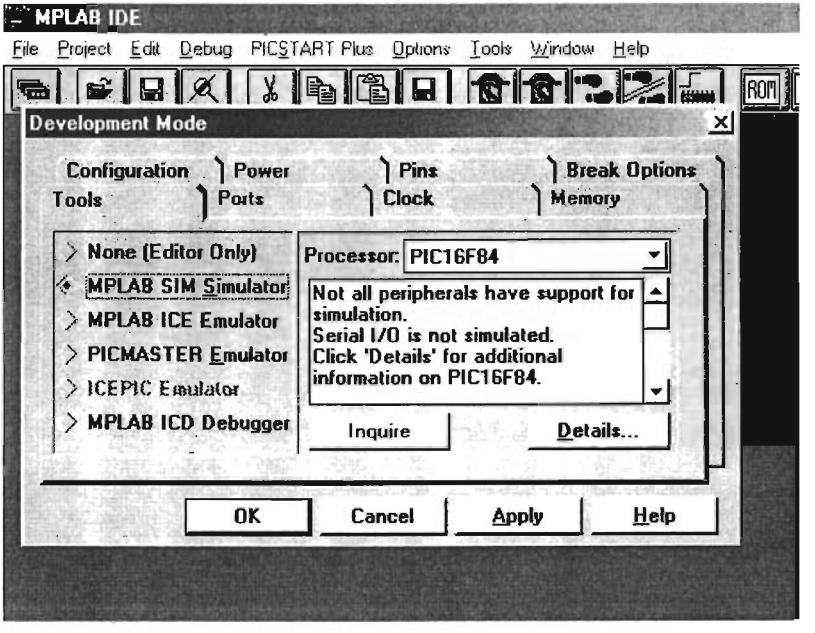

Figura 3.10. Menu para Modo de Desarrollo de Proyecto

En la pestaña Clock seleccionaremos el tipo de oscilador que empleamos en nuestro proyecto, que será un XT de 4 Mhz como se aprecia en la figura 3.11.

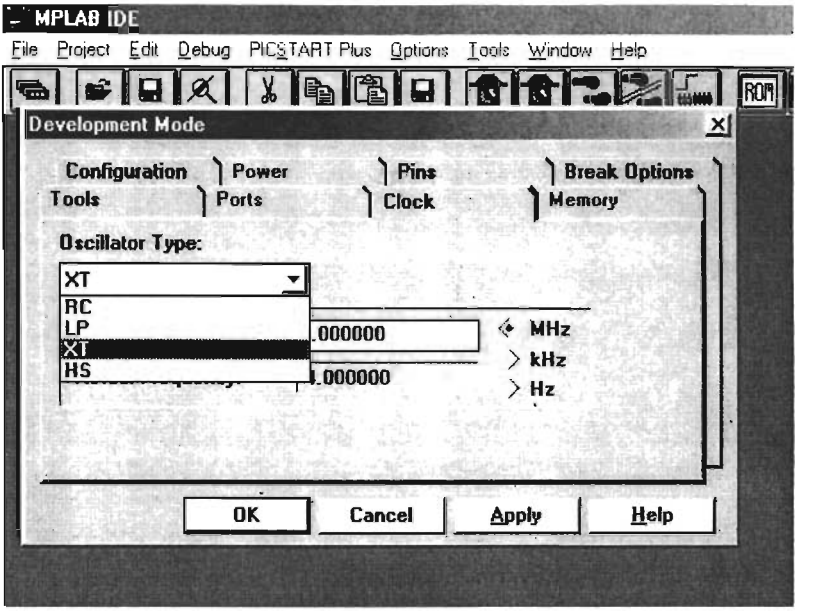

Figura 3.11. Selección de Oscilador

Las demás opciones deben dejarse en su estado original, con estos cambios es suficiente para nuestro proyecto.

Creando un nuevo proyecto

El siguiente pasó es crear un proyecto y un nodo correspondiente, ya que debe existir un archivo de código fuente y una conexión con el archivo hexadecimal que será vaciado en el microcontrolador para que el ensamble sea correcto.

Para crear un nuevo proyecto seleccionaremos en el menú *File/New* y a continuación aparecerá el cuadro de dialogo mencionando que no existe ningún proyecto y si deseamos crear uno nuevo, seleccionamos "yes". Aparecerá el cuadro de dialogo mostrado en la figura 3.12.

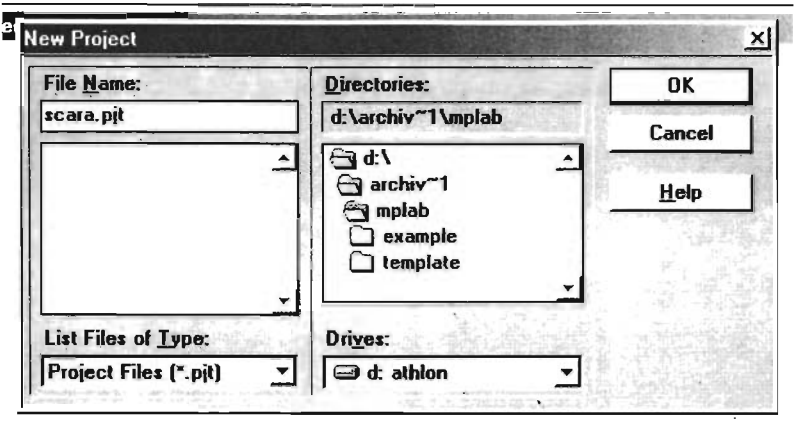

Figura 3.12. Cuadro de dialogo Nuevo Proyecto

Seleccionamos un nombre para nuestro proyecto y lo guardamos en la carpeta que nos sea más conveniente. A continuación aparecerá un cuadro de dialogo como el mostrado en la figura 3.13.

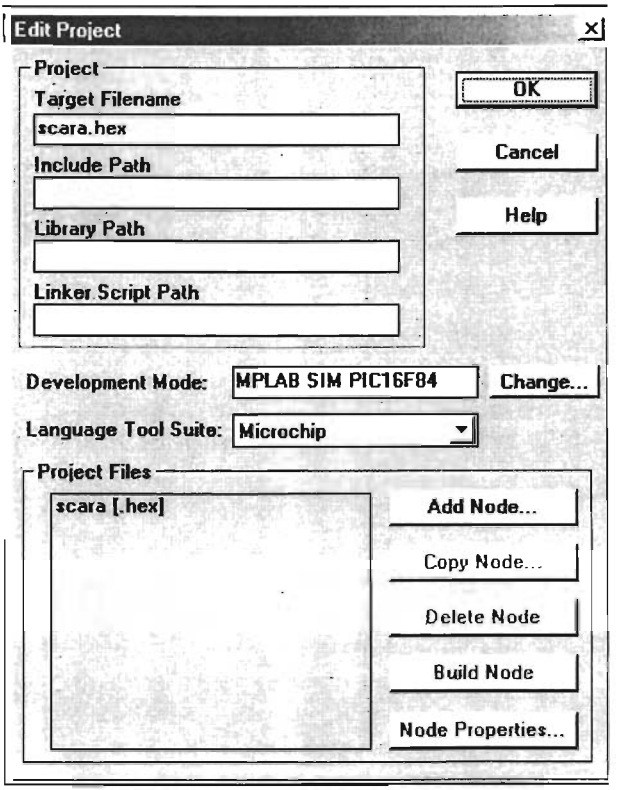

Figura 3.13. Cuadro de dialogo Edición de Proyecto.

Podemos notar que en este cuadro las opciones de Modo de Desarrollo y Lenguaje han sido llenadas con la información que proporcionamos cuando escogimos el modo de desarrollo en el menú *Options*. Debemos agregar un nodo con el botón *Add Node*, aparecerá la siguiente ventana mostrada en la figura 3.14

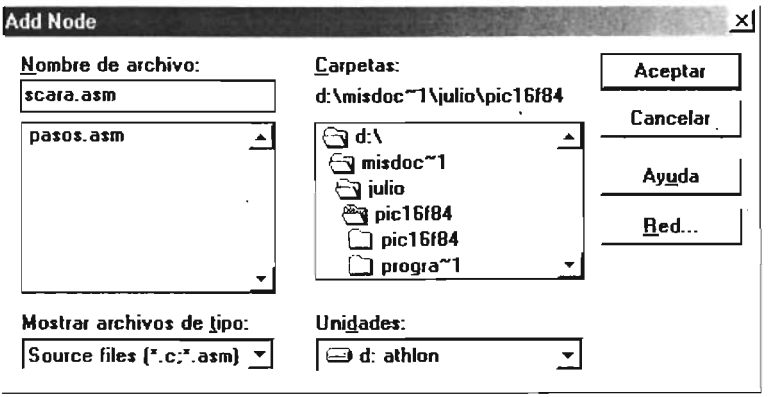

Figura 3.14. Cuadro de diálogo Add Node

En este cuadro de dialogo ubicaremos nuestro archivo hexadecimal, que puede llevar el mismo nombre que el del proyecto para evitar confusiones. Seleccionamos un nombre adecuado y damos clic en OK.

En la ventana *Projet Files* seleccionamos nuestro nodo: *scara.hex* y el botón *Node Properties* será accesible. Damos un clic en el y aparecerá la ventana mostrada en la figura 3.15.

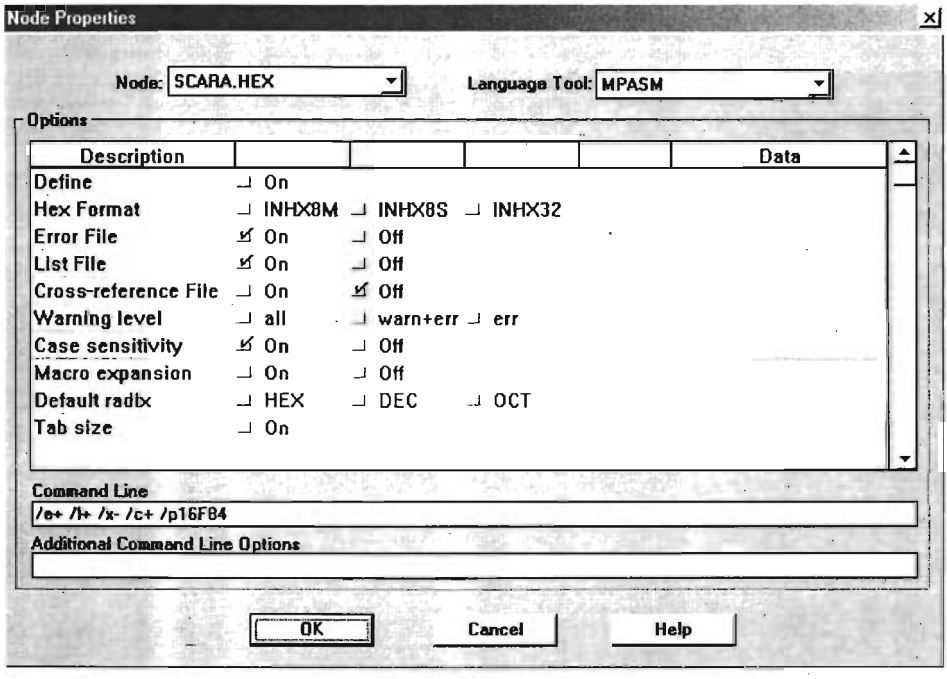

Figura 3.15. Propiedades de Nodo.

En esta ventana se muestra la configuración con la que trabajará nuestro ensamblador cuando sea invocado por el MPLAB. Por el momento dejaremos las opciones como se encuentran, pero es bueno tenerlas en cuenta para futuros cambios que pudiera requerir el proyecto.

El siguiente paso es crear el archivo de código fuente, cuando creamos nuestro proyecto en el espacio de trabajo apareció una ventana con el nombre de archivo *Untitled*, en ella podemos comenzar a escribir nuestro código fuente, en nuestro caso podemos cerrarla y abrir uno de los archivos creados por nuestro software para programación y simulación, del cual daremos detalle en páginas posteriores, o copiar el contenido de este y pegarlo. Una vez que este listo debemos guardar el archivo pero con el mismo nombre que dimos a nuestro nodo, seleccionamos el menú *File/Save as*. En nuestro caso será "*scara.asm*" como podemos ver en la figura 3.16.

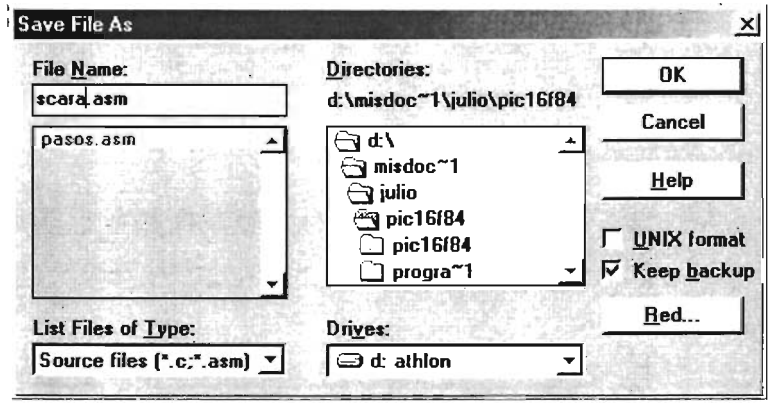

Figura 3.16. Cuadro de diálogo Guardar Archivo Como

Si el código fuente se encuentra listo y los nodos correctamente guardados, procedemos a ejecutar el ensamble del programa para crear nuestro archivo hexadecimal que será vaciado en nuestros microcontroladores. Ejecutamos en el menú *Project/Build All*, aparecerá una pantalla indicando los resultados de el ensamble y en el caso de que exista un error aparecerá en estos resultados las líneas que contienen el error y tipo de error. También contienen mensajes de advertencia o comentarios.

Una vez creado el archivo en formato hexadecimal este podrá ser vaciado al controlador por cualquiera de los software proporcionados por terceros. Este archivo se encontrará siempre en la misma carpeta donde creamos el proyecto y se encuentra el nodo Para nuestro proyecto utilizamos el Pic-Prog, del cual se darán más detalles posteriormente.

Mplab es una herramienta poderosa en desarrollo de programas para microcontroladores de la marca Microchip®, pero los alcances de esta introducción no abarcarán más allá de las funciones básicas, debido a que para la puesta en marcha de nuestro proyecto no es necesario. Cabe mencionar que en el desarrollo de los primeros prototipos de programa fue de gran utilidad debido a la cantidad de herramientas de simulación, como visores de registros y memoria.

3.3 El software Pic-Prog.

En este apartado se describirá el funcionamiento básico y características de este software pequeño pero de gran utilidad.

Pic-Prog es un software creado por un grupo argentino dedicado a proyectos de robótica, que inclusive dispone de una página en la Web dedicada al tema. Aunque hay que aclarar que este sitio esta dirigido a los aficionados al tema, por lo cual dispone de algunas herramientas dentro del campo de la electrónica, pero no es posible encontrar en el información de temas más serios como las teorías de control, por ejemplo. Por otra parte, la principal ventaja de este programa es que puede conseguirse de forma gratuita. Su única función es la de grabar las instrucciones en la memoria del PIC a partir del archivo hexadecimal que creamos con ayuda de MPLAB®. En la figura 3.17 podemos apreciar su interfaz típica.

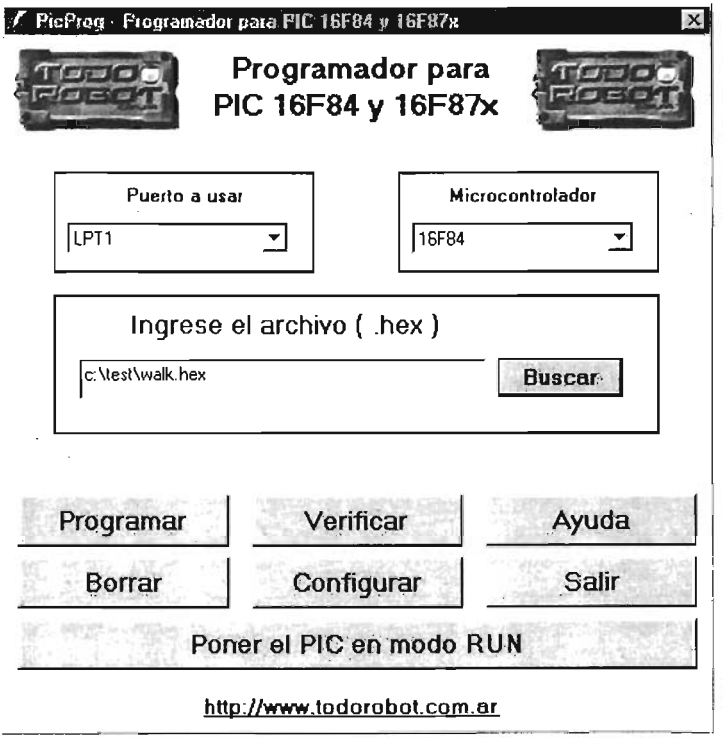

Figura 3.17. Interfaz del Pic-Prog.

En la figura 3.2.9 podemos apreciar las partes que lo integran. El puerto a usar, donde se conecta el Pic o el circuito para programarlo, El tipo de microcontrolador, que puede dejarse en automático, el archivo hexadecimal que deseamos integrar y los botones con las diversas funciones. Al final podemos apreciar la dirección Web donde fue obtenido, y donde en caso de alguna contingencia con este software podemos encontrar ayuda.

Algunas de las características por las que agradó este programa son las siguientes:

- Autodetección del Pic a programar
- El Pic-Prog checa cada ubicación de memoria y solamente escribe aquellas direcciones que son necesarias, prolongado la vida útil del chip y programando más rápidamente.
- Puede efectuar el borrado completo del Pic, por si el programa contiene protección de código.
- Puede verificar que el programa se haya cargado correctamente.
- Dispone de una utilidad que permite poner en modo *run* el Pic, esto es de gran ayuda cuando se experimenta con un programa debido a que desde la pantalla se puede programar y correr las instrucciones sin necesidad de recurrir al circuito cada vez que deseamos probar la nueva programación. (Solamente cuando se utiliza un Pic.)
- La función Configurar es de gran utilidad, debido a que si el archivo no contiene información sobre los bits de configuración, aquí puede asignársele la configuración deseada o, incluso si el archivo contiene dicha información, podemos cambiarla con esta opción.

Este programa es capaz de programar los PIC16F84A así como toda la familia de PIC16F87x. Es posible adquirir programas con tarjetas grabadoras pero además de oscilar su costo en los $500 pesos solamente envían las instrucciones al controlador y no realizan ningún tipo de verificación.

Por lo tanto será muy sencilla la programación del robot contando con los elementos necesarios. Teniendo nuestros archivos hexadecimales procederemos a grabar el archivo Scara1.hex al microcontrolador 1 con la ayuda del Pic-Prog y de igual forma el Scara2.hex al microcontrolador 2.

Claro que existen algunas consideraciones que se deben tener en cuenta para el uso de este software.

- El circuito debe de estar en modo programación siempre que el Pic sea puesto en modo Run desde software, para evitar daños a la tarjeta.
- No en todos los puertos paralelos funciona, ya que existen algunas computadoras anteriores que no disponen de la potencia necesaria en el puerto paralelo para lograr una comunicación eficaz con el circuito programador.

Teniendo las debidas precauciones para proteger nuestro hardware, podemos utilizar este pequeño software con gran confiabilidad.

3.4. Software para la simulación y programación

Empleando las teorías matemáticas y todos los conocimientos adquiridos sobre programación en Visual Basic y microcontroladores procedemos a la creación de un software que nos permita simular los movimientos y programar posteriormente el brazo manipulador con la rutina seleccionada y simulada con anterioridad. Como ya se ha visto en el capítulo anterior, en la figura 3.18 podemos apreciar la interfaz principal de nuestro software propio de programación.

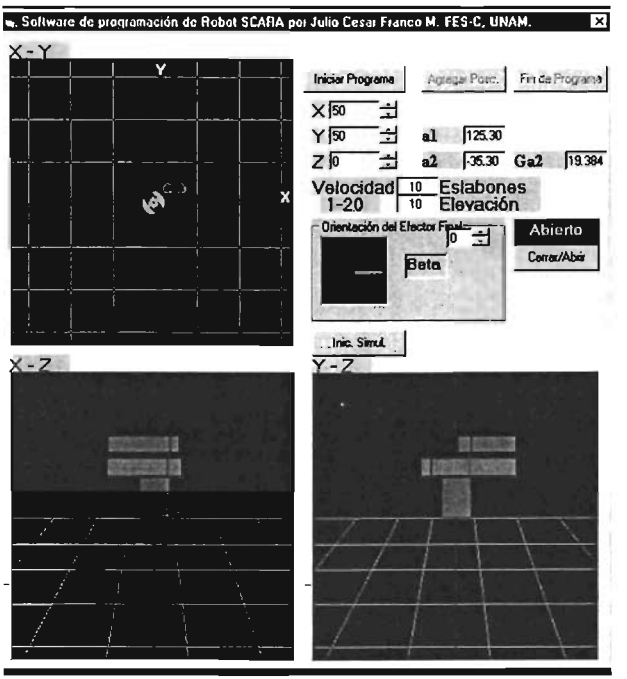

Figura 3.18. Interfaz principal del programa SCARA.

La imagen de la misma y los colores pueden variar dependiendo de la versión y configuración del sistema operativo.

A continuación veremos un poco del código fuente para observar con mas detalle donde se emplean las teorías y conocimientos expuestos en los capítulos anteriores.

Lo primero que haremos en Visual Basic® después de tener nuestra ventana con lo elementos necesarios, será definir las variables que se usaran en todos los módulos, esto lo hacemos en el módulo"General".

```
Public step, flag, acel10, acel11, acel20, acel21, dinamica As Integer
Public t1ant, t2ant, REL1, REL2, REL3, REL4, simxa, ... etc. As Double
```

En este caso definimos en primera instancia la variables que serán enteros, posteriormente, definimos las variables de doble precisión. Lo que sigue es la inicialización del programa para el PIC. Todo el proceso abajo descrito comienza con la pulsación del Botón "Iniciar Programa" que internamente es llamado Command1

```
Private Sub Command1_Click() ' Inicializacion de programas en Pic16f84
Open "Prog1.asm" For Output As #1 ' Creamos un nuevo archivo
Print #1, " ", "List", "p=16f84A"
Print #1, " ", "Include", "<p16f84A.inc>"
Print #1, " ", "   __CONFIG  _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC"
Print #1, "POS1", "equ", "0x0c"
Print #1, "POS2", "equ", "0x0d"
Print #1, "DESACEL", "equ", "0x0e"
Print #1, "DELAY1", "equ", "0x0f"
Print #1, "PASOSG", "equ", "0x10"
Print #1, "DELAYV", "equ", "0x11"
Print #1, "ACEL", "equ", "0x12"
Print #1, "PASOS", "equ", "0x13"

Print #1, "", "ORG 0x000"
Print #1, "", "MOVLW", "B'00000101'"
Print #1, "", "TRIS", "PORTA"    ' configuración de los puertos
Print #1, "", "MOVLW", "B'00000000'"
Print #1, "", "TRIS", "PORTB"
Print #1, "", "MOVLW", "B'00000010'"
Print #1, "", "OPTION"              ' Inicialización del registro Option
Print #1, "", "CLRF", "STATUS"
Print #1, "", "MOVLW", "B'11001100'"
Print #1, "", "MOVWF", "PORTB"
Print #1, "", "MOVLW", "B'0'"
Print #1, "", "MOVWF", "POS1"
Print #1, "", "MOVWF", "POS2"
Print #1, "", "BSF", "PORTA,1"
Print #1, "", "GOTO", "INICIO" ' Salto al inicio del programa
```

En el ejemplo anterior alcanzamos a apreciar la inicialización completa del PIC, definiendo sus parámetros, variables definidas por usuario, la configuración de los puertos y la activación doble de los dos motores en sus bobinas iniciales. El resto del código de este apartado se omitirá, debido a la longitud del mismo. Al final de este modulo, cerramos con las siguientes lineas.

```
Close #1  ' se cierra el archivo de inicializado del PIC"
Command1.Enabled = False ' Activacion y desactivación de los botones "Command"
Command2.Enabled = True
Command3.Enabled = True
Open "sim.jul" For Output As #1
Open "simpp.jul" For Output As #2  ' Abrimos dos nuevos archivos para datos auxiliares
Print #2, textox, textoy, textoz, Text1(3).Text, vel(0).Text, Ang(0).Text, Ang(2).Text
Print #1, textox, textoy, textoz, Text1(3).Text
simxa = textox ' Actualizamos las posiciones para la simulación
simya = textoy
simza = textoz
simefa = Text1(3).Text
posalfa1 = Val(Ang(0).Text)
posalfa2 = Val(Ang(2).Text)
posalfa3 = Val(Ang(1).Text) - Val(Text1(3).Text)
Close #2 ' Cerramos los dos nuevos archivos.
Close #1
```

En el siguiente modulo se tratará que hacer cada vez que se pulse el botón "Agregar Posición" conocido como Command2.

```
Private Sub Command2_Click() 'Agregado de posiciones y movimientos
REL1 = 10 * 3.14 / 400
REL2 = (7.5 / 2) * (20 / 168) ' GRADOS/PASO A MEDIO PASO POR PIÑON/CORONA
REL3 = (7.5 / 2) * (1 / 8) 'IDEM
REL4 = (7.5 / 2) * (14 / 40)
```

Se calcula con respecto a la relación de engranes y al movimiento en grados del manipulador, para saber la cantidad de pasos que deberá de dar cada motor.

En caso de que el movimiento sea nulo, se mandara la rutina "STOP" del microcontrolador que no hace más que esperar a que el otro PIC termine su tarea y le envié la señal de sincronía.

```
Open "Prog1.asm" For Append As #1
Print #1, "", "CALL", "STOP"
Close #1

Open "Prog2.asm" For Append As #1
Print #1, "", "CALL", "STOP"
Close #1
```

Después se procede a calcular si el movimiento correra en un ciclo o dos del Microcontrolador, y el sentido que llevará.

```
Open "Prog2.asm" For Append As #1
 If pasos <> 0 And ciclo < 2 Then Print #1, "", "MOVLW", "D'"; Format(Velocidad); "'"
 If pasos <> 0 And ciclo < 2 Then Print #1, "", "MOVWF", "DELAYV"
 'If pasos = 0 Then Print #1, "", "CALL", "STOP"
 If pasos < 0 And ciclo = 1 Then Print #1, "", "MOVLW", "D'"; Format(-pasos - 40); "'"
 If pasos > 0 And ciclo = 1 Then Print #1, "", "MOVLW", "D'"; Format(pasos - 40); "'"
 If pasos < 0 And ciclo <> 1 Then Print #1, "", "MOVLW", "D'"; Format(-pasos); "'"
 If pasos > 0 And ciclo <> 1 Then Print #1, "", "MOVLW", "D'"; Format(pasos); "'"
 If pasos < 0 Then Print #1, "", "CALL", "DER1"
 If pasos > 0 Then Print #1, "", "CALL", "IZQ1"
Print #1, ""
Close #1
```

Empleando algunas banderas y los clásicos comandos de Basic, hacemos la selección correcta. También podría ser utilizada la función "Select Case" en esta sección.

Para el caso del movimiento del efector final se realiza una selección similar

```
' creacion del programa de efector final
If Pasos3 < 256 And Pasos3 > 0 Then pasos = Pasos3: If Pasos3 < 256 And Pasos3> 0 Then Pasos3 = 0
If Pasos3 > 255 Then pasos = 255: If Pasos3 > 255 Then Pasos3 = Pasos3 - 255
If Pasos3 < 0 And Pasos3 > -256 Then pasos = Pasos3: If Pasos3<0 And Pasos3>-256 Then Pasos3 = 0
If Pasos3 < -255 Then pasos = -255: If Pasos3 < -255 Then Pasos3 = Pasos3 + 255
```

Cuando terminamos de programar las posiciones se pulsa el botón "Fin de Programa" y la rutina asociada a este es

```
Private Sub Command3_Click() ' fin de programa
Open "Prog1.asm" For Append As #1
```

```
Print #1, "", "NOP"
Print #1, "", "GOTO", "$-1"
Print #1, "", "end"
Close #1
Open "Prog2.asm" For Append As #1
Print #1, "", "NOP"
Print #1, "", "GOTO", "$-1"
Print #1, "", "end"
Close #1
Command1.Enabled = False
Command2.Enabled = False
Command3.Enabled = False
```

En el siguiente modulo iniciamos la simulación de los movimientos, este modulo se encuentra integro, ya que es muy sencillo su funcionamiento, y lo complicado de esta sección se encuentra en el archivo creado más que en la ejecución de dicho archivo.

```
Private Sub Command4_Click() ' iniciar simulacion
Timer1.Enabled = True
t1ant = Val(Ang(0).Text) ' inicializacion de variable para analisis velocidad eslabon 1
Open "sim.jul" For Input As #1
Open "simppc.jul" For Input As #5
Input #1, simx, simy, simz, simef
Text1(0).Text = simx
Text1(1).Text = simy
Text1(2).Text = simz
Text1(3).Text = simef
   ' Modulo de coordenadas internas
   textox = simx
   textoy = simy
   textoz = simz
   textof = simef
Open "vel.jul" For Output As #2
Close #2

Open "temp.jul" For Output As #3
Print #3, simx, simy, simz, simef, Ang(0).Text, Ang(1).Text
Close #3

End Sub
```

Adicionalmente tenemos otros módulos como por ejemplo, el que nos abre el cuadro para generar las gráficas de aceleración y velocidad.

```
Private Sub Command5_Click()
Load Form2
Form2.Show
End Sub
```

Una vez abierta esta ventana, contiene sus propios módulos y variables que no se estudiarán aquí.

Otro modulo adicional es el que permite programar la apertura o cierre del efector.

```
Private Sub EFECTOR_Click()
If Eftexto.Text = "Abierto" Then
 Eftexto.Text = "Cerrado"
 Open "Prog1.asm" For Append As #1
 Print #1, "", "BSF", "PORTA3"
 Close #1
Elself Eftexto.Text = "Cerrado" Then
 Eftexto.Text = "Abierto"
 Open "Prog1.asm" For Append As #1
 Print #1, "", "BCF", "PORTA3"
 Close #1
End If
End Sub
```

Uno de los módulos más importantes, es donde se encuentra el calculo de los ángulos para cada eslabón, en el cual empleamos la teoría de mecanismos adaptando las ecuaciones a nuestro proyecto, quedando de la siguiente manera, en lenguaje Basic.

```
 D = esl1   'R2 o Eslabon2
 C = esl2   'R3 o Eslabon3
 A = textox 'R1 o X
 B = textoy 'R2 o Y
 teta2 = (89.999999999999 / 180) * PI
   If A > 0 And B > 0 Then
   cuadrante = 1
```

```
  ElseIf A < 0 And B > 0 Then
  cuadrante = 2
  ElseIf A < 0 And B < 0 Then
  cuadrante = 3
  ElseIf A > 0 And B < 0 Then
  cuadrante = 4
  End If
 If A = 0 Then A = 0.0000000001
 If B = 0 Then B = 0.0000000001
 G = A / B
 H = A / D
 i = (B ^ 2 - C ^ 2 + D ^ 2 + A ^ 2) / (2 * B * D)
 j = A / C
 K = (D ^ 2 - A ^ 2 - B ^ 2 - C ^ 2) / (2 * B * C)
 M = Cos(teta2) - G - H * Cos(teta2) + i
 N = -2 * Sin(teta2)
 O = G - (H + 1) * Cos(teta2) + i
 L = Cos(teta2) - G + j * Cos(teta2) + K
 e = -2 * Sin(teta2)
 f = G + (j - 1) * Cos(teta2) + K
  If Val(Text1(0).Text) = 270 And Val(Text1(1).Text) = 270 Then
  teta31 = PI / 2
  teta41 = PI
  Else
  If Val(Text1(1).Text) > -1 Then
  raizteta = e ^ 2 - 4 * L * f
  teta31 = -2 * Atn((-e + Sqr(e ^ 2 - 4 * L * f)) / (2 * L))
  teta41 = -2 * Atn((-N + Sqr(N ^ 2 - 4 * M * O)) / (2 * M))
  Else
  teta31 = -2 * Atn((-e - Sqr(e ^ 2 - 4 * L * f)) / (2 * L))
  teta41 = -2 * Atn((-N - Sqr(N ^ 2 - 4 * M * O)) / (2 * M))
  End If
 End If
If teta41 < 0 Then
teta41 = 2 * PI + teta41
```

Y para graficar recurrimos al comando "Picture" y "Line"

```
Picture1.Cls  'graficacion de los eslabones en los planos
 Picture1.Scale (-500, -500)-(500, 500)

 x1 = D * Cos(teta31)
 x2 = C * Cos(teta41)
 y1 = D * Sin(teta31)
```

```
y2 = C * Sin(teta41)
'graficacion del plano XY
'eslabon 2
Picture1.Line (0, 0)-(x1, -y1), vbRed
Picture1.Line -(x1 - x2, -y1 + y2), vbBlue
Picture1.Circle (0, 0), 20, vbYellow
```

Para el resto de la graficación se emplea el mismo comando, solamente que se debe calcular donde se encontrara cada parte del eslabon para graficarlo correctamente.

Volviendo al funcionamiento del programa, en la ventana principal pueden observar los tres planos coordenados principales, en los cuales se aprecia el movimiento del brazo mecánico, representado esquemáticamente, ya sea en tiempo de programación o en simulación. En el plano XY, se puede apreciar un circulo exterior que representa el alcance máximo del brazo dependiendo este de la configuración física del manipulador, el cual puede ser variable y esa variación de las longitudes se puede cambiar internamente en el programa para hacerlo compatible con diferentes configuraciones de robot SCARA.

Este valor solo será accesible en tiempo de programación del software, ya que una vez compilado no será posible modificar dicho valor, aunque en un futuro podría cambiarse esto para dar acceso al usuario a cambiar dichos parámetros.

Los otros dos planos serán respectivamente XZ y YX, o vistas frontal y lateral. La pequeña ventana en el cuadrante superior derecho representa la orientación del efector final. Para iniciar la escritura del código para los microcontroladores y puntos para posterior simulación pulsamos el botón que aparece en la parte superior, "Iniciar programa", tomando la posición en que se encuentre el manipulador en ese momento como posición inicial. En ese instante se crean en el directorio donde se tenga instalado el programa los archivos que contienen el código fuente base para los microcontroladores, además de otros archivos que empleará el programa para su funcionamiento interno. En este apartado cabe aclarar que si existía alguna secuencia programada con anterioridad, ésta será sustituida por la nueva secuencia que se introduzca.

En la figura 3.19 podemos apreciar como después de iniciado el programa se activan los demás botones.

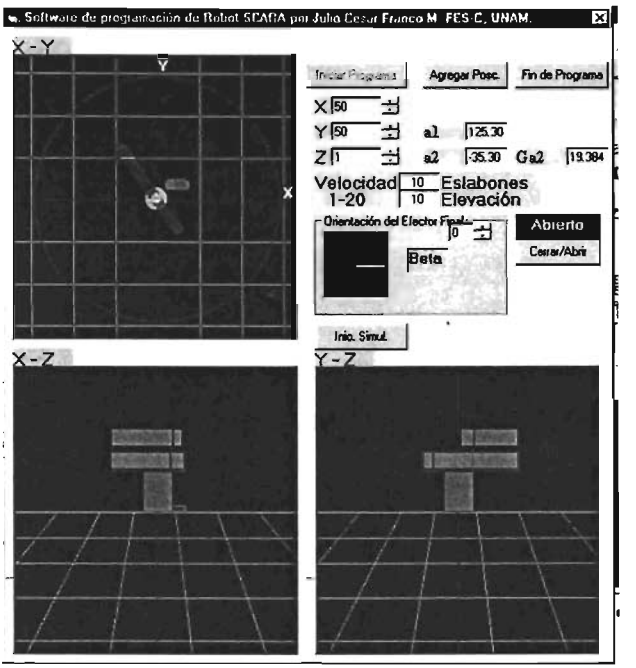

Figura 3.19. Programa Iniciado

Los botones subsecuentes, son respectivamente "Agregar posición" y "Fin de programa", que se explicarán a continuación:

- *Agregar Posición*: Sirve para agregar puntos estacionarios, que serán alcanzados por el manipulador tanto en simulación como en funcionamiento
- *Fin de programa*: Utilizaremos este botón para indicarle a nuestro programa ya esta terminado y no deseamos agregar más puntos estacionarios al programa.

En este momento los códigos fuentes de ambos microcontroladores se encuentran completos y listos para ser compilados por el MPLAB®, y posteriormente ser cargados al robot.

Adicionalmente en la parte superior de este cuadrante, debajo del botón de inicio de programa, se encuentran las casillas destinadas a la entrada de datos coordenados, X, Y, Z, los cuales se pueden introducir con el teclado o por medio de las barras de desplazamiento, tal como se muestra en la figura 3.20.

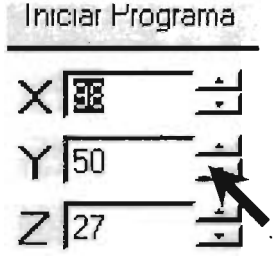

Figura 3.20. Casillas para entrada de datos coordenados.

Cada que se modifique alguno de los datos, el programa de forma automática colocara nuestro manipulador (virtual) en la posición especificada, por lo cual se puede considerar que la entrada de los datos y la respuesta del programa se da en tiempo real, ya que gracias a la potencia de los procesadores actuales, no toma más que unas fracciones de segundo a la computadora realizar las operaciones necesarias para encontrar los ángulos correspondientes para la posición deseada.

Análogamente, encontraremos un cuadro muy similar para la entrada de datos en el efector final, tal como se aprecia en la figura 3.21.

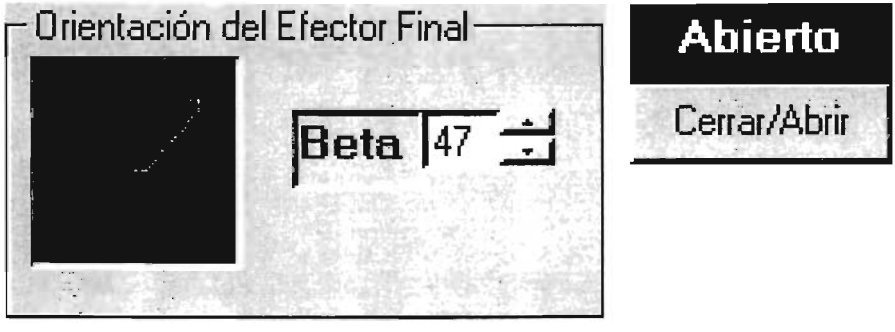

Figura 3.21. Entrada de datos para orientación de efector final.

En este caso, al ángulo que forma el efector final con el sistema de coordenadas universal lo llamaremos Beta, y estará dado en grados.

La entrada de datos en esta casilla funcionará análogamente a la entrada de datos coordenados. Por otra parte el botón ubicado a la derecha nos servirá para activar o desactivar el efector final, siendo la ventanilla azul, la indicativa del estado actual del mismo.

Además se puede observar que en la parte derecha de las entradas de datos coordenados, se encuentran los ángulos a1, a2 y Ga2, que corresponden a la orientación de los eslabones principales y el tercero a la orientación del eslabón 2 con respecto al 1. Estos datos pueden ser de utilidad para visualizar el comportamiento del robot, pero además son de gran utilidad para saber cuanto deben de girar cada uno de los motores.

En la figura 3.22 podemos apreciar los campos para la velocidad a la que deseamos que se mueva nuestro manipulador.

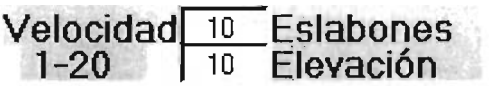

Figura 3.22. Velocidad de Movimiento

La selección de velocidad de los tres grados rotativos, será independiente de la velocidad de elevación. Ambas pueden ser configuradas en una escala del 1 al 20, siendo éstas unidades de movimiento angular, no lineal, puesto que el movimiento lineal varía a cada instante en esta configuración de manipuladores.

En la parte superior del plano coordenado YZ se encuentra el botón para iniciar la simulación, el cual puede ser oprimido en cualquier momento para visualizar el movimiento y los puntos guardados hasta el momento.

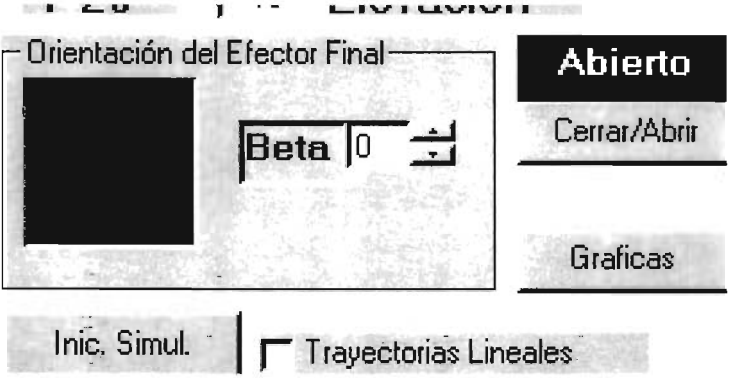

Figura 3.23. opciones Graficar y Trayectorias Lineales

Se contará también con dos opciones adicionales, una de ellas, la casilla de verificación para trayectorias lineales, que, aunque por limitaciones de programación en los microcontroladores no se encuentra implementadas dichas trayectorias en el modelo físico, se prevé a futuro para posibles mejoras y se analizará dinámicamente esta opción en capítulos posteriores.

El funcionamiento de esta opción es bastante sencillo e intuitivo: si se encuentra verificada la casilla, el manipulador seguirá interpolaciones lineales entre los puntos designados ajustando constantemente sus parámetros dinámicos. Si por el contrario, no se encuentra activada, cada grado de libertad, comenzará a avanzar a velocidad constante hacia el nuevo ángulo designado, sin importar la trayectoria que se siga para dicho fin, que será en realidad como funcionará este modelo.

Por otra parte, la opción "Graficas" abrirá una nueva ventana, la cual permitirá graficar las velocidades y aceleraciones de los eslabones, previamente simulados, ya que será en el proceso de simulación cuando el programa tomará los datos para realizar las operaciones necesarias y tener listos los archivos de graficación. Se puede apreciar la nueva ventana en la figura 3.24.

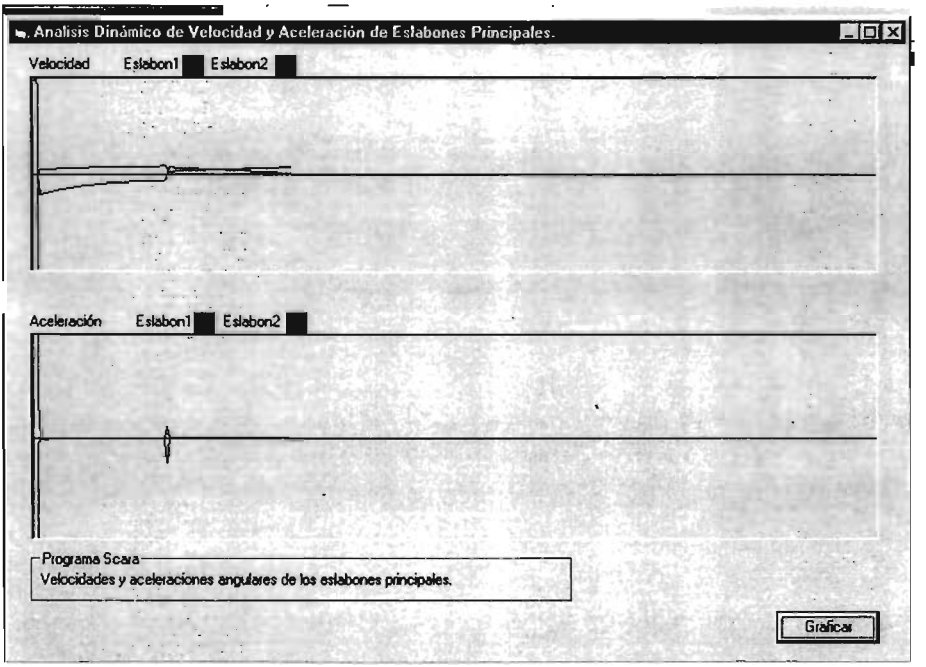

Figura 3.24. Ventana de análisis dinámico

Las ventanas de gráficos aparecerán en un principio vacías, y será hasta que pulsemos el botón de "Graficar" cuando se dibujarán éstas. Los resultados que ellas muestran, y los diferentes resultados que se obtuvieron se tratarán con mas detalle en un capitulo posterior.

Capitulo 4

Diseño de la Estructura del brazo Manipulador

4. Diseño del Brazo Manipulador

4.1 Materiales a Utilizar

Cuando se trabaja en el diseño de un brazo mecánico, como cualquier mecanismo que va a estar sometido a esfuerzos y cargas inerciales, se debe considerar la potencia que pueden proporcionar nuestros actuadores para poder estimar los materiales que deberán ser empleados así como la masa que será capaz de mover. En la mayoría de los robots industriales se emplea acero y aleaciones de materiales ligeros considerando en el diseño los contrapesos para poder balancear estáticamente en la medida de lo posible, los eslabones o grados de libertad más importantes de un manipulador.

En nuestro caso contamos con motores eléctricos de una potencia modesta, y por lo limitado de los recursos disponibles, tendremos que utilizar materiales que nos den la resistencia mecánica suficiente, no en todos los casos la ideal, y que sean lo más ligeros posibles.

Pues así dividiremos en cuatro principales partes la estructura de nuestro manipulador.

1. Base. Que será de acero, para que proporcione un anclaje adecuado del resto del manipulador, considerando que ésta será fija, lo que nos importa es que tenga un buen peso, contrario a lo que sucederá con el resto del manipulador.
2. Elevador del manipulador. En el emplearemos básicamente acrílico y PVC, debido a su bajo costo, ligereza y buena resistencia mecánica a esfuerzos de compresión longitudinales.
3. Eslabones. Éstos estarán constituidos de acrílico de 6mm de espesor para las partes superiores e inferiores y de aluminio debidamente aligerado en las partes laterales.
4. Vástago. Será la parte final del manipulador, el último grado de libertad del mismo y donde existirá la posibilidad de montar un efector, el cual no se considera parte del robot, sino una herramienta de trabajo. Estará constituido primordialmente de aluminio.

Como se puede ver, el empleo del aluminio y plástico será predominante, en aras de reducir la masa, y por lo tanto las fuerzas de inercia.

4.2 Diseño mecánico

La configuración del robot tipo SCARA la podemos apreciar en la figura 4.1.

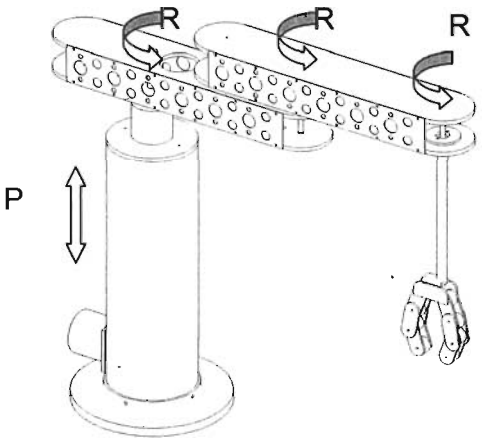

Figura 4.1. Estructura Típica de un robot SCARA

Se pueden apreciar en ella los 4 grados de libertad de que dispone nuestro manipulador. En la estructura del SCARA tradicional, el acoplamiento prismático se encuentra en el final del manipulador, pero con el fin de hacer este modelo de mayor capacidad de carga y menor inercia en el extremo, lo hemos trasladado al principio de la cadena cinemática. De esta forma quedan en el siguiente orden.

1. GDL[1]. Tipo prismático, variando solamente en el eje Z (elevación)
2. GDL. Tipo rotativo, correspondiente al hombro, variando en el plano XY, la posición del tercer sistema coordenado.(posición)
3. GDL. Tipo rotativo, correspondiente al codo, variando en el plano XY, la posición de cuarto sistema coordenado.
4. GDL. Tipo rotativo, correspondiente a la muñeca, variando la orientación del cuarto y último sistema coordenado, y del efector final. Este ultimo puede girar indefinidamente.

Después de haber analizado la configuración que tendrá nuestro manipulador, procederemos a describir cada uno de los puntos y criterios de diseño.

Comenzaremos por la base, la cual la podemos apreciar en la figura 4.2.

La base, como se dijo anteriormente, será la parte más sólida y pesada de todo el conjunto, en ella se encontrará internamente el sistema de elevación. Consta

[1] GDL. Grado de Libertad ...

de una base circular de placa de acero de ½", y el resto está constituido por tubo de acero de 4 ½". La unión estará formada por un anillo soldado a la placa que da cabida a 4 opresores de 3/8". En la parte inferior del cuerpo cilíndrico se montará nuestro primer actuador, un motor de pasos con par retentivo de 40lb-in, con resolución de 200 pasos por revolución. Este por medio de un sistema de cable y poleas será el encargado de la elevación del resto del conjunto.

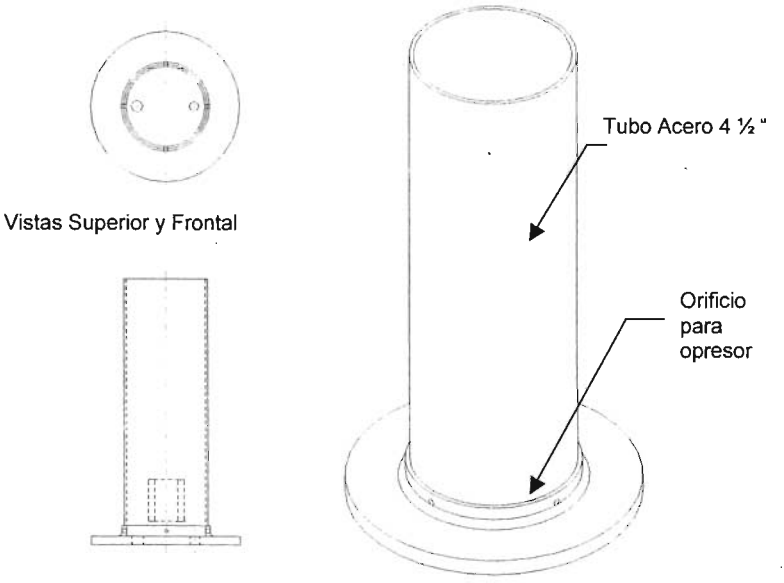

Figura 4.2 Base del brazo manipulador.

Dentro de él se alojarán también dos barras de acero circular, que servirán de rieles para el carro elevador, lo cual permitirá que el movimiento sea suave y preciso. En la parte superior complementará el conjunto una tapa de acrílico de dos piezas, la primera de ellas con orificios para la salida del cuerpo cilíndrico interior y acomodar correctamente en la parte superior los rieles de acero. Podemos apreciar el conjunto en la figura 4.3.

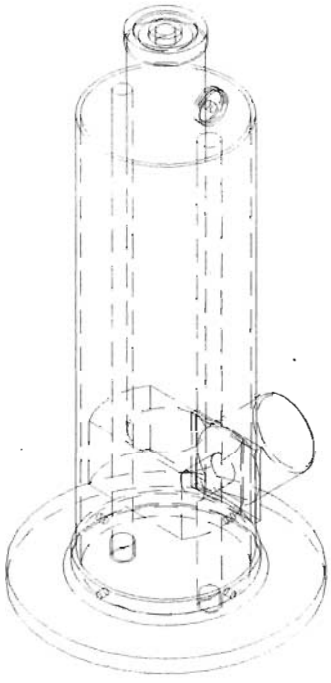

Figura 4.3. Conjunto de Base y Sistema de Elevación

La segunda se unirá a esta pero solo contará con un orificio de salida para el cuerpo cilíndrico de elevación, ambas piezas las podemos apreciar en la figura 4.4.

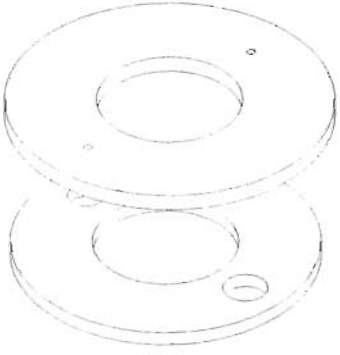

Figura 4.4. Tapa superior de la Base de Manipulador.

El siguiente componente será el conjunto elevador, cuerpo cilíndrico interno y corona superior con eje, puede apreciarse en la figura 4.5.

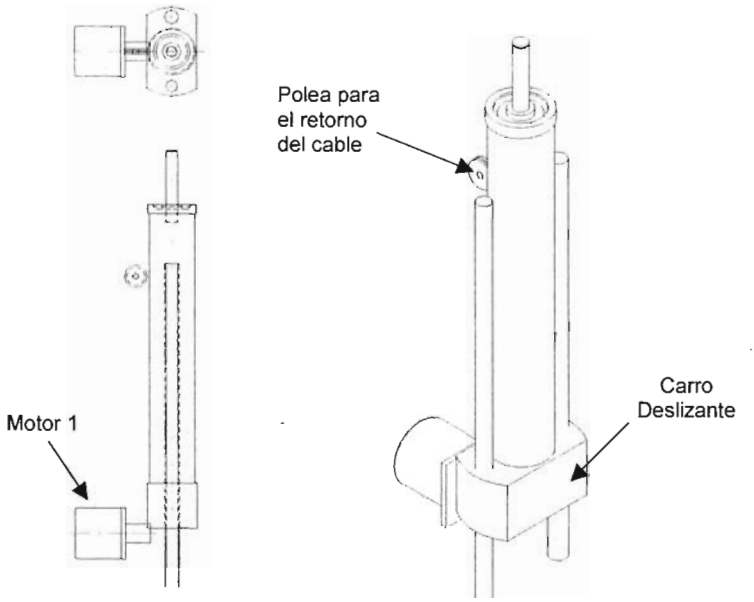

Figura 4.5. Conjunto elevador

Éste como se puede apreciar en la figura, estará constituido por un pequeño "carro" deslizante, sobre el cual se montará el cuerpo cilíndrico interno, el cual soportará el peso del resto del conjunto, con un esfuerzo de compresión longitudinal. En la parte superior, se montará el eje para el segundo eslabón y la corona sobre la cual girará el segundo actuador.

El sistema de elevación estará constituido por un sistema de cable, el cual se enrollará en el motor 1, subirá hasta la polea y regresará para anclarse finalmente en el carro.

En la parte superior de la corona montaremos baleros de carga axial de agujas, con dimensiones 45x30x4. éstos tendrán la función de evitar fricciones entre el primer y segundo eslabón.

El eje será de acero y en la parte inferior se colocará un anillo que centre los baleros de carga axial.

Primer Eslabón Rotativo

Estará constituido principalmente por dos bases de acrílico, de 6mm de espesor donde se montarán el resto de los elementos incluyendo bujes, ejes, motores, circuitos y las tapas laterales de aluminio, podemos apreciar ambas piezas en la figura 4.6.

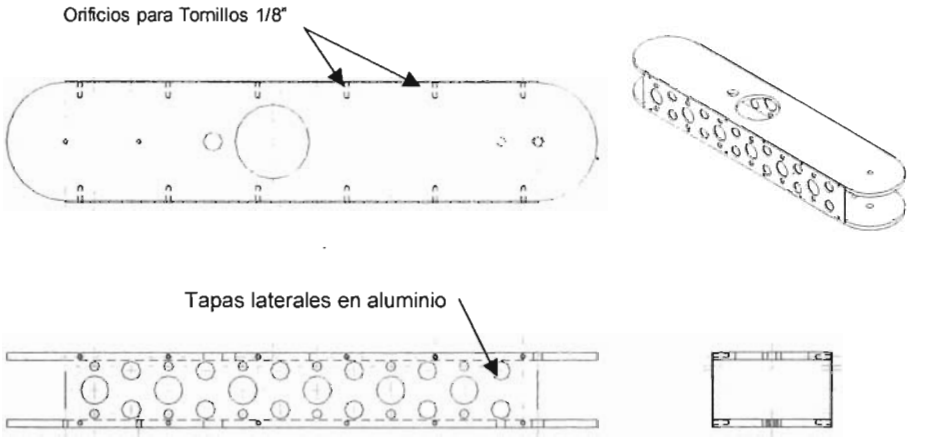

Figura 4.6. Primer eslabón rotativo

Podemos ver la distribución de los orificios con cuerda para recibir la tornillería de 1/8" que sujetará las tapas laterales.

El eje de rotación se encuentra en la posición mostrada debido a que en la parte posterior se montará el primer motor, el cual se desea que realice funciones de contrapeso, el segundo se encontrará en la parte opuesta del eje pero lo más cerca posible de él, para evitar que el momento de la inercia sea menor. Éste se conectará a través de una banda dentada a un juego de engranes en la punta del eslabón, para transmitir desde ahí el movimiento al segundo eslabón rotativo. En la tapa superior se montará un anillo que servirá para ajustar el eje y centrar los baleros de carga axial donde descansará el segundo eslabón rotativo.

Segundo Eslabón Rotativo

Al igual que el primer eslabón, su diseño desplaza el eje y lo coloca 100mm alejado del centro de la parte circular extrema, en el espacio sobrante entre el eje y el extremo posterior colocaremos el motor que será el encargado de transmitir por medio de una banda dentada el movimiento al grado de libertad final.

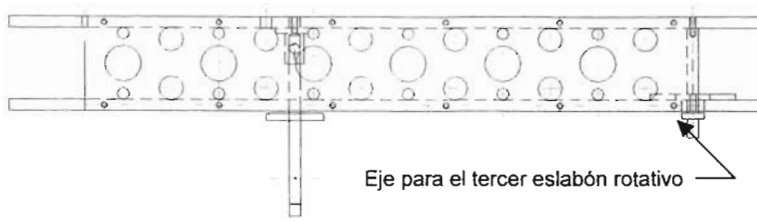

Figura 4.7. Segundo eslabón rotativo

Estará constituido de igual forma por acrílico en las bases y aluminio en los extremos, creando así una estructura ligera pero suficientemente rígida.

Básicamente las diferencias entre el primer y segundo eslabón serán las dimensiones de los barrenos para ejes y baleros, aunque exteriormente tendrán las mismas dimensiones.

Tercer eslabón rotativo

El tercer eslabón rotario o muñeca estará constituido por un eje, un vástago de aluminio y la rueda dentada para la transmisión, lo podemos apreciar en la figura 4.7 también.

Este eslabón será el encargado de posicionar el efector final, o herramienta que se desee colocar en la punta del mismo.

Análisis de esfuerzos

En este apartado se mencionaran los primero y segundo eslabón rotativos como eslabón 1 y 2 respectivamente, aunque en realidad sean los eslabones 2 y 3 del manipulador.

Conjuntamente al diseño mecánico, realizaremos estudio de esfuerzos por medio del Análisis de Elemento Finito (FEA). Lo primero que haremos será calcular los pesos de todos los elementos de que está compuesto el manipulador y suponer también una carga en el efector final para poder realizar el análisis.

Partiendo del diseño que tenemos en Mechanical Desktop, calcularemos los pesos de los elementos plásticos y las tapas laterales de aluminio, bastando para esto asignar un material a cada elemento, para que nos de el peso exacto.

Asignaremos acrílico con densidad de 1.19 gramos/cm³ a las tapas superiores e inferiores de cada eslabón. En Mechanical Desktop bastara con abrir las propiedades del objeto y el programa nos pedirá que asignemos un material. Asignamos el Material y el programa nos dará resultados como muestra la figura 4.2.8. No entraremos mucho en detalle, en el manejo de éste software, debido a que ya existen trabajos dedicados a dicho paquete. Solamente tomaremos algunas referencias de el para complementar este trabajo.

Figura 4.8. Ventana de Propiedades del Sólido

Una vez asignado el material a cada uno de los elementos obtendremos los pesos correspondientes, los cuales se enumeran a continuación.

Tapa inferior eslabón 1:	180 gramos
Tapa superior eslabón 1:	168 gramos
Tapa inferior eslabón 2:	216 gramos
Tapa superior eslabón 2:	217 gramos

De la misma manera, asignaremos aluminio a las tapas laterales de los eslabones, las cuales son iguales las 4, dándonos un peso estimado de

Tapa lateral de aluminio aligerada: 66.7 gramos

Adicionalmente, tendremos los pesos de los motores de pasos que utilizamos como actuadores, los cuales se tienen en dos medidas, al mayor pesando 220 gramos y dos más pequeños con un peso de 120 gramos.

Teniendo todos los datos procedemos a calcular estáticamente las fuerzas debidas al peso así como los momentos que se generan en los ejes y repercuten en los eslabones. Todo esto lo podemos apreciar en forma condensada en la figura 4.9.

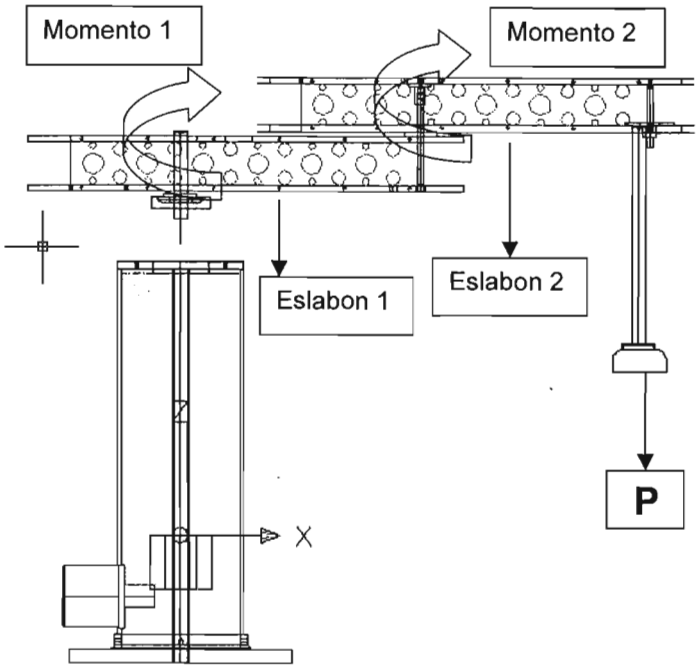

Figura 4.9. Fuerzas estáticas presentes en el manipulador

El peso de cada eslabón será la suma de sus costados, sus acrílicos ejes y motores que lo integren, y su peso se referirá a su centro de masa. La carga P será el peso del efector final, aproximadamente 190 gramos más una carga supuesta de 200 gramos. El eslabón 1 pesa aproximadamente 688 gramos, mientras el eslabón 2 pesa aproximadamente 565 gramos, sin contar el efector final, por supuesto.

En base a esto, y realizando algunos cálculos para encontrar el centro de masa, y posteriormente los pares resultantes, tendremos los centros de masa en X = 167.56 para el eslabón 1 y X = 146.27 para el eslabón 2.

Calculando los esfuerzos en los ejes, para obtener la resultante sobre los rodamientos, tenemos que el momento en el eje 1 será de 364,010 gramos-mm, es decir 71.34 N-mm y el peso sobre su acrílico inferior, es decir la componente normal será 1,611 gramos. Mientras en el acrílico superior tendrá un esfuerzo normal de 923 gramos y un par de 1773.8 gramos-mm, ó 17.48 N-mm.

Ya que Mechanical trabaja con milímetros, solo es necesario aplicar las fuerzas y los soportes adecuados para que el paquete realiza los cálculos de elemento finito, arrojando los siguientes resultados.

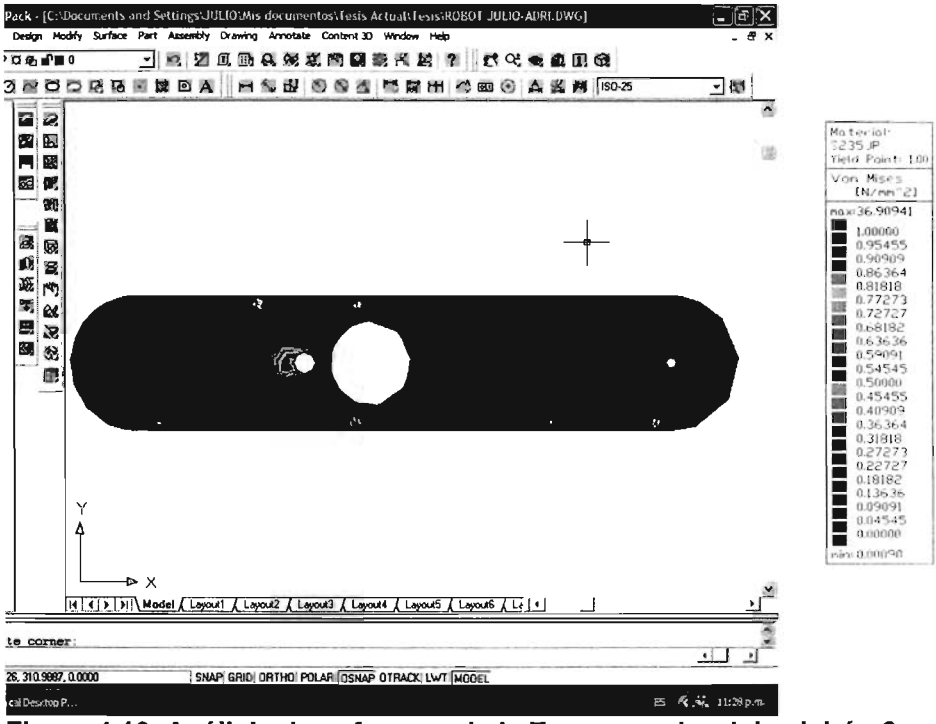

Figura 4.10. Análisis de esfuerzos de la Tapa superior del eslabón 2

En la Figura 4.10 podemos apreciar las áreas de esfuerzo del componente superior del eslabón 1 y del lado derecho los valores de dichas áreas, expresados en N/mm², en este caso el esfuerzo mayor es de 1 N/mm², considerando que la resistencia del acrílico es de 68.6 N/mm² nos damos cuenta que el material está bastante sobrado para el proyecto.

Como era de esperarse, los mayores esfuerzos se concentran en las áreas más delgadas de nuestro diseño, lo cual era de esperarse, así como en los barrenos más cercanos a éstas áreas de esfuerzo máximo.

Cabe destacar que el análisis que realiza Mechanical Desktop es tridimensional, por lo cual se vuelve más confiable, al considerar fuerzas de diferentes planos.

En la figura 4.11 podemos apreciar lo que sucede con la tapa inferior del eslabón 1, en la cual los resultados nuevamente eran de esperarse, repitiéndose el fenómeno de esfuerzo en los barrenos cercanos a el área del eje.

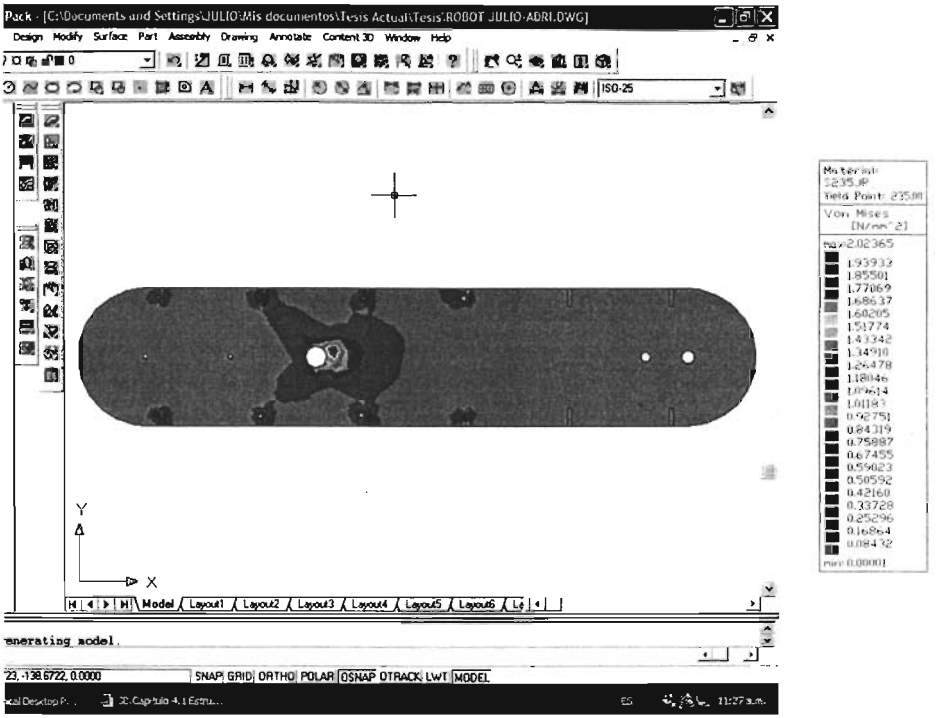

Figura 4.11. Análisis de esfuerzos de la tapa inferior del eslabón 1

Nuevamente al lado derecho podemos apreciar la tabla con los valores correspondientes a cada zona, siendo 1.9 N/mm² el máximo esfuerzo que se grafica en nuestra pieza. En este caso en la parte derecha de nuestra pieza no se localizan los esfuerzos en el barreno para el eje del siguiente eslabón de la forma que sucede con la tapa superior, esto debido a que la parte inferior es más una guía mientras la superior aparte de ser una guía soporta el peso del siguiente eslabón, del efector y de la carga.

En la figura 4.12 apreciamos de igual manera los esfuerzos de la parte inferior del eslabón 2, donde razón de la zona de máximo esfuerzo se puede explicar debido a que es ahí donde sufre la torsión reflejada por el peso en la punta derecha.

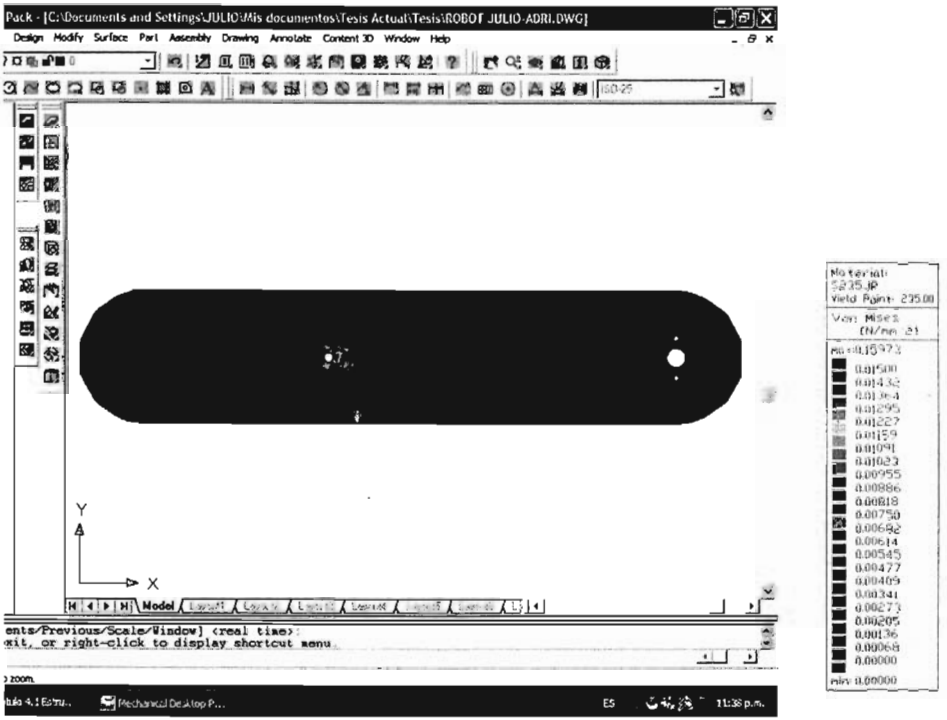

Figura 4.12. Análisis de esfuerzo de la parte inferior del eslabón 2

En la tabla de la derecha, nuevamente apreciamos los valores de esfuerzo para cada zona de la parte, siendo éstos cada vez menores debido a que mientras más nos acercamos a la punta, obviamente son menores los pares que actúan, así como es menos el peso que soporta la extremidad. Podemos apreciar que el esfuerzo máximo en esta ocasión es de 0.015 N/mm .

Por último apreciamos en la figura 4.13 el análisis de esfuerzo de la parte superior del eslabón, la última que se analizará, debido a que las tapas laterales de aluminio, no son tan críticas considerando que por ser metálicas tienen una resistencia mucho mayor que las piezas acrílicas.

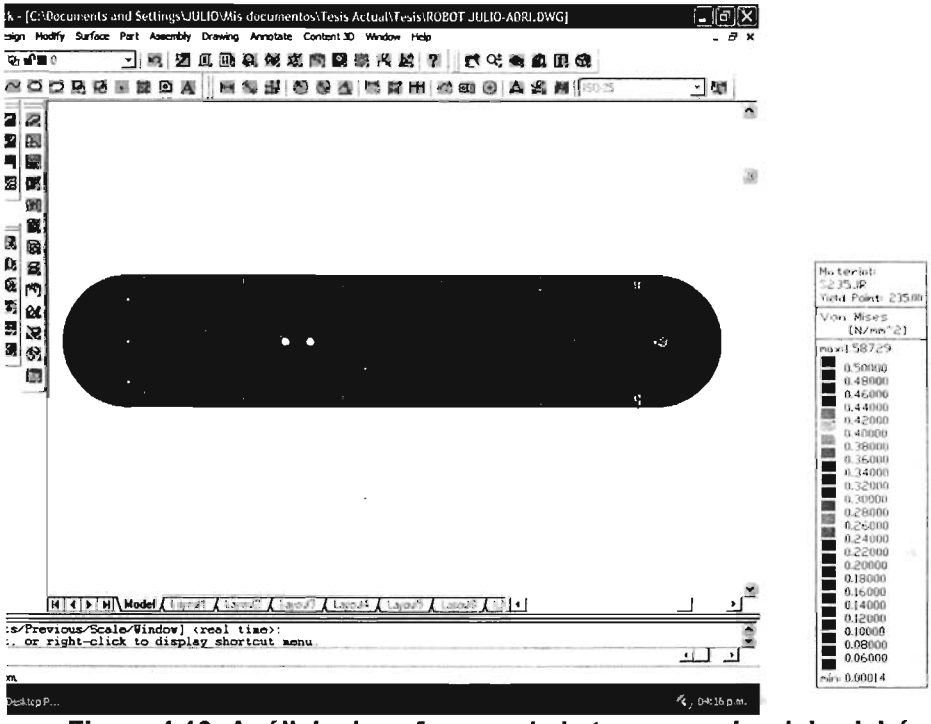

Figura 4.13. Análisis de esfuerzos de la tapa superior del eslabón 2

En esta figura alcanzamos a apreciar pequeños puntos de esfuerzo en la parte izquierda, estos debidos a la carga del actuador ahí, y en la parte derecha se encuentra la mayor zona de esfuerzo debido a que es en ese punto donde se soporta el peso del efector final y la carga.

Por último, hay que mencionar que si se requiere un análisis de la estructura cuando esta se encuentra en movimiento, se tendría que agregar fuerzas adicionales como cargas inerciales y aceleraciones de Coriolis, lo cual haría el problema más complejo e interesante a la vez, pero quedará para un estudio posterior el resolverlo, puesto que sale de lo proyectado en los objetivos de este trabajo.

Capitulo 5

Análisis de Trayectorias

5.1 Simulación de Trayectorias con software desarollado

Una vez que se han conjuntado las teorías en un software capaz de simular los movimientos del manipulador, es posible analizar las diferentes trayectorias que se pueden dibujar con el prototipo, o los puntos que son posibles de alcanzar con él. Además de lo comentado en el capitulo referente al software, se podrán ver más a fondo algunas de las características de este software.

Lo primero que se hará es tomar una serie de puntos en el espacio para que el manipulador los vaya alcanzando uno a uno, en modo programación y simulación. Normalmente el software comenzará con el punto 50,50,0 y su efector en 0 grados, pero puede ser llevado a cualquier posición para tomar esta como punto inicial. Se puede ver el punto inicial por defecto en la figura 5.1.

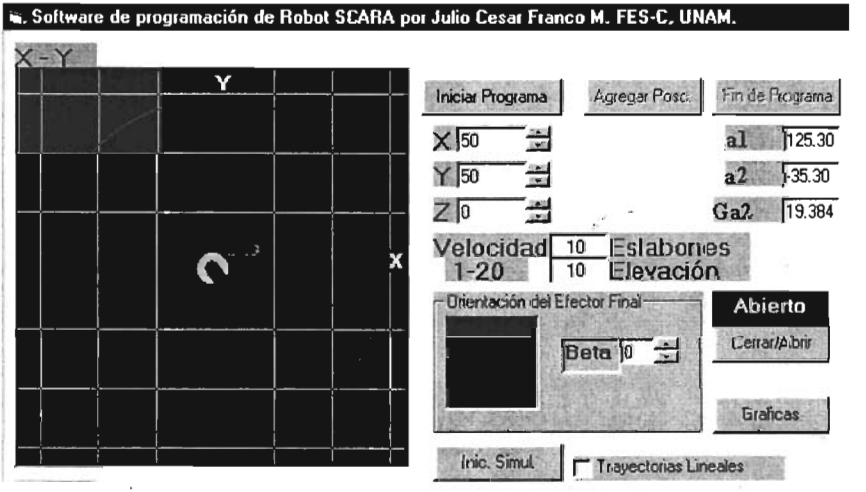

Figura 5.1. Posición Inicial del Manipulador

Posteriormente vamos a desplazarnos hacia el punto 200,200,0 y se observará el comportamiento de los parámetros articulares. Esto se puede observar en la figura 5.2.

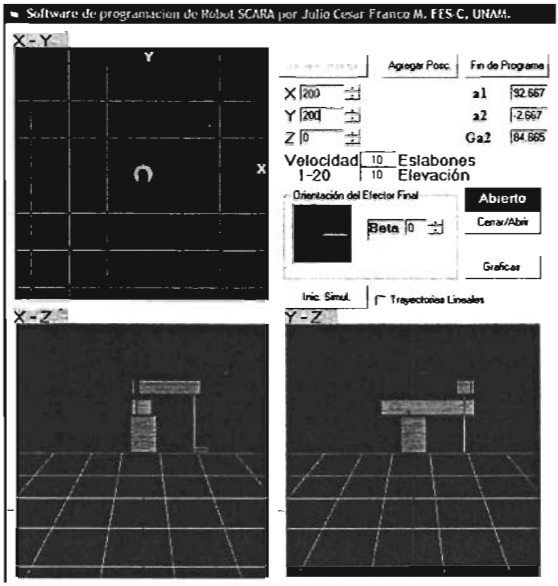

Figura 5.2, Alcanzando el punto 200,200,0 en modo programa

Nótese el cambio que ocurre en los ángulos **a1**, **a2** y **Ga2**, que son los que determinan la posición del brazo.

Ahora se introducirá como nueva coordenada –40,-40,20 y se observará que sucede con el manipulador.

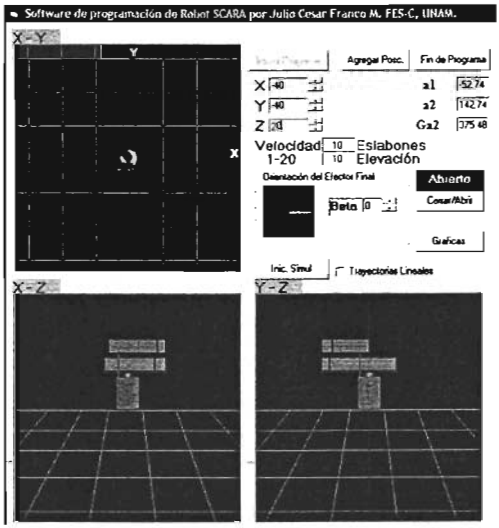

Figura 5.3, Alcanzando punto –40,-40,20

Se finalizará esta pequeña demostración del programa regresando al punto 50,50,0, como se muestra en la figura 5.4.

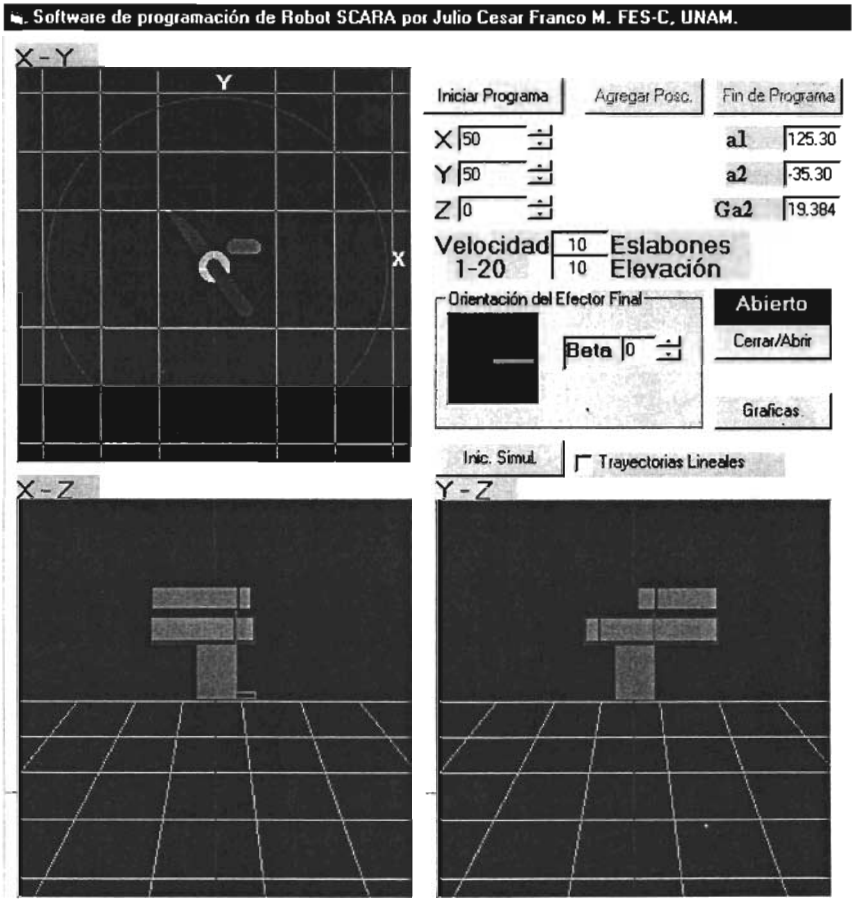

Figura 5.4. Posición Final del Robot

Ahora se procederá a seleccionar un tipo de trayectoria entre los puntos, ya sea punto a punto o por medio de trayectoria lineal, para observar el diferente comportamiento del manipulador. Habiéndose hecho esta selección, damos clic en el botón "Iniciar simulación", para comenzar a simular los puntos grabados en la fase de programación.

Comenzaremos analizando las trayectorias lineales del manipulador y se observara lo que sucede, a partir de la figura 5.5.

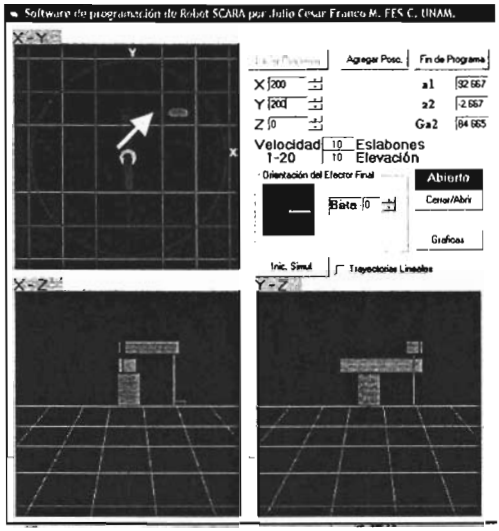

Figura 5.5, alcanzando la primera posición en modo simulación

Hasta este punto como era de esperarse, no existe mayor problema en alcanzar la posición deseada, pero se observara que pasa con los demás puntos. Obsérvese la figura 5.6

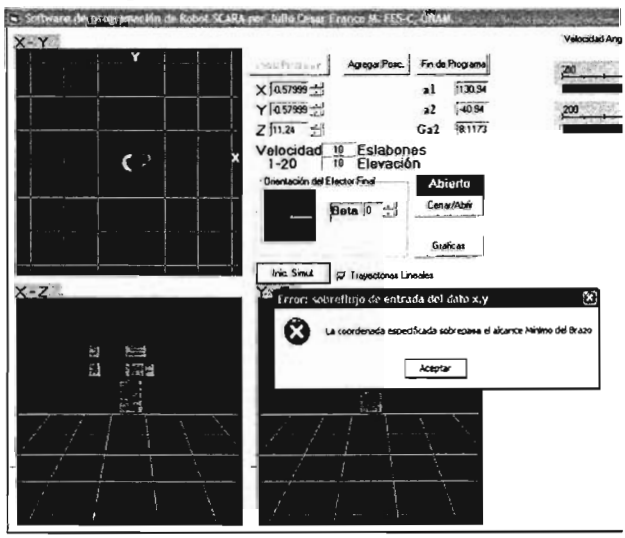

Figura 5.6, mensaje de error al intentar alcanzar el punto 3 por trayectoria lineal

Cuando el manipulador intenta alcanzar el punto –40,-40,20 desde el punto 400,400,0 por medio desde una trayectoria lineal, se registra un error de

alcance mínimo de brazo, es decir que se trata de alcanzar un punto demasiado cercano que se interfiere la base, lo cual era obvio, ya que entre la línea que une los dos puntos se encuentra el origen 0,0 del plano XY. Se considero en el software que el alcance mínimo estaría considerado por una distancia de 30 mm desde el origen, en modo programa ni simulación no puede ser alcanzado ningún punto dentro de estos límites, porque se tendrá como resultado un error similar. Estos limites pueden ser modificados, pero no esta al alcance del usuario del Software hacerlo, ya que es antes de la compilación cuando se establecen estos parámetros, como también el largo de los eslabones y limites de alcance.

La historia es diferente si las trayectorias son no lineales. Ahora se verá el resultado del mismo conjunto de puntos para la trayectoria no lineal. Se comenzará en el mismo punto tal como lo muestra la figura 5.7.

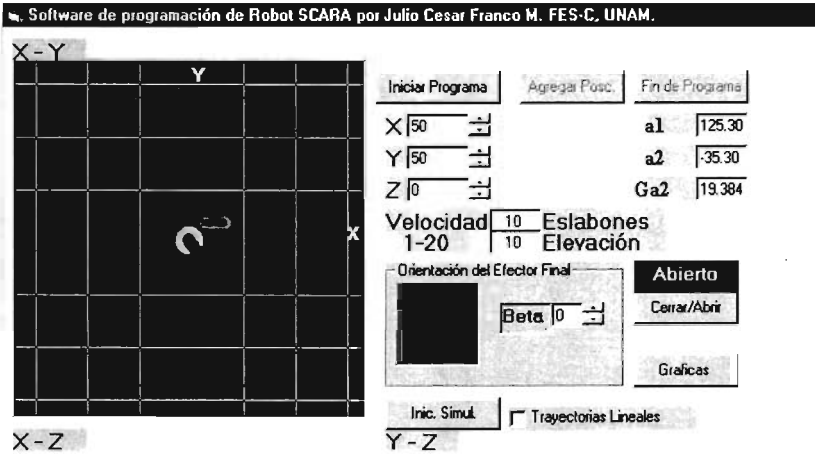

Figura 5.7. Iniciando la Simulación con trayectorias no lineales

Como la trayectoria queda grabada en la computadora, solo necesitamos desactivar el cuadro de verificación de "Trayectorias Lineales". Los resultados del primer movimiento pueden ser apreciados en la figura 5.8.

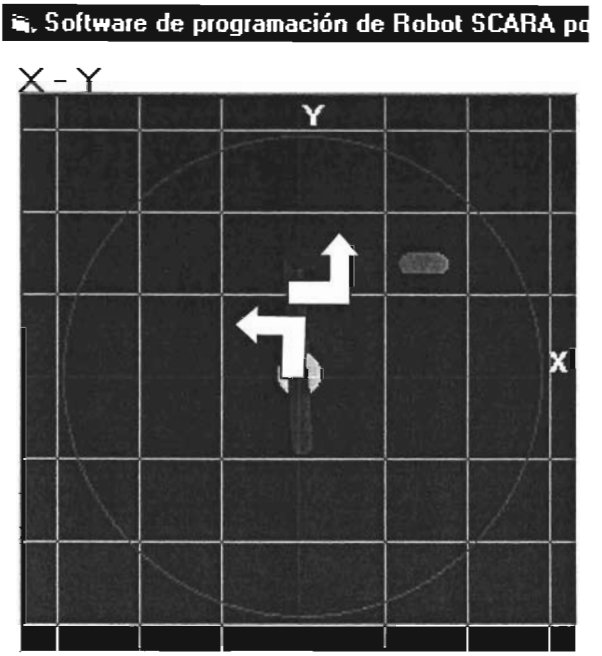

Figura 5.8. Primer movimiento en trayectoria no lineal

A diferencia de las trayectorias lineales, en este caso cada grado de libertad del robot girará a una velocidad predefinida hasta alcanzar la posición angular requerida, sin importar la sincronía con los demás grados, es decir, que cada grado de libertad puede llegar en diferente instante, y no necesariamente se trazará una línea como trayectoria.

En la figura 5.9 se continúa con la serie de puntos de la trayectoria anterior. El punto que a continuaciones alcanza es el –40,-40,20. se puede observar que no presentamos problemas de alcance, o de choque, en este tipo de trayectoria.

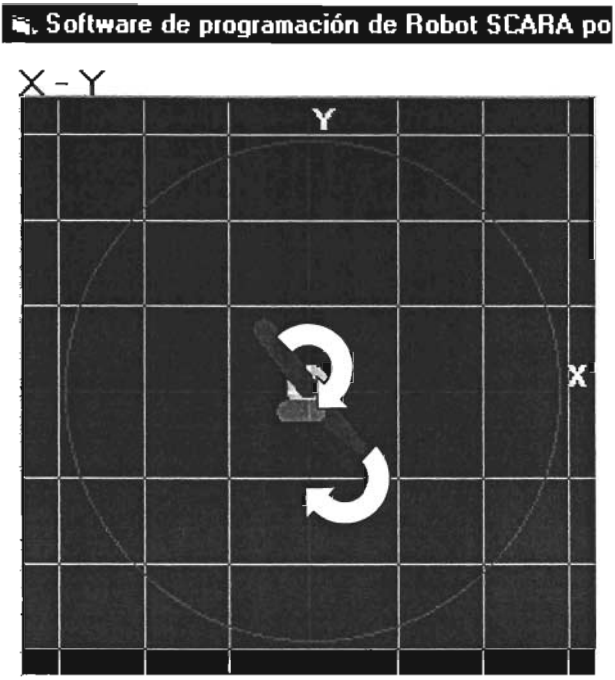

Figura 5.9. Alcanzando el punto –40,-40-20

Como cada grado de libertad alcanzará su posición angular sin importar, o tener que forzar su trayectoria a una línea entre dos puntos, no existe la posibilidad de que en la simulación de trayectorias no lineales, exista una colisión del efector final con la base, debido a que en la programación no se permite agregar ningún punto de esta naturaleza. Debido a esto, se debe de escoger el tipo de trayectoria más adecuado dependiendo de la tarea que efectuará el manipulador.

5.2. Análisis de Velocidad y Aceleración para trayectorias punto a punto con perfil estacionario.

Una vez que se ha introducido una serie de puntos en el programa y se comienza la simulación, el programa cuenta con un análisis dinámico simple de las velocidades angulares de los grados de libertad principales, los cuales pueden observarse en tiempo real cuando se esta simulando una trayectoria en la parte superior derecha de la interfase, como se puede apreciar en la figura 5.10.

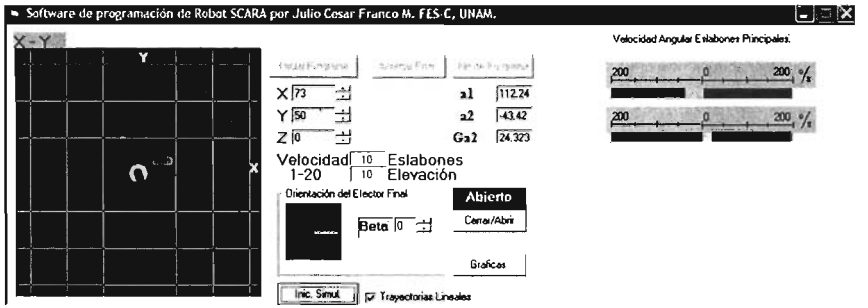

Figura 5.10, velocidad angular de los eslabones en tiempo real

Cuando las trayectorias son lineales en este tipo de manipuladores las velocidades que se presentan en los grados de libertad principales, es decir, los que posicionan el manipulador en el plano XY, varían constantemente para preservar una velocidad lineal constante.

Los alcances en la simulación y análisis dinámico se limitaran a una velocidad lineal fija, y en el caso de trayectorias no lineales, a una velocidad angular fija también. El variar los parámetros afectará solo al modelo real, esto al menos, hasta que se realice una nueva versión del programa, posterior a la que en el momento de escribir esto se utiliza, la cual es 2.1.

Adicionalmente a las barras de velocidad, en la interfase principal contamos con un botón "Graficas" el cual al ser presionado mostrara la pantalla mostrada en la figura 5.11.

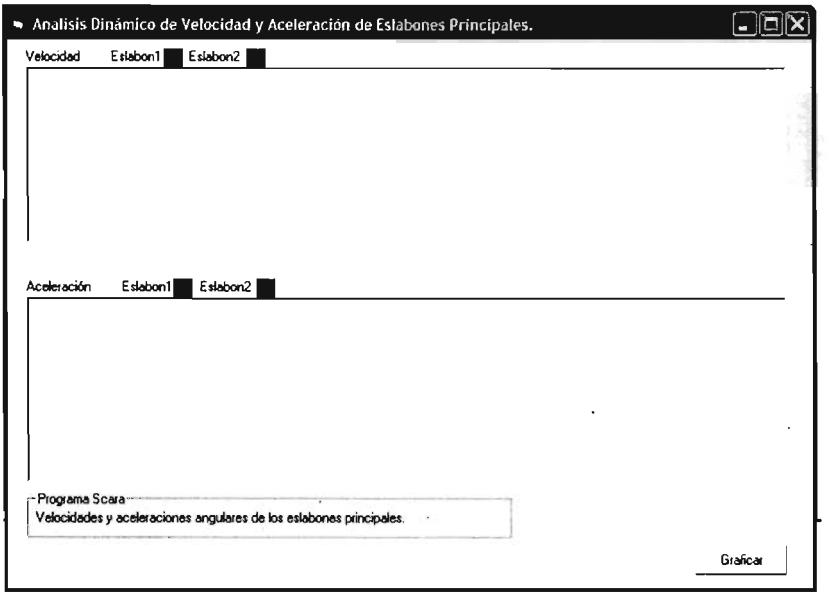

Fig. 5.11. Interfaz de gráficas Velocidad y Aceleración

En la pantalla podemos apreciar que la nueva interfaz que aparece cuenta con dos ventanas, la primera de ellas para graficar la velocidad de los eslabones principales y la segunda las aceleraciones de los mismos.

Para que el programa genere las respectivas gráficas bastara con presionar el botón "Graficar" y veremos como se dibujan las graficas de velocidad y aceleración contra tiempo. Puede apreciarse en la Figura 5.12

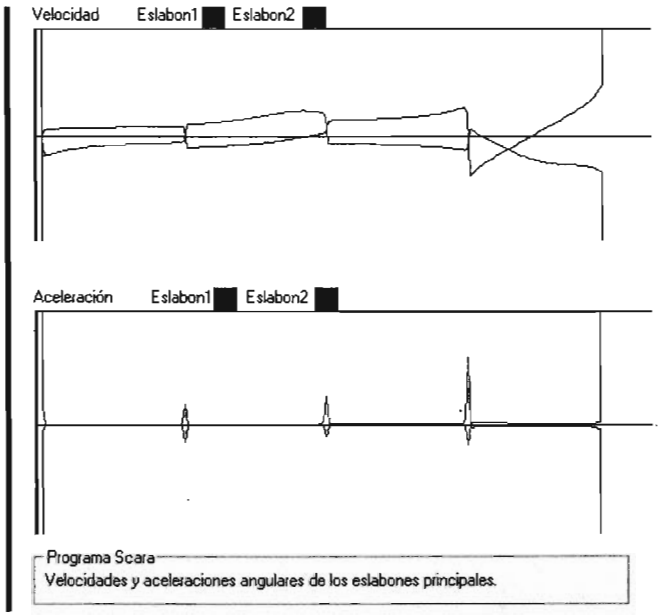

Figura 5.12. Dibujado de las Gráficas Velocidad y Aceleración para eslabones principales

En la anterior figura se puede apreciar como la velocidad varía constantemente, esto se debe a que la gráfica corresponde a una trayectoria lineal. Ahora, hay que decir, que la curva se encuentra idealizada, ya que como se puede apreciar no existe vibraciones en las gráficas, esto es el resultado de emplear las nuevas ecuaciones derivadas de las ecuaciones de posición. Con la aceleración sucede algo análogo.

Cabe mencionar, que en realidad esto es imposible en el modelo real, ya que los motores de pasos, y en menor medida los servomotores, no alcanzan cada punto angular requerido para que la velocidad sea variada de una forma ideal, produciéndose así "ruido". Si deseamos una gráfica más real del modelo, debemos de tomar esto en consideración. El programa también lo incorpora, aunque no en la compilación final, y es interesante ver los resultados, los cuales se pueden comparar con algunos de los más recientes "Papers" de investigación sobre robótica, los cuales nos muestran una gráfica muy similar, siendo de la forma mostrada en la figura 5.13.

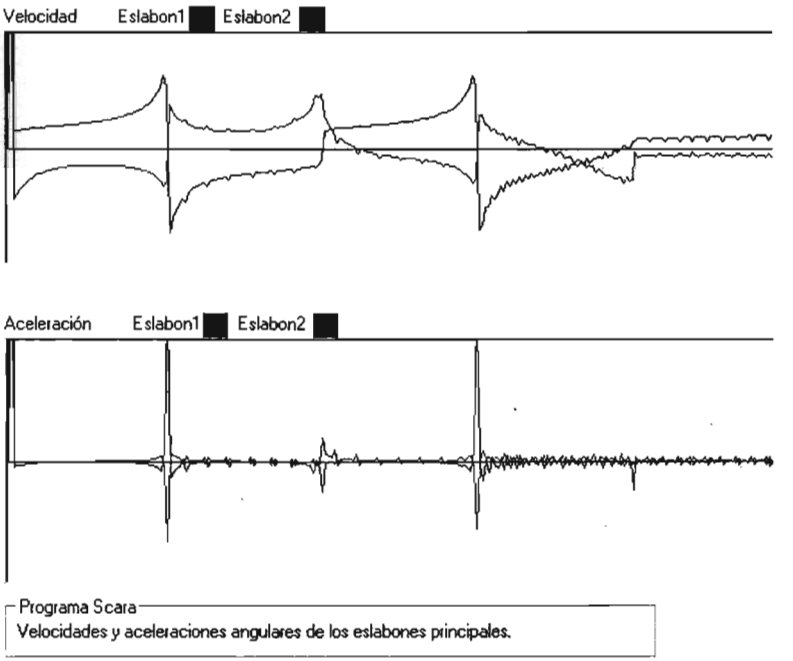

Figura 5.13. Graficas de Velocidad y Aceleración cuando se presenta el "Ruido" provocado por las limitaciones fisicas de los actuadores.

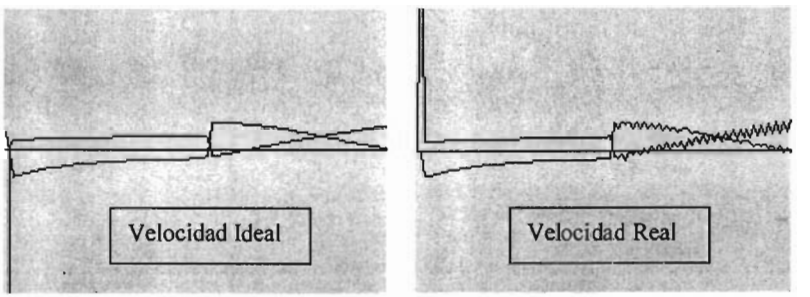

Fig. 5.14. Comparativa de Gráfica Ideal y Gráfica Real

En la figura 5.14 se muestra la comparativa de las gráficas del movimiento Real y el Ideal, de Movimientos similares para poder apreciar de mejor forma las evidentes diferencias en la continuidad de la línea.

Conclusiones

Conclusiones

Es viable la utilización de microcontroladores para el diseño de la tarjeta de control de un robot tipo SCARA, debido a que resultan ser lo suficientemente prácticos, de bajo costo y altas prestaciones para encarar un proyecto como este, pero no se debe perder de vista que así como ventajas, existen algunas limitaciones, la principal de ellas, es la memoria de programa del microcontrolador.

En un microcontrolador se pueden almacenar hasta 1 K de instrucciones, esto es, 1024 líneas de programación, pero el lenguaje ensamblador, de bajo nivel, hace que esta cantidad se vuelva insuficiente al tener que realizar una programación más compleja si queremos que se manejen los cálculos internamente en el controlador, de forma que si realizamos una interpolación lineal con el manipulador, consumiría mucha memoria y limitaría el numero de posiciones estacionarias.

Por otra parte, además de su bajo costo, la velocidad a la que trabaja el microcontrolador, que es de 1 Mhz. En tiempo real, nos permite gran confiabilidad de que el programa será ejecutado con rapidez y precisión, además de una independencia del uso de una PC.

La validez de la teoría de mecanismos, queda demostrada no solo al hacer la comparación matemática con respecto a los cuaternios, sino implementándola en el modelo real, obteniendo resultados coherentes inclusive en el análisis dinámico (sencillo) que se efectúa basándose en esto.

A través de Visual Basic, tomando como referencia las teorías de control, nos es posible crear un software que sea capaz de dar control al manipulador, mediante la simulación y la creación de un programa en lenguaje ensamblador, que posteriormente es compilado y cargado con software de terceros, siendo posible en un futuro implementar la compilación y vaciado en el mismo software, lo cual de momento, queda fuera del alcance de este trabajo.

Por último, para diseñar el brazo manipulador, es recomendable recurrir a algún paquete de diseño por computadora, para agilizar la tarea y tener un proceso de construcción mejor orientado. Para el presente trabajo, se recurrió a Mechanical Desktop, software popular en ingeniería mecánica por sus prestaciones y que permitió incluso, realizar análisis de esfuerzos en los materiales empleados.

Queda para posteriores trabajos, relacionados a esta rama de la ingeniería, la tarea de implementar mejores dispositivos para el control, manejo de información y análisis.

Apéndice A

Comandos del Lenguaje Ensamblador

APÉNDICE "A"
COMNDOS DEL LENGUAJE ENSAMBLADOR

ADDLW Agregar el valor de L a W.

Sintaxis: [Etiqueta] ADDLW k

Operandos: $0 \leq k \leq 255$

Operación: (W) + k → W

Estados Afectados: C, DC, Z.

Descripción: Los contenidos del acumulador W y la literal 'k' son sumados, y el resultado de la operación es colocado en el acumulador W. Este es una instrucción de un solo ciclo.

Ejemplo: ADDLW 0x15

Antes de la instrucción

W = 0x10

Después de la instrucción

W = 0x25

ADDWF Sumar W y f

Sintaxis: [Etiqueta] ADDWF f,d

Operandos: $0 \leq f \leq 127$, $d \in [0,1]$

Operación: (W) + (f) →destino

Estados afectados: C, DC, Z

Descripción: suma los contenidos del registro 'f' con los del acumulador W. El resultado de la operación tiene como destino el acumulador W, si 'd' es 0, y como destino el registro 'f' si 'd' es 1.

Ejemplo ADDWF FSR, 0

Antes de la instrucción

W = 0x17

FSR = 0xC2

Después de la instrucción

W = 0xD9

FSR = 0xC2

ANDLW 'Y' lógico de la literal con W

Sintaxis: [etiqueta] ANDLW k

Operandos: $0 \leq k \leq 255$

Operación: (W).Y. (k) $\rightarrow$W

Estado Afectado: Z

Descripción: Los contenidos del registro W y la literal 'k' de ocho bits son comparados con la operación lógica 'Y' y el resultado de la operación es colocado en el acumulador W.

```
Ejemplo 1  ANDLW 0x5F
           Antes de la instrucción        ; 0101 1111 (0x5F)
           W = 0xA3                       ; 1010 0011 (0xA3)
           Después de la  Instrucción     ;_____________________
           W = 0x03                       ; 0000 0011 (0x03)
```

ANDWF 'Y' lógico del acumulador W con el registro f

Sintaxis: [etiqueta] ANDWF f,d

Operandos: $0 \leq f \leq 127$ d $\in [0,1]$

Operación: (W).AND. (f) $\rightarrow$ destino

Estado Afectado: Z

Descripción: los valores del registro 'f' y los del acumulador W son comparados por medio de la operación lógica 'Y', el resultado de la operación se guarda en el acumulador W si el valor en 'd' es 1, y se guardará en el registro 'f' si el valor en 'd' es 0.

```
Ejemplo:    ANDWF FSR, 1
            Antes de la instrucción          ; 0001 0111 (0x17)
            W = 0x17                         ; 1100 0010 (0xC2)
            FSR = 0xC2                       ;---------- ------
            Después de la instrucción.       ; 0000 0010 (0x02)
            W = 0x17
            FSR = 0x02
```

BCF Limpiar bit en f

Sintaxis: [etiqueta] BCF f,b

Operandos: $0 \leq f \leq 127$, $0 \leq b \leq 7$

Operación: 0 $\rightarrow$f<b>

Estado Afectado: Ninguno

Descripción: El bit 'b' en el registro 'f' es limpiado, es decir, toma el valor 0 o estado bajo.

```
Ejemplo:  BCF FLAG_REG, 7
            Antes de la Instrucción
            FLAG_REG = 0xC7
            Después de la Instrucción        ; 1100 0111
            FLAG_REG = 0x47                  ; 0100 0111
```

BSF Activar bit en f

Sintaxis: [etiqueta] BSF f,b

Operandos: 0 ≤f ≤127, 0 ≤b ≤7

Operación: 1 → f<b>

Estado Afectado: Ninguno

Descripción: el bit 'b' en el registro 'f' es activado, toma el valor de 1 o estado alto.

BTFSC Probar bit, saltar si esta bajo.

Sintaxis: [etiqueta] BTFSC f,b

Operandos: 0 ≤ f ≤127 0 ≤b ≤7

Operación: saltar si (f<b>) = 0

Estado Afectado: Ninguno

Descripción: Si el bit 'b' en el registro 'f' tiene un estado bajo, entonces la siguiente instrucción no se ejecutará, y en vez, se ejecutara un NOP, es decir 'ninguna operación' convirtiéndose en una instrucción de dos ciclos de programa. En caso de tener un estado alto el bit probado, continuara la ejecución del programa de forma normal.

```
Ejemplo:   HERE BTFSC FLAG, 4
           FALSE GOTO PROCESS_CODE
           TRUE•
                 •
                 •
```

En este caso existen dos posibilidades, una es que el bit se encuentre en estado alto y la siguiente instrucción lleve el programa a la etiqueta PROCESS_CODE. La segunda es que se encuentre en estado bajo y por consiguiente no se efectúe la siguiente operación y continúe el programa en la etiqueta TRUE.

BTFSS Probar bit 'b' en registro 'f', saltar si esta alto.

Sintaxis: [etiqueta] BTFSS f,b

Operandos: 0 ≤f ≤127, 0 ≤b < 7

Operación: saltar si (f<b>) = 1

Estado Afectado: Ninguno

```
Ejemplo    HERE BTFSS FLAG, 4
           FALSE GOTO PROCESS_CODE
           TRUE•
                •
                •
```

Esta instrucción es similar a la anterior, con la diferencia de que esta vez la siguiente instrucción no será ejecutada cuando el bit 'b' del registro 'f' se encuentre alto, y se ejecutará cuando este se encuentre bajo. Estas dos últimas instrucciones son de gran utilidad cuando se quiere conocer el estado de las entradas y tomar una decisión en tiempo real dependiendo del estado externo, son vitales para crear un sistema de lazo cerrado, pero en general pueden comprobar el estado de cualquier bit en cualquier registro.

CALL Llamar subrutina

Sintaxis: [etiqueta] CALL k

Operandos: 0 ≤k ≤2047

Operación: (PC)+ 1→TOS, k →PC<10:0>, (PCLATH<4:3>) →PC<12:11>

Estado Afectado: Ninguno

Descripción: Llamado de una subrutina. Primero, la dirección de regreso que sería PC + 1 es colocada en el 'apilador' o 'stack', esto para recordar a donde debe volver el programa después de la subrutina, posteriormente la dirección de 11 bits es cargada en nuestro Contador de Programa, 'PC', y los bits más significativos son cargados con el valor de PCLATH, bits 4 y 3. Esta es una instrucción de dos ciclos de programa. Esta es una instrucción muy similar, podríamos decir igual, a la empleada en el lenguaje Basic.

CLRF Borrar f

Sintaxis: [etiqueta] CLRF f

Operandos: $0 \leq f \leq 127$

Operación: 00h $\rightarrow$f, 1 $\rightarrow$Z

Estado Afectado: Z

Descripción: Los contenidos del registro 'f' son borrados, y por consiguiente al presentarse un 'cero' el bit de estado Z es afectado y se activa.

Ejemplo 1 CLRF FLAG_REG
Antes de la Instrucción
FLAG_REG=0x5A
Después de la Instrucción
FLAG_REG=0x00
Z =1

CLRW Borrar W

Sintaxis: [etiqueta] CLRW

Operandos: Ninguno

Operación: 00h $\rightarrow$W, 1 $\rightarrow$Z

Estado Afectado: Z

Descripción: El registro W, nuestro acumulador, es borrado, de igual forma se afecta el bit de estado Z.

Ejemplo 1 CLRW
Antes de la Instrucción
W = 0x5A
Después de la Instrucción
W = 0x00
Z =1

CLRWDT Clear Watchdog Timer

Sintaxis: [etiqueta] CLRWDT

Operandos: Ninguno

Operación: 00h →WDT, 0 →Cuenta del preescalador de WDT ,

1 →TO, 1 →PD. Estado Afectado: TO, PD

Descripción: Esta instrucción, borra el temporizador de perro guardián. También se borra el preescalador para el WDT. Los bits de estado TO y PD se activan.

Ejemplo 1 CLRWDT
Antes de la Instrucción
Contador del WDT = x
Preescalador del WDT =1:128
Después de la Instrucción
Cuenta del WDT =0x00
Cuenta del preescalador del WDT =0
TO =1
PD =1
Preescalador del WDT =1:128

COMF Complementar f

Sintaxis: [etiqueta] COMF f,d

Operandos: $0 \leq f \leq 127$, $d \in [0,1]$

Operación: (f) →Destino.

Estado Afectado: Z

Descripción: Los contenidos del registro 'f' son complementados con 1's, si 'd' es 0 el resultado es guardado en W, y si 'd' es 1 el resultado de la operación es guardado en el registro 'f'.

Ejemplo 1 COMF REG1, 0 REG1= 0x13
Antes de la Instrucción W = 0xEC
REG1= 0x13
Después de la Instrucción

DECF Decrementar f

Sintaxis: [etiqueta] DECF f,d

Operandos: $0 \leq f \leq 127$, $d \in [0,1]$

Operación: (f) - 1 $\rightarrow$destino

Estado Afectado: Z

Descripción: Se decrementa el valor contenido en el registro 'f' en 1. Si 'd' es 0, el resultado se guardará en W, si 'd' es 1, el resultado será guardado en el registro 'f'.

Ejemplo: DECF CNT, 1
Antes de la Instrucción
CNT = 0x01
Z =0
Después de la Instrucción
CNT = 0x00
Z =1

DECFSZ Decrementar f, saltar si es 0

Sintaxis: [etiqueta] DECFSZ f,d

Operandos: $0 \leq f \leq 127$, $d \in [0,1]$

Operación: (f) - 1 $\rightarrow$destino; saltar si el resultado = 0

Estado Afectado: Ninguno

Descripción: Los contenidos del registro 'f' se decrementan en 1, si 'd' es 0 el resultado se guarda en el acumulador W, si 'd' es 1 entonces el resultado se guarda en el registro 'f'. Si al decrementar el registro, el resultado es 0, la siguiente instrucción se ignorará llevándose a cabo en su lugar un NOP, haciéndola una instrucción de dos ciclos.

```
Ejemplo   HERE DECFSZ CNT, 1
          GOTO LOOP
          CONTINUE •
                   •
                   •
```

Existen dos posibilidades, si al decrementar el registro CNT se genera un 0, entonces no se ejecutará el GOTO, y el programa continuara en la etiqueta CONTINUE. En caso de que al decrementar el registro CNT, no sea 0 el resultado, se ejecutara el GOTO.

GOTO Salto incondicional.

Sintaxis: [etiqueta] GOTO k

Operandos: $0 \leq k \leq 2047$

Operación: k $\rightarrow$PC<10:0>, PCLATH<4:3> $\rightarrow$PC<12:11>

Estado Afectado: Ninguno

Descripción: GOTO es un salto incondicional, el valor de 11 bits 'k' es cargado en el contador de programa PC. Los bits superiores del PC, son cargados con los bits 4 y 3 del registro PCLATH. Esta es una instrucción de dos ciclos de programa.

Ejemplo GOTO THERE
Después de la Instrucción
Dirección de PC = THERE

INCF Incrementar f

Sintaxis: [etiqueta] INCF f,d

Operandos: $0 \leq f \leq 127$, $d \in [0,1]$

Operación: (f) + 1 $\rightarrow$destinación

Estado Afectado: Z

Descripción: Los contenidos del registro 'f' son incrementados en una unidad, si 'd' es 0, el resultado de la operación se guardará en W, y si 'd' es 1, el destino de la operación será el registro 'f'.

Ejemplo 1 INCF CNT, 1
Antes de la Instrucción
CNT = 0xFF
Z =0
Después de la Instrucción
CNT = 0x00
Z =1

INCFSZ Incrementar f, saltar si es 0

Sintaxis: [etiqueta] INCFSZ f,d

Operandos: $0 \leq f \leq 127$, $d \in [0,1]$

Operación: (f) + 1 →destino, saltar si el resultado es = 0

Estado Afectado: Ninguno

Descripción: Los contenidos del registro 'f' son incrementados en una unidad, si 'd' es 0, el resultado se guardará en el acumulador W, si 'd' es 1, el resultado será guardado en el registro 'f'.

```
Ejemplo   HERE INCFSZ CNT, 1
          GOTO LOOP
          CONTINUE •
```

Similarmente a la instrucción anterior, solamente que en ves de restar 1 al registro 'f', esta ves lo adicionamos, de forma que si el valor del registro es 0xff, al sumarle 1, este se volverá 0x00 y la próxima instrucción será ignorada.

IORLW 'O' inclusivo entre f y W

Sintaxis: [etiqueta] IORLW k

Operandos: $0 \leq k \leq 255$

Operación: (W).OR. k →W

Estado Afectado: Z

Descripción: Los contenidos del registro 'f' y los del acumulador W, serán comparados a través de la operación lógica 'O', y el resultado será colocado en el acumulador W.

```
Ejemplo   IORLW 0x35
          Antes de la Instrucción
          W = 0x9A
          Después de la Instrucción
          W = 0xBF
          Z =0
```

IORWF 'O' inclusivo entre W y f

Sintaxis: [etiqueta] IORWF f,d

Operandos: $0 \leq f \leq 127$, $d \in [0,1]$

Operación: (W).OR. (f) $\rightarrow$ destino

Estado Afectado: Z

Descripción: El registro 'f' y el acumulador W son comparados a través de la operación lógica 'O', los resultados de la operación se guardarán en el acumulador W.

Ejemplo IORWF RESULT, 0
Antes de la Instrucción
RESULT=0x13
W = 0x91
Después de la Instrucción
RESULT=0x13
W = 0x93
Z =0

MOVLW Mover Literal a W

Sintaxis: [etiqueta] MOVLW k

Operandos: $0 \leq k \leq 255$

Operación: k $\rightarrow$ W

Estado Afectado: Ninguno

Descripción: El acumulador W será cargado con el valor de la literal 'k' de ocho bits.

Ejemplo MOVLW 0x5A
Después de la Instrucción
W = 0x5A

MOVF Mover f

Sintaxis: [etiqueta] MOVF f,d

Operandos: $0 \leq f \leq 127$, $d \in [0,1]$

Operación: (f) $\rightarrow$ destinación

Estado Afectado: Z

Descripción: Los contenidos del registro 'f' serán movidos de acuerdo al destino deseado, si 'd' es 0, el destino será el acumulador W, si 'd' es 1 el destino de la instrucción será el registro 'f' mismo. Esta instrucción es utilizada ya sea para mover el valor de un registro a W o para comparar el registro 'f' consigo mismo, esto es útil ya que cuando 'f' es 0, se afectará el bit de estado Z al ser movido a sí mismo.

Ejemplo MOVF FSR, 0

Antes de la Instrucción

W = 0x00

FSR = 0xC2

Después de la Instrucción

W = 0xC2

Z = 0

MOVWF Mover W a f

Sintaxis: [etiqueta] MOVWF f

Operandos: $0 \leq f \leq 127$

Operación: (W) $\rightarrow$f

Estado Afectado: Ninguno

Descripción: Mueve la información contenida en el acumulador W al registro 'f'.

Ejemplo MOVWF OPTION_REG
Antes de la Instrucción
OPTION_REG=0xFF
W = 0x4F
Después de la Instrucción
OPTION_REG=0x4F
W = 0x4F

NOP Ninguna Operación

Sintaxis: [etiqueta] NOP

Operandos: Ninguno

Operación: No Operación

Estado Afectado: Ninguno

Ejemplo HERE NOP
Antes de la Instrucción
Dirección del PC = HERE
Después de la Instrucción
Dirección del PC = HERE + 1

OPTION Cargar el registro Option

Sintaxis: [etiqueta] OPTION

Operandos: Ninguno

Operación: (W) →OPTION

Estado Afectado: Ninguno

Descripción: Esta instrucción carga el contenido del acumulador W en el registro especial Option. El fabricante del controlador no recomienda el uso de esta instrucción para futura compatibilidad con sus productos, y ya que el registro Option es leíble y escribible, puede ser cargado con un MOVF por ejemplo. En el PIC16F84A, puede emplearse una u otra opción sin ningún problema.

RETFIE Regresar de una interrupción.

Sintaxis: [etiqueta] RETFIE

Operandos: Ninguno

Operación: TOS →PC, 1 →GIE

Estado Afectado: Ninguno

Descripción: Regresa de una interrupción. La dirección de 13 bits cargada en la parte superior del apilador (top of stack) se carga en nuestro contador de programa PC. El bit de interrupciones globales, GIE (INTCON 7) se activa automáticamente, activando las interrupciones. Esta instrucción consume dos ciclos de programa.

```
Ejemplo   RETFIE
          Después de la Instrucción
          PC = TOS
          GIE = 1
```

RETLW Regresar con la literal cargada en W

Sintaxis: [etiqueta] RETLW k

Operandos: $0 \leq k \leq 255$

Operación: $k \rightarrow W$; $TOS \rightarrow PC$

Estado Afectado: Ninguno

Descripción: El registro W es cargado con la literal W y la dirección en la parte superior del apilador es cargada en el contador de programa. Esta instrucción consume dos ciclos de programa.

RETURN Regresar de una subrutina.

Sintaxis: [etiqueta] RETURN

Operandos: Ninguno

Operación: $TOS \rightarrow PC$

Estado Afectado: Ninguno

Descripción: Esta instrucción termina una subrutina, la dirección guardada en la parte superior de nuestro apilador (stack) se carga en el contador de programa, es decir, regresamos a la instrucción que seguía antes de el salto.

Ejemplo HERE RETURN
Después de la Instrucción
PC = TOS (parte superior del apilador)

RLF Rotar 'f' a la izquierda a través del acarreo

Sintaxis: [etiqueta] RLF f,d

Operandos: $0 \leq f \leq 127$, $d \in [0,1]$

Operación: Rota los bits del registro hacia la izquierda.

Estado Afectado: C

Descripción: los contenidos del registro 'f' son rotados un bit hacia la izquierda a través de la bandera 'Carry', si 'd' es 0 los resultados son guardados en el acumulador W, si es 1 el resultado es guardado en el registro 'f'.

Ejemplo RLF REG1,0
Antes de la Instrucción
REG1= 1110 0110
C =0
Después de la Instrucción
REG1=1110 0110
W =1100 1100
C =1
Register f C

RRF Rotar a la Izquierda a través de la bandera de Acarreo.

Sintaxis: [etiqueta] RRF f,d

Operandos: $0 \leq f \leq 127$ $d \in [0,1]$

Operación: Rota los bits hacia la derecha

Estado Afectado: C

Descripción: Los contenidos del registro 'f' son rotados un bit hacia la derecha a través de la bandera 'Carry'. Igualmente si 'd' es 0 el resultado se colocara en el acumulador W y si es 1 en el registro 'f'.

Ejemplo RRF REG1,0
Antes de la Instrucción
REG1= 1110 0110
W = xxxx xxxx
C =0
Después de la Instrucción
REG1= 1110 0110
W = 0111 0011
C =0
Register f C

Parecería que estas dos últimas instrucciones son ideales para el control de motores Paso a Paso, ya que en estos se trata de rotar los campos, pero debido a la configuración de medio paso que empleamos, y a que los dos motores de cada controlador se encontrarán en el puerto B, no nos resulta conveniente, así que utilizaremos un sistema de banderas así como la activación y desactivación de los bits individualmente.

SLEEP

Sintaxis: [etiqueta] SLEEP

Operandos: Ninguno

Operación: 00h → WDT, 0 → WDT cuenta del preescalador,1→ TO, 0 → PD

Estado Afectado: TO, PD

Descripción: El bit de estado PD es borrado. El bit de estado de Time out TO se activa, el temporizador de perro guardián, WDT, y su preescalador son borrados. El CPU entra en un estado de reposo y el oscilador se detiene.

Ejemplo: SLEEP

SUBLW Sustraer W de la Literal

Sintaxis: [etiqueta] SUBLW k

Operandos: $0 \leq k \leq 255$

Operación: k - (W) → W

Estado Afectado: C, DC, Z

Descripción: El acumulador W es restado de la literal 'k' y el resultado es colocado de vuelta en el acumulador W.

Ejemplo: SUBLW 0x02

Antes de la Instrucción

W = 0x01

C =x

Z =x

Después de la Instrucción

W = 0x01

C = 1; El resultado es positivo.

Z =0

SUBWF Sustraer W de f

Sintaxis: [etiqueta] SUBWF f,d

Operandos: $0 \leq f \leq 127$ $d \in [0,1]$

Operación: (f) - (W) $\rightarrow$ destino

Estado Afectado: C, DC, Z

Descripción: Se resta el valor contenido en el acumulador W al registro 'f', y el resultado es guardado en W si 'd' es 0 y en el mismo registro 'f' si 'd' es 1.

Ejemplo 1: SUBWF REG1,1

Antes de la Instrucción

REG1= 3

W =2

C =x

Z =x

Después de la Instrucción

REG1= 1

W =2

C = 0 ; El resultado es negativo.

Z =0

SWAPF invertir bits en f

Sintaxis: [etiqueta] SWAPF f,d

Operandos: $0 \leq f \leq 127$, $d \in [0,1]$

Operación: (f<3:0>) $\rightarrow$ destinación <7:4>, (f<7:4>) $\rightarrow$ destinación<3:0>

Estado Afectado: Ninguno

Descripción: Los bits mas significativos del registro 'f' son intercambiados con los bits menos significativos, y el resultado se guarda en W si 'd' es 0 y en el registro 'f' si d'd es 1.

Ejemplo SWAPF REG, 0

Antes de la Instrucción

REG1= 0xA5

Después de la Instrucción

REG1= 0xA5

W = 0x5A

TRIS cargar el registro TRIS

Sintaxis: [etiqueta] TRIS f

Operandos: $5 \leq f \leq 7$

Operación: (W) → registroTRIS f;

Estado Afectado: Ninguno

Descripción: Esta instrucción carga el valor contenido en el acumulador W, pero no se recomienda su uso para prevenir futura compatibilidad con los productos de Microchip®. Aún así, hay que destacar, que es la manera más fácil de inicializar los puertos.

XORLW O exclusivo entre W y la literal

Sintaxis: [etiqueta] XORLW k

Operandos: $0 \leq k \leq 255$

Operación: (W).XOR. k → W

Estado Afectado: Z

Descripción: Los contenidos del acumulador W son comparados a través de la operación lógica 'O' negada y el resultado es colocado en W.

```
Ejemplo: XORLW 0xAF          ; 1010 1111 (0xAF)
     Antes de la Instrucción ; 1011 0101 (0xB5)
     W = 0xB5                ; --------- ------
     Después de la Instrucción ; 0001 1010 (0x1A)
     W = 0x1A
     Z =0
```

XORWF O exclusivo entre W y f

Sintaxis: [etiqueta] XORWF f,d

Operandos: $0 \leq f \leq 127$, $d \in [0,1]$

Operación: (W).XOR. (f) $\rightarrow$ destinación

Estado Afectado: Z

Descripción: Los contenidos del acumulador W y el registro 'f' son comparados a través de la operación lógica 'O' negada, los resultados se colocan en el registro W si 'd' es 0 y en el registro 'f' mismo si 'd' es 1.

```
Ejemplo: XORWF REG, 1          ; 1010 1111 (0xAF)
         Antes de la Instrucción ; 1011 0101 (0xB5)
         REG= 0xAF             ; --------- ------
         W = 0xB5              ; 0001 1010 (0x1A)
         Después de la Instrucción
         REG= 0x1A
         W = 0xB5
```

BIBLIOGRAFÍA.

REFERENCIAS BIBLIOGRAFÍAS

1. James L. Fuller, 1991. Robotics: Introduction, programming and projects.
2. José Maria Angulo Usategui, 1986. Robótica práctica: Tecnología y aplicaciones.
3. José Maria Angulo Usategui, 1985. Curso de robótica.
4. Kost Gerald J. & Welsh Judith, 1996. Handbook of clinical automation robotics and aplications.
5. D. A. Bradley, 1991. Mechatronics: Electronics in product and processes.
6. Mc Cloy Don, Michael Harris, 1992. Robótica, una introducción.
7. Jim Cox, 1995. Maquinas eléctricas.
8. Luis Reyes Ávila, 1990. Unite de Recherche INRIA-ROCQUENCOURT: QUATERNIONS UNE REPRESENTATION PARAMETRIQUE SYSTEMATIQUE DES ROTATIONS FINIES.
9. Luis Reyes Ávila, 1991. Unite de Recherche INRIA-ROCQUENCOURT: QUATERNIONS UNE REPRESENTATION PARAMETRIQUE SYSTEMATIQUE DES ROTATIONS FINIES.Partie II.
10. Robert E. Parkin, 1984. Aplied Robotic Analysis.
11. Jose Armando Cárdenas, 1980. Diseño de maquinas.
12. Pollone Giuseppe, 1985. Engranajes.
13. Saorayana Hajine, 1990. Una visión a la robótica (diseño de robots).
14. López Pierre, 1987. Introducción a la robótica I: Enseñanza e investigación y desarrollo.
15. López Pierre, 1987. Introducción a la robótica II: Comunicación hombre maquina, programación y control.
16. Neil Schmitt M., 1988. A fondo: Robótica y sistemas automáticos.
17. Robert L. Norton, Diseño de Máquinas, Mc Graw Hill

18. Curso de Programación de Visual Basic®. Francisco Javier Cevallos. Alfa Omega Editores.
19. Autómatas programables. Joseph Bacells. Editorial Marcombo. 1997.
20. Fundamentos de Robótica. Antonio Barrientos. Mc Graw Hill. 1997.
21. Cómo y cuando aplicar un Robot Industrial. Daniel Audi Piera