Universidad Autónoma de Querétaro

Facultad de Ingeniería

Sistema de Control para Robots Industriales

Tesis

Que como parte de los requisitos
para obtener el grado de

Maestro en ciencias con línea terminal en instrumentación y control automático

Presenta

Miguel Angel Martínez Prado

Querétaro, Qro., Octubre de 2007

Universidad Autónoma de Querétaro
Facultad de Ingeniería
Ingeniería en Automatización

Sistema de Control para Robots Industriales **TESIS**

Que como parte de los requisitos para obtener el grado de
Maestro en Ciencias con Línea Terminal en Instrumentación y Control Automático

Presenta:
Miguel Angel Martínez Prado **Dirigido por:**

Dr. Gilberto Herrera Ruiz

SINODALES

Dr. Gilberto Herrera Ruiz
Presidente — Firma

Dr. Rodrigo Castañeda Miranda
Secretario — Firma

Dr. Roque Alfredo Osornio Rios
Vocal — Firma

Dr. Luis Alfonso Franco Gasca
Suplente — Firma

M. en C. José Angel Colín Robles
Suplente — Firma

Dr. Gilberto Herrera Ruiz
Director de la Facultad de Ingeniería

M. en C. Manuel Toledano Ayala
Coordinador de Ingeniería en Automatización.

Centro Universitario
Querétaro, Qro. Febrero de 2007 México

Resumen

La introducción de robots manipuladores a las líneas de producción de las empresas representa un avance tecnológico significativo en el área de la robótica y el control automático. Sin embargo, la mayoría de las empresas que utilizan esta tecnología son empresas de gran poder adquisitivo, casi en totalidad la industria del automóvil. El objetivo del presente trabajo es el diseño y construcción de un sistema de control para robots industriales. Este sistema de control basado en computadora debe ser económico, reprogramable, flexible, robusto, confiable y fácil de operar. Algunas de las principales tareas son encontrar el modelo cinemático del robot FANUC S 420-F, como principal objetivo; diseño de un compilador para un lenguaje de programación de robots, diseño e implementación de software y el diseño e integración de hardware.

Palabras clave: robot, compilador, software.

Summary

The introduction of manipulator robots to the production lines of the companies represents a technological advance in the robotic and automatic control fields. However, most of these companies who used this technology they are companies of huge economical power, almost all them are the automotive industries. The target of this work is designing and to build a control system for industrial robots. This control system based on computer must be economical, reprogrammable, flexible, robust, reliable and easy to operate. Some of the tasks in this work are to get the kinematic model of the robot FANUC S 420-F, is the robot in which we are interested, design of a compiler for a robot programming language, design and implementation of software and design and integration of hardware.

Keywords: robot, compiler, software.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Descripción del problema

Debido a la continua competencia comercial que ha introducido la globalización en los mercados de todo el mundo, el sector privado demanda nuevas y más rápidas técnicas de manufactura. Las empresas han entrado en un juego de competitividad apostando ya sea por mano de obra barata o por sistemas automáticos de producción.

La calidad juega un gran papel en nuestro entorno globalizado. Es gran menester para una empresa la satisfacción total de sus clientes. Esto conlleva constante evaluación, desarrollo e innovación. Solo aquellas empresas que sean competitivas habrán de sobrevivir, y por supuesto, aquellas que no lo sean estarán destinadas a desaparecer.

En México, son pocas las empresas competitivas. Estamos rodeados de marcas extrangeras; grandes empresas líderes en su ámbito. Y por si fuera poco con tal competencia, nuestra industria enfrenta otro enorme problema: los mercados asiáticos. Dichas empresas ofrecen productos de muy bajo precio, artículos pobres en calidad y de todos tipos. Sin embargo, es bien conocido que son quizá de lo más adquirido por los consumidores.

Nuestro país requiere de empresas nacionales que aporten recursos a nues-

tra economía. En virtud de la naturaleza de la globalización, nuestro bienestar económico es función de economías de otros países. De aquí surge la pregunta ¿Cómo hacer que las pequeñas y medianas empresas nacionales sean competitivas?. La respuesta es tal vez una combinación de muchas necesidades. Una en particular, y es la que como ingenieros nos concierne, es proveer a las empresas de nuevas y mejores alternativas de producción automática.

Actualmente existe gran número de empresas dedicadas al desarrollo de soluciones de automatización. Bien podría una empresa buscar los productos que incrementen su calidad. Sin embargo, estamos tratando con empresas que están iniciando o apenas emprendiendo; la adquicición de estas tecnologías está fuera de su alcance. Estamos hablando por ejemplo, de una celda robotizada la cual, puede costar hasta 120,000 dólares. Tal vez para el lector no parezca esa cantidad algo descabellado, pero si a eso le agregamos la capacitación, accesorios y mantenimiento, quizá ya no sea una cantidad despreciable. El problema no es si existen o no las soluciones para incrementar la productividad de una empresa, el problema es lo lejana que está su adquisición por empresas de bajos recursos.

1.2. Antecedentes

La robótica es una ciencia relativamente moderna. Está constituida por diciplinas como las Matemáticas, Electrónica, Computación y Mecánica. A pesar de que la aparición del primer robot industrial no se dió sino hasta el siglo XX, las bases de esta ciencia se remontan a los inicios del estudio mecánico de cuerpos rígidos.

La robótica es un tema de estudio constante y progresivo en todo el mundo. El uso de robots se extiende desde aplicaciones médicas hasta operaciones dentro de invernaderos.

Un ejemplo claro de lo anterior es el trabajo titulado “Robot design and testing for greenhouse applications” por G. Belforte, R. Deboli y P. Piccarolo

en 2007. Aquí se presenta el diseño y desarrollo de un robot acorde a las necesidades agrícolas dentro de un invernadero. En virtud de lo anterior, podemos apreciar la funcionalidad, es decir, la gama de tareas que un robot puede realizar no está sujeta al número de grados de libertad. Esta alternativa de producción reduce ampliamente costos de operación, sin embargo, es ligeramente más lenta en comparación con el trabajo de un operador humano. Además, a jucio propio, existen otras soluciones de automatización un tanto más rústicas, pero mucho más eficientes y de menor costo [Belforte, 2006].

Otro trabajo de interés en nuestro tema se titula "Modeling and identification of an industrial robot for machining applications" por E. Abele, M. Weigold y S. Rothenbücher en 2007. En este trabajo se presenta el modelo cinemático obtenido de un robot de seis grados de libertad, siendo el sexto intercambiado por husillo con un motor de alta velocidad. El problema que involucra que un robot realice tareas de maquinado radica en la precisión que se requiere. Tomando en cuenta que el diseño de uniones para robots presentan tolerancias de error y que esas tolerancias están pensadas en escenarios ideales de operación nos hace imposible imaginar que un robot pueda realizar estas tareas. En este trabajo se presenta una solución simple para solucionar este problema, la idea es la calibración compensada del robot, es decir, cuantificar el error que tiene cada articulación para ajustar sus parámetros [Abele, 2007].

Desde la consolidación de robots industriales como alternativas de operación económicas, eficientes, robustas y flexibles, y el estudio de sus arquitecturas, los aspectos de interés y estudio han sido hasta nuestros días proveer a estas máquinas con toda la información y herramientas necesarias para realizar su trabajo más rápido, preciso y seguro.

Un ejemplo de ello es el trabajo "A robotic system for road lane painting" de SangkyunWoo, Daehie Hong,Woo-Chang Lee, Jae-Hun Chung y Tae-Hyung Kim en 2007. Este trabajo trata el problema de repintar las líneas en las carreteras usando un robot móvil. Su importancia se centra en el uso

de visión artificial para la detección de intervalos que requieran de pintura. Su utilidad contra la complejidad del proyecto es alta debido a que el robot móvil solo requiere de una cámara para realizar su trabajo. Implementa algunos filtros bastante conocidos como el umbral[1] y el detector de bordes Sobels[2] para reconocer las líneas blancas del camino [Woo, 2006].

En 2007 se publicó a través de Elsevier Ltd el trabajo titulado "A kinematic calibration method for industrial robots using autonomous visual measurement" de los autores A. Watanabe1, S. Sakakibara, K. Ban, M. Yamada y G. Shen. Aquí se propone un nuevo método de calibración para robots industriales fuera de línea basado en mediciones visuales, es decir, el órgano terminal del robot es acompañado por una cámara digital que proporciona imágenes del objetivo. Aunque si bien es cierto que el uso de cámaras para la medición de objetos no es nueva, lo es la idea de su uso para la realimentación de posición de un robot industrial [Watanabe, 2007].

El trabajo titulado "Design and hybrid control of the pneumatic force-feedback systems for Arm-Exoskeleton by using on/off valve" presenta el modelo neumático de un sistema con realimentación de fuerzas que consiste de cilindros de doble efecto, un conjunto de válvulas on-off y un controlador de tipo fuzzy[3]. Para el modelado se consideran los efectos no lineales del flujo a través de las válvulas, compresibilidad del aire en las cámaras de los cilindros, el retardo y atenuación de la presión de entrada [Chen, 2007].

El trabajo "Application of genetic programming to the calibration of industrial robots" introduce una nueva técnica de calibración cinemática basada en programación genética, la cual es usada para establecer el modelo estructural del robot y sus parámetros. La principal ventaja de esta ténica es que no está limitada a modelos estructurales predefinidos y por lo tanto tiene la

[1]Operación de una imagen. Convierte la imagen a su forma binaria.

[2]Detector de bordes de imágenes. Se basa en el gradiente de los valores de los pixeles de la imagen.

[3]Control basado en la pertenencia de un elemento a múltiples conjuntos con cierto grado de pertenencia.

capacidad de producir un modelo de calibración más completo y preciso que los métodos convencionales [Dolinsky, 2007].

El trabajo "A mechatronic approach for robotic deburring" trata el diseño y la implementación de un algoritmo de control basado en la fuerza y velocidad de una herramienta para el desbaste de piezas planas con forma desconocida. Su implementación se realiza sobre un manipulador de arquitectura SCARA con un software para el la detección de la forma de la pieza a partir su fuerza normal y velocidad tangencial [Ziliani, 2007].

El trabajo titulado "Kinematic aspects of a robot-positioner in an arc welding application" se enfoca al control cinemático de un sistema robotizado de soldadura. Aquí se propone una nueva técnica de orientación del extremo final del robot y del manipulador de la pieza para la optimización de sus movimientos y reducción de los efectos de la gravedad [Pashkevich, 2002].

El trabajo "Automatic model generation and PLC-code implementation for interlocking policies in industrial robot cells" propone un método que hace uso de la información de un entorno simulado del robot para extraer automaticamente un modelo de su estado. Este modelo puede ser usado para generar un supervisor para asegurar que las situaciones de colisión qeu pudiesen ocurrir debido a la agregación de objetos dentro del espacio de trabajo del robot puedas ser evitados y También es posible reducir el tiempo ciclo de trabajo del robot [Flordal, 2002].

En el trabajo "Lifted system iterative learning control applied to an industrial robot" se propone un algoritmo de control de aprendizaje iterativo para sistemas variantes en el tiempo con una alta velocidad de convergencia. Aunque el trabajo desde el punto de vista teórico es de gran relevancia, su implementación se hace usando una aproximación lienal del modelo del robot. Esto es, el controlador requiere de un modelo lineal. Sin embargo, la mayoria de los sistemas en nuestro entorno son de carácter no lineal [Hakvoort, 2007].

	Instalaciones anuales			
Ciudad	**2004**	**2005**	**2006**	**2009**
América	**15,400**	**21,550**	**17,200**	**20,100**
Brazil	208	320		
USA, México, Canada	15,170	21,136	16,500	19,100
Otras	22	99		

Tabla 1.1: Nuevas instalaciones en América según la Federación Internacional de Robótica.

1.3. Justificación

El mercado mundial de robots alcanzó en 2005 cerca de 126,700 unidades nuevas instaladas; 30 % más que en 2004. Este es el valor más alto alcanzado en un año. Sin embargo, este desarrollo se ha dado de forma distinta en las regiones industrializadas de Europa, Asia y América.[4]

La industria del automóvil ha sido un factor determinante en dicho desarrollo. Los sectores de componentes electrónicos y de comunicaciones han beneficiado en gran medida la investigación en este campo.

En cuanto toca a México, Argentina y Brazil la industria del automóvil instaló en 2005, 47 % más robots industriales que en 2004. Muchas otras industrias también han incrementado sus instalaciones notablemente. La demanda de la industria del metal (incluyendo maquinaria, productos y metáles básicos) se incrementó al 52 %, la industria química al 41 % y la industria eléctrica-electrónica al 32 %. Se estima que el número total de unidades instaladas en todo el mundo se incrementará de aproximadamente 922,900 unidades al término de 2005 a 1'112,500 unidades al término de 2009, lo que representa un promedio anual de 4.9 %. [5]

Querétaro es un estado con una extensión territorial de 11,688 km^2, un

[4]Federación Internacional de Robótica. www.ifr.org

[5]IFR Statistical Department. www.worldrobotics.org

Sector	**Porcentaje**
Metal-mecánica y autopartes	36.3 %
Alimentos y bebidas	29.1 %
Química, caucho y plástico	19.5 %
Papel, imprenta y sus derivados	7.0 %
Minerales no metálicos	4.1 %
Textiles	2.0 %
Madera	0.9 %
Otros	0.5 %
Metal básico	0.4 %

Tabla 1.2: Sectores de empresas en el estado por la Secretaría de Desarrollo Sustentable.

producto interno bruto (PIB) de 13.6 BUSD, ingreso per cápita de 8,617 USD y un crecimiento en el sector de la industria manufacturera de 4.1 % en 2006.[6]

El sector metal-mecánico y de autopartes conforma el 36.3 % de la industria en nuestra entidad. Algunas otras como alimentos y bebidas cubren el 29.1 %, química, caucho y plásticos el 19.5 %, papel, imprenta y sus derivados el 7 %, minerales no metálicos el 4.1 %, textiles el 2 %, madera el 0.9 % y metálica básica con el 0.4 %.

En Querétaro existen más de 233 empresas del sector del automóvil, 3 ensambladoras (Irizar, Man Ferrostal y Scania) y más de 150 proveedores de autopartes. Nombres como ArvinMeritor, Condumex, Dana, Eaton, Kostal, Ronal, VRK, TRW y Bosal, entre otros, son bastante conocidos por el desarrollo que han tenido y el beneficio aportado a la entidad con la generación de empleos.[7]

Actualmente, empresas como Fanuc, Motoman, ABB y Panasonic se ded-

[6]Banamex, Secretaría de Desarrollo Sustentable, SIREM, INEGI.

[7]Secretaría de Desarrollo Sustentable.

ican a la construcción e implementación de celdas robotizadas. Sus productos proveen una solución flexible a problemas de automatización como ensamble, pintura, paletizado, manipulación y otros, además de su gran robustez. Sin embargo, una solución de esta índole se cotiza alrededor de los 75,000 y 100,000 USD.

En un contexto local, a nivel entidad, nos encontramos con una gran cantidad de empresas de pequeñas dimensiones. Muchas de estas empresas no poseen los recursos suficientes para la implementación de un celda robotizada como las ofrecidas por las empresas antes mencionadas.

El desarrollo de las PYMES (pequeñas y medianas empresas) recae en gran medida al desarrollo tecnológico de centros de investigación locales que provean soluciones mucho más económicas y flexibles, manteniendo un compromiso con la calidad, funcionalidad y robustez como los sistemas comerciales.

1.4. Objetivos

Proveer al sector privado de menos recursos con la tecnología apropiada para hacer a la pequeña y mediana industria más competitiva. La idea es incrementar la calidad y productividad en sus líneas de operación y decrementar los tiempos y sus costos.

Esto nos lleva al diseño de un sistema de control flexible para robots manipuladores industriales. Este sistema de control basado en computadora deberá ser reprogramable, flexible, robusto, confiable y fácil de operar.

1.5. Hipótesis

Es factible y sostenible la construcción de un sistema de control para robots industriales económico, flexible, funcional, confiable y robusto como una solución alternativa a los sistemas comerciales existentes en el mercado.

Capítulo 2

Fundamentación teórica

2.1. Cinemática de un cuerpo rígido

La *cinemática* es la ciencia del movimiento que lo estudia sin importar las fuerzas que lo provocan. La cinemática estudia la posición, velocidad, aceleración, y todas las derivadas posteriores de las variables de posición (con respecto al tiempo o a otras variables). Por lo tanto, el estudio de la cinemática de los manipuladores se refiere a toda la geometría y propiedades del movimiento en base al tiempo.

Los manipuladores consisten de *eslabones* rígidos los cuales son conectados por uniones que permiten el movimiento relativo a sus eslabones vecinos. Estas uniones son usualmente instrumentadas con sensores de posición los cuales permiten la medición de la posición relativa a sus eslabones vecinos. En el caso de uniones rotatorias o *revolutas*, esos desplazamientos son llamados *ángulos de unión*. Algunos manipuladores contienen deslizadores, o uniones *prismáticas* en las cuales el desplazamiento relativo entre eslabones es traslacional.

El número de *grados de libertad* que un manipulador posee es el número de variables de posición independientes, las cuales deberían ser especificadas en orden para la localización de todas las partes del mecanismo. Este es

un término general usado para cualquier mecanismo. En el caso de robots industriales típicos, dado que un manipulador es usualmente una cadena cinemática abierta (es decir, los eslabones no forman un lazo), y cada unión es usualmente definida con una única variable, el número de uniones es igual al número de grados de libertad.

En el estudio de los robots, estamos constantemente relacionados con el estudio de objetos en el espacio tridimensional. Esos objetos son los eslabones del manipulador, las partes y herramientas con las cuales interactúa, y otros objetos en el entorno del manipulador. En un burdo pero importante nivel, esos objetos son descritos con solo dos atributos: su *posición* y *orientación*. Naturalmente, un tema de interés inmediato es la manera en la cual representamos esas cantidades y las manipulamos matemáticamente.

Primeramente, para describir la posición y orientación de un cuerpo en el espacio siempre fijaremos un *sistema coordenado*, o *marco de referencia*, de forma rígida al objeto. Entonces, procedemos a describir la posición y orientación de su marco con respecto a algún otro sistema coordenado fijo, al cual llamaremos *sistema coordenado universal*.

Dado que cualquier marco de referencia puede servir como un sistema de referencia, dentro del cual se expresa la posición y la orientación de un cuerpo, frecuentemente nos encontramos en la necesidad de transformar o cambiar la descripción de esos atributos de un cuerpo de un marco a otro con la intención de modelar su movimiento.

Cinemática directa de los manipuladores

El problema de la cinemática directa consiste en que dados los valores de las variables de unión del manipulador, determinar la posición y orientación del órgano terminal. Las varibles de unión son los ángulos entre los eslabones para los robots con uniones rotatorias.

Cinemática inversa de los manipuladores

Este problema es presentado como sigue: Dada la posición y orientación del órgano terminal del manipulador, se debe calcular todos los conjuntos posibles de ángulos de las uniones los cuales podrían ser usados para alcanzar la posición y orientación deseada. Este es el problema fundamental en el uso práctico de manipuladores.

El problema de la cinemática inversa no es tan simple como el de la cinemática directa en virtud de que las ecuaciones cinemáticas son no lineales, su solución no siempre es fácil o inclusive posible en una forma cerrada. También, la pregunta de existencia de una solución, y de múltiples soluciones presentes.

La existencia o no existencia de una solución cinemática define el espacio de trabajo de un manipulador. La falta de una solución significa que el manipulador no puede lograr la posición y orientación deseada porque se encontraría fuera de su espacio de trabajo.

2.1.1. Descripción de un cuerpo y transformaciones en el espacio

Los manipuladores, por definición, implican que partes y herramientas serán movidas alrededor de un espacio por alguna clase de mecanismo. Esto naturalmente conduce a la necesidad de representar posiciones y orientaciones de las partes, herramientas, y del mecanismo por sí mismo. Para definir y manipular cantidades matemáticas las cuales representan la posición y orientación debemos definir sistemas coordenados y desarrollar convenciones para su representación.

Describiremos todas las posiciones y orientaciones con respecto al sistema coordenado universal o con respecto a otros sistemas coordenados cartesianos los cuales son (o podrían ser) definidos relativos al sistema coordenado universal.

Proyecciones

La proyección de un vector de posición $[\mathbf{p}]$, denotada por $[\mathbf{p}]\prime$ sobre un plano Π de norma unitaria $[\mathbf{n}]$ es el mismo $[\mathbf{p}]$ menos la componente de $[\mathbf{p}]$ a lo largo de $[\mathbf{n}]$, esto es,

$$[\mathbf{p}]\prime = [\mathbf{p}] - [\mathbf{n}]([\mathbf{n}]^T[\mathbf{p}]). \tag{2.1}$$

Donde el superíndice T denota a cualquier vector o matriz su transposición y $[\mathbf{n}]^T[\mathbf{p}]$ es equivalente al usual producto punto $[\mathbf{n}] \cdot [\mathbf{p}]$. La matriz identidad 1 es definida como el mapeo de un espacio vectorial $\mathcal{V}$ a si mismo dejando cualquier vector $[\mathbf{v}]$ sin cambios, esto es,

$$1[\mathbf{v}] = [\mathbf{v}]. \tag{2.2}$$

De esta forma se tiene

$$[\mathbf{p}]\prime = 1[\mathbf{p}] - [\mathbf{n}][\mathbf{n}]^T[\mathbf{p}] = (1 - [\mathbf{n}][\mathbf{n}]^T)[\mathbf{p}], \tag{2.3}$$

y por lo tanto, la proyección ortogonal P sobre Pi puede ser representada como

$$P = 1 - [\mathbf{n}][\mathbf{n}]^T, \tag{2.4}$$

donde el producto $[\mathbf{n}][\mathbf{n}]^T$ resulta una matriz de 3×3.

Reflexiones

Aquí tenemos que tomar en cuenta que las reflexiones ocurren frecuentemente acompañadas por rotaciones, que aún no hemos estudiado. Podríamos empezar por estudiar solamente las reflexiones, esto es, aquellas que ocurren sin ninguna rotación.

Una reflexión R del espacio vectorial ε^3 sobre el plano Pi pasando a través del origen y teniendo una unidad normal n, es una transformación de dicho

espacio a si mismo, tal que un vector de posición [**p**] es mapeado por R a un vector [**p**]′ dado por

$$[\mathbf{p}]' = [\mathbf{p}] - 2[\mathbf{n}][\mathbf{n}]^T[\mathbf{p}] \equiv (1 - 2[\mathbf{n}][\mathbf{n}]^T)[\mathbf{p}]. \tag{2.5}$$

Así, la reflexión R puede ser expresada como

$$R = 1 - 2[\mathbf{n}][\mathbf{n}]^T. \tag{2.6}$$

Al observar más a detalle una reflexión pura encontramos una propiedad interesante

$$R = R^{-1}. \tag{2.7}$$

Además, al tomar la descomposición ortogonal de un vector [**v**] dado en sus dos componentes, una a lo largo y una normal al vector unitario [**e**]. La componente de [**v**] a lo largo de [**e**], determinada como la componente axial, $[\mathbf{v}]_{\parallel}$ es dada simplemente como

$$[\mathbf{v}]_{\parallel} = [\mathbf{e}][\mathbf{e}]^T[\mathbf{v}]. \tag{2.8}$$

Mientras la correspondiente componente normal, $[\mathbf{v}]_{\perp}$ es simplemente la diferencia

$$[\mathbf{v}]_{\perp} = [\mathbf{v}] - [\mathbf{v}]_{\parallel} \equiv (1 - [\mathbf{e}][\mathbf{e}]^T)[\mathbf{v}]. \tag{2.9}$$

Rotación de un cuerpo rígido

Un isomorfismo lineal, esto es, una transformación uno a uno que mapea de un espacio vectorial V a sí mismo, es llamado una *isometría* si se preservan las distancias entre cualquiera dos puntos de V. Si [**u**] y [**v**] son dos vectores de posición correspondientes a dos puntos, la distancia d entre estos dos puntos es definida como

$$d = \sqrt{([\mathbf{u}] - [\mathbf{v}])^T([\mathbf{u}] - [\mathbf{v}])}. \tag{2.10}$$

El volumen V de un tetaedro definido por el origen y tres puntos de un espacio tridimensional Euclidiano de vectores de posición [**u**], [**v**] y [**w**] es obtenido como un sexto del valor absoluto del doble producto mixto de esos tres vectores

$$\mathcal{V} \equiv \frac{1}{6}|[\mathbf{u}] \times [\mathbf{v}] \cdot [\mathbf{w}]| = \frac{1}{6}|\det [[\mathbf{u}][\mathbf{v}][\mathbf{w}]]|. \tag{2.11}$$

Esto es, si se define un arreglo de 3×3 [**A**] en términos de las componente de [**u**], [**v**] y [**w**], en una base dada, entonces la primera columna de [**A**] es dada por las tres componentes de [**u**], la segunda y tercera columnas de forma análoga.

Ahora, sea Q un mapeo isométrico de la triada $\{[\mathbf{u}], [\mathbf{v}], [\mathbf{w}]\}$ a $\{[\mathbf{u}]\prime, [\mathbf{v}]\prime, [\mathbf{w}]\prime\}$. Por otra parte, las distancias del origen a los puntos de los vectores de posición [**u**], [**v**] y [**w**] son dadas simplemente como $\|[\mathbf{u}]\|$, $\|[\mathbf{v}]\|$ y $\|[\mathbf{w}]\|$, las cuales son definidas como

$$\|[\mathbf{u}]\| = \sqrt{[\mathbf{u}][\mathbf{u}]^T} \qquad \|[\mathbf{v}]\| = \sqrt{[\mathbf{v}][\mathbf{v}]^T} \qquad \|[\mathbf{w}]\| = \sqrt{[\mathbf{w}][\mathbf{w}]^T}, \tag{2.12}$$

claramente

$$\|[\mathbf{u}]\| = \|[\mathbf{u}]^T\| \qquad \|[\mathbf{v}]\| = \|[\mathbf{v}]^T\| \qquad \|[\mathbf{w}]\| = \|[\mathbf{w}]^T\|, \tag{2.13}$$

y

$$\det [[\mathbf{u}]\prime[\mathbf{v}]\prime[\mathbf{w}]\prime] = \det [[\mathbf{u}][\mathbf{v}][\mathbf{w}]]. \tag{2.14}$$

Si, en las relaciones anteriores, el signo del determinante es preservado, la isometría representa una rotación, si no, representa una reflexión. Ahora,

sea $[\mathbf{p}]$ el vector de posición de cualquier punto en ε^3, su imagen bajo una rotación $[\mathbf{Q}]$ es $[\mathbf{p}]\prime$. Por lo tanto, la preservación de la distancia requiere que

$$[\mathbf{p}]^T[\mathbf{p}] = [\mathbf{p}]\prime^T[\mathbf{p}]\prime, \tag{2.15}$$

donde

$$[\mathbf{p}]\prime = [\mathbf{Q}][\mathbf{p}]. \tag{2.16}$$

La matriz producto cruz

Sean $[\mathbf{v}]$ y $[\mathbf{x}]$ dos vectores en ε^3, entoncés el producto cruz de ambos vectores, denotado por $[\mathbf{v}] \times [\mathbf{x}]$ está dado como

$$[\mathbf{v}] \times [\mathbf{x}] = \det \begin{vmatrix} i & j & k \\ [\mathbf{v}]_1 & [\mathbf{v}]_2 & [\mathbf{v}]_3 \\ [\mathbf{x}]_1 & [\mathbf{x}]_2 & [\mathbf{x}]_3 \end{vmatrix} = \begin{bmatrix} [\mathbf{v}]_2[\mathbf{x}]_3 - [\mathbf{v}]_3[\mathbf{x}]_2 \\ -[\mathbf{v}]_1[\mathbf{x}]_3 + [\mathbf{v}]_3[\mathbf{x}]_1 \\ [\mathbf{v}]_1[\mathbf{x}]_2 - [\mathbf{v}]_2[\mathbf{x}]_1 \end{bmatrix}, \tag{2.17}$$

de donde podemos agrupar los términos de $[\mathbf{x}]$ premultiplicados por la *matriz producto cruz* $[\mathbf{V}]$, es decir

$$[\mathbf{v}] \times [\mathbf{x}] = \begin{bmatrix} 0 & -[\mathbf{v}]_3 & [\mathbf{v}]_2 \\ [\mathbf{v}]_3 & 0 & -[\mathbf{v}]_1 \\ -[\mathbf{v}]_2 & [\mathbf{v}]_1 & 0 \end{bmatrix} \begin{bmatrix} [\mathbf{x}]_1 \\ [\mathbf{x}]_2 \\ [\mathbf{x}]_3 \end{bmatrix} \tag{2.18}$$

La matriz producto cruz $[\mathbf{V}]$ de cualquier vector tridimensional es antisimétrica, esto es,

$$[\mathbf{V}]^T = -[\mathbf{V}], \tag{2.19}$$

y como consecuencia

$$v \times (v \times x) = [\mathbf{V}]^2[\mathbf{x}]. \tag{2.20}$$

Donde $[\mathbf{V}]^2$ se puede demostrar fácilmente al ser

$$[\mathbf{V}]^2 = -\|[\mathbf{v}]\|^2 1 + [\mathbf{a}][\mathbf{a}]^T, \tag{2.21}$$

donde $\|\cdot\|$ denota la *norma euclidiana*.

La matriz de rotación

En apartados previos se habló de la matriz de rotación como una representación de un marco de referencia en términos de otro. A continuación se estudiará más a fondo dicha matriz.

Teorema 1 (Euler, 1776) *El movimiento de un cuerpo rígido en torno a un punto O deja fijo un conjunto de puntos sobre una línea $\mathcal{L}$ que pasa sobre O y es paralelo al eigenvector* $[\mathbf{e}]$ *de* $[\mathbf{Q}]$ *asociado con el eigenvalor* $+1$.

Para la representación de una matriz de rotación, debemos tener en cuenta el teorema anterior, el cual sugiere que es posible una representación explícita de $[\mathbf{Q}]$ en términos de su eigenvector $[\mathbf{e}]$. Por otra parte, está representación debe contener información sobre el resultado de la rotación bajo estudio, la cual no es otra cosa que el ángulo de rotación. Además, la línea $\mathcal{L}$, es determinada como el eje de rotación del movimiento de interés. Para analizar la representación antes mencionada, considere la rotación presentada en la figura 2.1 de un ángulo ϕ sobre la recta $\mathcal{L}$.

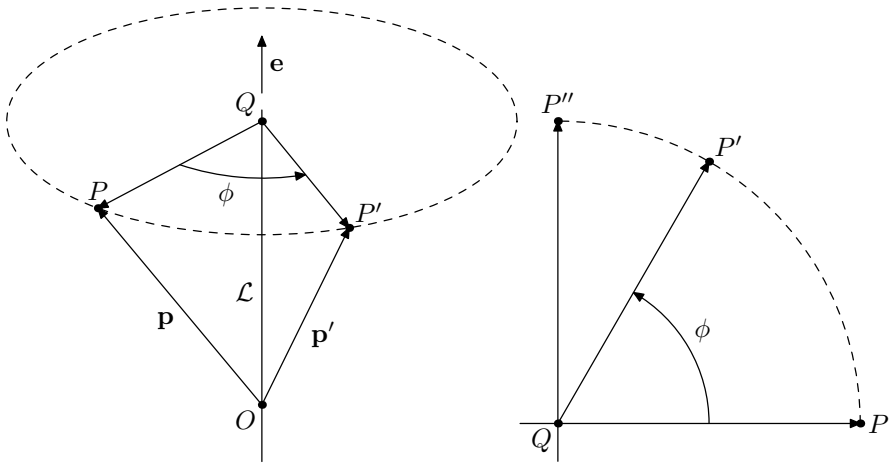
Claramente, se puede escribir

$$[\mathbf{p}]\prime = \overrightarrow{OQ} + \overrightarrow{OP\prime}, \tag{2.22}$$

donde $\overrightarrow{OQ}$ es la componente axial de $[\mathbf{p}]$ a lo largo de $[\mathbf{e}]$, representada como

$$\overrightarrow{OQ} = [\mathbf{e}][\mathbf{e}]^T[\mathbf{p}]. \tag{2.23}$$

Además, de la figura 2.1(b)



(a) Vista en perspectiva (b) Vista superior

Figura 2.1: Rotación de ϕ radianes en torno al vector [**e**].

$$\overrightarrow{QP\prime} = \cos\phi\overrightarrow{QP} + \sin\phi\overrightarrow{QP\prime\prime}. \tag{2.24}$$

Con $\overrightarrow{QP}$ siendo la componente normal de [**p**] con respecto a [**e**], es decir

$$\overrightarrow{QP} = (1 - [\mathbf{e}][\mathbf{e}]^T)[\mathbf{p}], \tag{2.25}$$

y $\overrightarrow{QP\prime\prime}$ definida como

$$\overrightarrow{QP\prime\prime} = [\mathbf{e}] \times [\mathbf{p}] \equiv [\mathbf{E}] \times [\mathbf{p}]. \tag{2.26}$$

Por lo tanto,

$$\overrightarrow{QP} = \cos\phi(1 - [\mathbf{e}][\mathbf{e}]^T)[\mathbf{p}] + \sin\phi[\mathbf{E}][\mathbf{p}]. \tag{2.27}$$

En virtud de la igualdad anterior, [**p**$\prime$] puede ser escrito como

$$[\mathbf{p}\prime] = [\mathbf{e}][\mathbf{e}]^T[\mathbf{p}] + \cos\phi(1 - [\mathbf{e}][\mathbf{e}]^T)[\mathbf{p}] + \sin\phi[\mathbf{E}][\mathbf{p}]. \tag{2.28}$$

Agrupando términos, obtenemos

$$[\mathbf{p}\prime] = [[\mathbf{e}][\mathbf{e}]^T + \cos\phi(1 - [\mathbf{e}][\mathbf{e}]^T)[\mathbf{p}] + \sin\phi[\mathbf{E}]][\mathbf{p}]. \tag{2.29}$$

De lo anterior, es claro que $[\mathbf{p}]\prime$ es una transformación lineal de $[\mathbf{p}]$, dicha transformación está dada por la matriz dentro de los corchetes, la cual es la matriz de rotación $[\mathbf{Q}]$, esto es

$$[\mathbf{Q}] = [\mathbf{e}][\mathbf{e}]^T + \cos\phi(1 - [\mathbf{e}][\mathbf{e}]^T)[\mathbf{p}] + \sin\phi[\mathbf{E}]. \tag{2.30}$$

Forma canónica de la matriz de rotación

La matriz de rotación $[\mathbf{Q}]$ toma una forma más simple si el eje de rotación coincide con alguno de los ejes de un marco coordenado. Por ejemplo, si el eje X es paralelo al eje de rotación, es decir, paralelo al vector $[\mathbf{e}]$, se tiene:

$$[[\mathbf{e}][\mathbf{e}]^{\mathbf{T}}{}_X]_= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad [\mathbf{E}_X]_= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \tag{2.31}$$

$$[\mathbf{Q}_X]_= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}. \tag{2.32}$$

Así mismo, si hacemos coincidir el eje de rotación con los ejes Y y Z, obtenemos

$$[\mathbf{Q}_Y]_= \begin{bmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{bmatrix} \qquad [\mathbf{Q}_Z]_= \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.33}$$

2.2. Notación de Denavit-Hartenberg

Según vimos en el capítulo anterior, los eslabones de un robot pueden ser modelados como cuerpos rígidos, y como tales para analizarlos es suficiente

con asignarles una ubicación en el espacio definida por un marco coordenado, el cual contiene la información sobre la posición y la orientación del eslabón al cual está sujeto. Por supuesto, hay muchas maneras de sujetar un marco coordenado a un eslabón. Hay una manera estándar que fue propuesta por Denavit y Hartenberg en 1955 y que se describe a continuación.

Los eslabones se numeran del 0 al n y se define a la i-ésima unión como aquella que acopla al $(i-1)$-ésimo eslabón con el i-ésimo eslabón. De esta forma, se supone que el manipulador está compuesto por $n+1$ eslabones y n uniones, cada una de las cuales puede ser rotacional o prismática; no es necesario considerar otro tipo de pares cinemáticos inferiores pues, cualquier otro par cinemático inferior puede ser modelado como composición de uniones rotacionales y/o prismáticas.

Por convención, el eslabón 0 será la base fija, mientras que el eslabón n será el órgano terminal. Tras numerar los eslabones, debemos definir marcos coordenados $\mathcal{F}_i$ definidos con origen O_i y ejes coordenados X_i, Y_i y Z_i. Este marco coordenado es entonces sujetado al $(i-1)$-ésimo eslabón (el lector debe notar este desfazamiento de los índices, el cual es fácil de recordar si se tiene en mente que el primer marco coordenado corresponde a la base, la cual es el eslabón 0) para $i = 1, \ldots, n+1$.

Para los primeros n marcos coordenados (es decir, para todos menos para el correspondiente al órgano terminal), los marcos $\mathcal{F}_i$ se definen de la manera siguiente:

1. Z_i es el eje de la i-ésima unión. Si la unión es una revoluta, hay dos posibilidades para definir la dirección positiva del eje Z_i, en virtud de que el eje de rotación de una unión es simplemente una línea, no un segmento dirigido. Por otra parte, si la unión es prismática, existe un número infinito de posibilidades para definir la ubicación del eje de la unión pues, si bien la dirección queda bien definida por la dirección de desplazamiento, el eje de una unión prismática puede pasar por cualquier punto de la sección transversal y puede inclusive yacer afuera

de la unión.

2. Para la base, el eje X_1 se define de manera arbitraria según sea conveniente, con la única condición de que sea perpendicular a Z_1. Para $i = 2$, $\ldots$, n,el eje X_i se define como la perpendicular común a Z_{i-1} y Z_i en ese orden, i.e., dirigida de Z_{i-1} a Z_i; obsérvese que esta construcción garantiza la perpendicularidad de X_i y Z_i para $i = 2, \ldots, n$.

   En caso que Z_{i-1} y Z_i se intersecten, la perpendicular común a Z_{i-1} y Z_i se define con ayuda del producto cruz: si $[\mathbf{k}]_{i-1}$ y $[\mathbf{k}]_i$ son vectores unitarios sujetos a los ejes Z_{i-1} y Z_i, respectivamente, entonces $[\mathbf{k}]_{i-1} \times [\mathbf{k}]_i$ define la dirección y el sentido positivo de X_i.

   Por otra parte, si Z_{i-1} y Z_i son paralelos, entonces podemos definir todavía su perpendicular común, pero la ubicación de la misma queda indefinida. En este caso supondremos que X_i pasa por el origen del marco anterior $\mathcal{F}_{i-1}$.

   El origen O_i del marco coordenado $\mathcal{F}_i$ se define, naturalmente, como la intersección de X_i y Z_i y el eje Y_i se define por la regla de la mano derecha.

   Observemos que, en cualquier caso, X_{i+1} siempre intersecta a Z_i perpendicularmente; al punto de intersección de ambos ejes lo denotamos por $O_{i+1/2}$. Nótese que aún si Z_i y Z_{i+1} se intersectan, ello no significa que $O_{i+1/2} = O_i$.

3. La *distancia* entre Z_i y Z_{i+1} se define como a_i, la cual, como toda distancia, es no negativa.

4. La coordenada sobre el eje Z_i de la intersección $O_{i+1/2}$ de Z_i con X_{i+1} se denota por b_i. Hay que tomar en cuenta que esta cantidad es una coordenada y, por lo tanto, puede ser positiva o negativa y que aunque los ejes Z_i y Z_{i+1} se intersecten la coordenada b_i no es necesariamente

igual a cero. La distancia entre X_i y X_{i+1} es $|b_i|$ y se denomina *desplazamiento* entre perpendiculares comunes (los ejes X_i son perpendiculares comunes a Z_{i-1} y Z_i) sucesivas.

5. Nótese que, por construcción, X_{i+1} es perpendicular a Z_i y Z_{i+1}. Esto nos permite definir el ángulo α_i como el ángulo entre Z_i y Z_{i+1} medido alrededor de la dirección positiva de X_{i+1}. El ángulo α_i se conoce como *ángulo de torsión* entre pares de ejes sucesivos.

6. Z_i es perpendicular a X_i y a X_{i+1} por la definición misma de X_i y de X_{i+1}. El ángulo entre X_i y X_{i+1} se define como θ_i y se mide alrededor de la dirección positiva de Z_i.

El marco coordenado $\mathcal{F}_{n+1}$ está unido al extremo del n-ésimo eslabón. En virtud de que el manipulador no tiene $(n+1)$-ésimo eslabón. Dicho marco, para el caso de máquinas herramienta, el eje positivo Z_{n+1} se define apuntando en dirección opuesta al punto de operación, y los ejes X_{n+1} y Y_{n+1} generando un plano paralelo al plano de operación.

2.3. Relación entre Marcos Coordenados

Secciones atrás hemos hecho énfasis en la idea de considerar los eslabones de un manipulador como cuerpos rígidos y sobre la idea de sujetar marcos coordenados a ellos. Ahora bien, describiremos cómo lo anterior nos será de vital ayuda al modelar la postura del manipulador, considerando su posición como la suma de distancias entre los orígenes de los marcos coordenados y el producto de las matrices de rotación de cada marco como su orientación.

2.3.1. Matrices de Rotación

Consideraremos la distancia entre los orígenes O_i y O_{i+1} como un vector tridimensional descrito en $\mathcal{F}_i$. Ahora es evidente la utilidad de una matriz

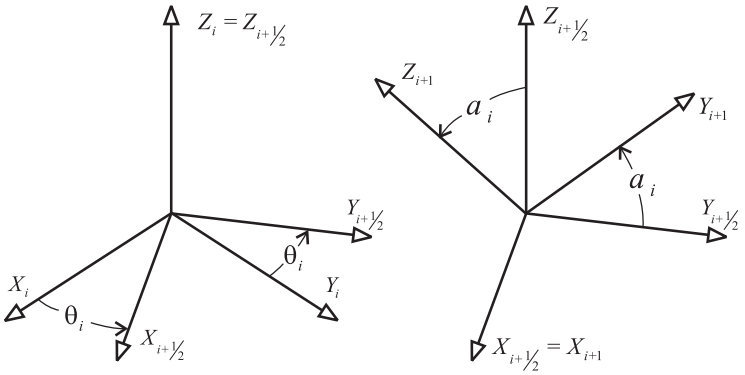


Figura 2.2: Marco de referencia intermedio.

de rotación. Como se vió en el capítulo anterior, la matriz de rotación nos ayudará a cambiar la descripción de un vector de un marco de referencia a otro.

Seguramente el lector se habrá dado cuenta que la relación existente entre los marcos coordenados sujetados a los eslabones de un manipulador según la notación de Denavit-Hartenberg, no son simples rotaciones alrededor de un eje, si no más bien, el producto de ellas. Para hacer lo anterior, nos auxiliaremos de un marco coordenado intermedio $\mathcal{F}_{i+1/2}$, el cual representa una rotación de θ_i grados alrededor del eje Z_i. La matriz $[\mathbf{\Theta}]_i$ que nos lleva del marco $\mathcal{F}_i$ al marco $\mathcal{F}_{i+1/2}$ tiene la forma

$$[\mathbf{\Theta}_i]_= \begin{bmatrix} \cos\theta_i & \sin\theta_i & 0 \\ \sin\theta_i & -\sin\theta_i & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.34}$$

Para alcanzar la orientación deseada es necesesario ahora aplicar una rotación de α_i grados alrededor del eje $X_{i+1/2}$ que nos lleva del marco $\mathcal{F}_{i+1/2}$ al marco $\mathcal{F}_{i+1}$. Sabemos que el ángulo de esta rotación es constante y que lo hemos definido con los parámetros de Denavit-Hartenberg. Si definimos a $\lambda_i = \cos\alpha_i$ y $\mu_i = \sin\alpha_i$, la matriz $[\mathbf{A}]_{i+1/2}$ que hace el cambio en la descripción de $\mathcal{F}_{i+1/2}$ a $\mathcal{F}_{i+1}$ queda definida como sigue

$$[\mathbf{A}_{i+1/2}]_{=}\begin{bmatrix}1 & 0 & 0\\ 0 & \lambda_i & -\mu_i\\ 0 & \mu_i & \lambda_i\end{bmatrix}. \tag{2.35}$$

Por lo tanto, es claro que la matriz de rotación $[\mathbf{Q}_i]_i$ que describe la rotación de $\mathcal{F}_i$ a $\mathcal{F}_{i+1}$ definida en el marco $\mathcal{F}_i$ está dada como

$$[\mathbf{Q}_i]_i = [\mathbf{\Theta}]_i[\mathbf{A}]_{i+1/2}, \tag{2.36}$$

es decir,

$$[\mathbf{Q}_i]_i = \begin{bmatrix}\cos\theta_i & -\lambda_i\sin\theta_i & \mu_i\sin\theta_i\\ \sin\theta_i & \lambda_i\cos\theta_i & -\mu_i\cos\theta_i\\ 0 & \mu_i & \lambda_i\end{bmatrix}. \tag{2.37}$$

2.3.2. Vectores de posición

Denotaremos a $\mathbf{a}_i$ como el vector de posición entre el origen O_i del marco $\mathcal{F}_i$ y el origen O_{i+1} del marco $\mathcal{F}_{i+1}$ teniendo en cuenta el marco intermedio $\mathcal{F}_{i+1/2}$ mencionado anteriormente.

El vector $\mathbf{a}_i$ puede ser definido como la suma de los vectores que representan las distancias a_i y b_i, es decir,

$$[\mathbf{a}_i]_i = [\overrightarrow{O_iO_{i+1}}_i]_{=}[\overrightarrow{O_iO_{i+1/2}}_i]_{+}[\overrightarrow{O_{i+1/2}O_{i+1}}_i]. \tag{2.38}$$

Dado que b_i es la coordenada sobre el eje $Z_i = Z_{i+1/2}$ del origen $O_{i+1/2}$, el primer vector de la ecuación 2.38 referenciado al marco $\mathcal{F}_i$ se puede escribir como

$$[\overrightarrow{O_iO_{i+1/2}}_i]_{=}\begin{bmatrix}0\\ 0\\ b_i\end{bmatrix}. \tag{2.39}$$

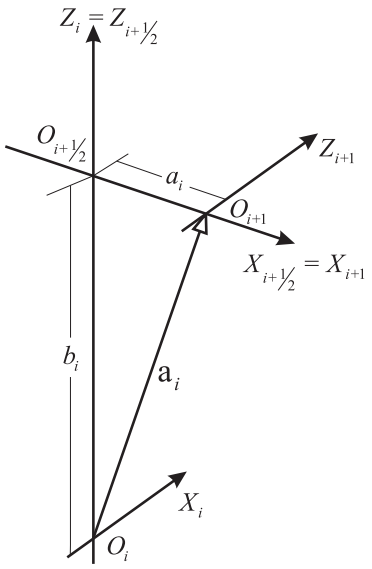


Figura 2.3: Vector de posicion entre O_i y O_{i+1} descrito en el marco $\mathcal{F}_i$.

Analogamente, para el vector de la coordenada a_i del origen O_{i+1} sobre el eje $X_{i+1/2} = X_{i+1}$ referenciado al marco $\mathcal{F}_{i+1}$ se tiene

$$[\overrightarrow{O_{i+1/2}O_{i+1}}_{i+1}] = \begin{bmatrix} a_i \\ 0 \\ 0 \end{bmatrix}. \tag{2.40}$$

Antes de operar estos vectores, debemos representarlos en el mismo marco de referencia. Para hacer esto haremos uso de la mtriz de rotación $[\mathbf{Q}]$ definida en la ecuación 2.37, es decir

$$[\overrightarrow{O_{i+1/2}O_{i+1}}_{i}] = [\mathbf{Q}_i]_i [\overrightarrow{O_{i+1/2}O_{i+1}}_{i+1}], \tag{2.41}$$

esto es,

$$[\overrightarrow{O_{i+1/2}O_{i+1}}]_i = \begin{bmatrix} \cos\theta_i & -\lambda_i \sin\theta_i & \mu_i \sin\theta_i \\ \sin\theta_i & \lambda_i \cos\theta_i & -\mu_i \cos\theta_i \\ 0 & \mu_i & \lambda_i \end{bmatrix} \begin{bmatrix} a_i \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} a_i \cos\theta_i \\ a_i \sin\theta_i \\ 0 \end{bmatrix}. \quad (2.42)$$

Ahora podemos realizar la suma vectorial como se muestra a continuación

$$[\mathbf{a}_i]_i = [\overrightarrow{O_iO_{i+1/2}}]_i + [\overrightarrow{O_{i+1/2}O_{i+1}}]_i = \begin{bmatrix} a_i \cos\theta_i \\ a_i \sin\theta_i \\ b_i \end{bmatrix}. \quad (2.43)$$

En virtud de que $[\mathbf{Q}_i]_i^{-1} = [\mathbf{Q}_i]_i^T$ podemos tener la representación del vector $\mathbf{a}_i$ en términos del marco coordenado $\mathcal{F}_{i+1}$

$$[\mathbf{a}_i]_{i+1}] = [\overrightarrow{O_iO_{i+1/2}}]_{i+1} + [\overrightarrow{O_{i+1/2}O_{i+1}}]_{i+1} \quad (2.44)$$

$$= \begin{bmatrix} \cos\theta_i & \sin\theta_i & 0 \\ -\lambda_i \sin\theta_i & \lambda_i \cos\theta_i & \mu_i \\ \mu_i \sin\theta_i & -\mu_i \cos\theta_i & \lambda_i \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ b_i \end{bmatrix} + \begin{bmatrix} a_i \\ 0 \\ 0 \end{bmatrix}$$

$$[\mathbf{a}_i]_{i+1} = \begin{bmatrix} a_i \\ b_i\mu_i \\ b_i\lambda_i \end{bmatrix}. \quad (2.45)$$

2.4. Compiladores

Un compilador es un programa que lee código escrito en un lenguaje, el código *fuente,* y lo traduce un código equivalente en otro lenguaje, el código *objeto.* Durante la compilación el compilador informa al programador acerca de errores encontrados en el programa fuente.

La compilación de un programa consiste principalmente dos fases: análisis y síntesis. Durante el análisis se divide al programa fuente en sus elementos

componentes y se crea una representación intermedia del programa fuente. En la fase de síntesis se construye el programa objeto deseado a partir de la representación intermedia.

2.4.1. Análisis y síntesis

El análisis del lenguaje fuente realizado por un compilador se puede divide en tres etapas, a saber:

Análisis léxico Esta etapa, también conocida como *exploración,* desglosa los elementos que constituyen el programa en código fuente de izquierda a derecha y agrupando al mismo en *componentes léxicos* o *tokens.* cada token es una cadena de caracteres que tiene un significado colectivo.

Análisis sintáctico Esta etapa, también conocida como *análisis jerárquico,* agrupa jerárquicamente los componentes léxicos en colecciones anidadas llamadas *árboles.* La posición de cada token dentro del árbol sintáctico determina las operaciones que son válidas para el mismo.

Análisis semántico Consiste en realizar ciertas validaciones para asegurar que los componentes de un programa se ajustan de un modo coherente.

La parte del compilador que realiza el análisis se denomina *extremo frontal.* La síntesis del lenguaje objeto se divide a su vez en tres etapas, a saber:

Generación de código intermedio Después de los análisis sintáctico y semántico, la mayoría de los compiladores generan una representación intermedia explícita del programa fuente, la cual generalmente es una representación apropiada para una máquina abstracta de pila. Dicha representación intermedia debe cumplir los requisitos de ser fácil de generar y fácil de traducir al programa objeto.

Optimización de código En esta fase el código intermedio es optimizado de tal manera que se obtenga un código máquina más rápido de ejecutar. Algunas optimizaciones son triviales. En los compiladores optimizadores buena parte del tiempo a veces se gasta en esta etapa.

Generación de código Consiste en generar código objeto, generalmente código de máquina localizable o código ensamblador.

La parte del compilador que realiza la síntesis se denomina *extremo posterior.*

Si la optimización de código no es innecesaria se pueden omitir las etapas de *generación de código intermedio* y de *optimización de código* y generar de una sola vez el código objeto por medio de la técnica conocida como *traducción dirigida por sintaxis.*

Definicion de la sintaxis

La sintaxis de los lenguajes se describe por medio de gramáticas; de particular interés son la sintaxis independientes del contexto, las cuales son suficientemente completas como para describir un amplio número de lenguajes de programación.

Una gramática describe de forma natural la estructura de muchas construcciones de los lenguajes de programación. Consideremos, por ejemplo, la proposición **if-else** de C, la cual tiene la forma:

si expresión proposición$_1$ **sino** proposición$_2$

esta proposición es la concatenación de la palabra clave **if** una expresión, la proposición$_1$, la palabra clave **else** y la proposición$_2$. Si empleamos la variable *expr* para denotar una expresión, y la variable *prop*, para una proposición, esta regla de estructuración se expresa como:

$$prop \rightarrow \textbf{if}\ expr\ prop\ \textbf{sino}\ prop \tag{2.46}$$

donde la flecha se define como "puede tener la forma;" la regla (2.46) se denomina *producción.* Los elementos que conforman una producción reciben el nombre de símbolos, y se subclasifican como *terminales* y *no terminales.* Los símbolos terminales son los símbolos tales como **if** y **else**, y se denominan de esa forma debido a que no se pueden descomponer en tokens más simples. Las variables *expr* y *prop* son no terminales debido a que pueden descomponerse aún más en elementos más simples; nótese que la producción (2.46) es recursiva.

Una *gramática independiente del contexto* tiene cuatro componentes:

1. Un conjunto de tokens, denominados símbolos terminales.

2. Un conjunto de símbolos no terminales.

3. Un conjunto de producciones, formadas por un no terminal, llamado el *lado izquierdo* de la producción, una flecha y una sucesión de tokens y/o no terminales, denominada *lado derecho* de la producción.

4. La selección de uno de los símbolos no terminales como *símbolo inicial.*

Por convención se especifica una gramática dando una lista de sus producciones, donde la producción del símbolo inicial se lista primero. Asimismo, los símbolos terminales de las producciones se muestran en **negrita** en caso de ser cadenas de caracteres y los símbolos no terminales en *cursiva;* cualquier símbolo que no se muestre en negrita o cursiva será considerado un símbolo terminal.

Por ejemplo, consideremos el lenguaje formado por dígitos, los signos + y − y las operaciones binarias usuales entre ellos de suma +, resta −, mul-

tiplicación $\times$ y división $\div$. Las producciones de esta gramática son:

$$\begin{aligned}
\textit{expr} &\rightarrow \textit{expr} + \textit{término} \\
\textit{expr} &\rightarrow \textit{expr} - \textit{término} \\
\textit{expr} &\rightarrow \textit{primer-término} \\
\textit{primer-término} &\rightarrow + \textit{término} \\
\textit{primer-término} &\rightarrow - \textit{término} \\
\textit{primer-término} &\rightarrow \textit{término} \\
\textit{término} &\rightarrow \textit{término} \times \textit{factor} \\
\textit{término} &\rightarrow \textit{término} \div \textit{factor} \\
\textit{término} &\rightarrow \textit{factor} \\
\textit{factor} &\rightarrow (\textit{expr}) \\
\textit{factor} &\rightarrow \textit{dígito} \\
\textit{dígito} &\rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{aligned} \tag{2.47}$$

Es claro que el símbolo inicial de esta gramática es *expr* (expresión). Sin embargo, hay tres producciones correspondientes a *expr*, es decir, *expr* tiene tres *derivaciones;* la derivación correcta depende de cuál sea la entrada del compilador. La técnica estándar para discernir cuál es la derivación correcta consiste en leer un símbolo de *preanálisis* (*lookahead* ó *LA* en inglés) de la entrada y emplearlo como criterio para elegir una de las producciones; esto es lo que se conoce como *análisis sintáctico predictivo.* Se dice que un compilador es *predictivo* si su análisis sintáctico es predictivo.

Un rápido análisis de la gramática (2.47) nos permite ver que no es posible realizar un análisis sintáctico predictivo a partir de la misma pues las dos primeras producciones

$$\begin{aligned}
\textit{expr} &\rightarrow \textit{expr} + \textit{término} \\
\textit{expr} &\rightarrow \textit{expr} - \textit{término}
\end{aligned}$$

son *recursivas por la izquierda,* i.e., no comienzan con un terminal y el primer no terminal es el mismo que el lado izquierdo de la producción. Un análisis recursivo que comience con la producción *expr* puede entrar en un ciclo recursivo infinito de llamadas al procedimiento para analizar la producción *expr* pues el símbolo de preanálisis no cambia al hacer una llamada a procedimiento. Para eliminar este inconveniente es usual *factorizar por la izquierda,* es decir, reemplazar las tres primeras producciones de (2.47) con

$$\begin{aligned} \textit{expr} &\rightarrow \textit{primer-término más-términos} \\ \textit{más-términos} &\rightarrow \epsilon \\ \textit{más-términos} &\rightarrow +\textit{término más-términos} \\ \textit{más-términos} &\rightarrow -\textit{término más-términos} \end{aligned}$$

introduciendo el nuevo no terminal *más-términos* y el símbolo ϵ para denotar la *palabra vacía* (ausencia de símbolos); aunque las últimas dos producciones son *recursivas por la derecha*, ese no es un inconveniente ya que los terminales (tokens) se leen por la izquierda. De este modo siempre es posible determinar cuál derivación emplear inspeccionando únicamente el símbolo de preanálisis.

La nueva gramática *factorizada por la izquierda* es

$$
\begin{aligned}
\textit{expr} &\rightarrow \textit{primer-término más-términos} \\
\textit{más-términos} &\rightarrow \epsilon \\
\textit{más-términos} &\rightarrow + \textit{término más-términos} \\
\textit{más-términos} &\rightarrow - \textit{término más-términos} \\
\textit{primer-término} &\rightarrow +\textit{término} \\
\textit{primer-término} &\rightarrow -\textit{término} \\
\textit{primer-término} &\rightarrow \textit{término} \\
\textit{término} &\rightarrow \textit{factor más-factores} \\
\textit{más-factores} &\rightarrow \epsilon \\
\textit{más-factores} &\rightarrow \times \textit{factor más-factores} \\
\textit{más-factores} &\rightarrow \div \textit{factor más-factores} \\
\textit{factor} &\rightarrow (\textit{expr}) \\
\textit{factor} &\rightarrow \textit{dígito} \\
\textit{dígito} &\rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{aligned}
\tag{2.48}
$$

Un ejemplo es conveniente para comprender mejor esto. Si la entrada es igual a "$-3 + 4 \times 5$" al leer de izquierda a derecha podemos ir analizando la sintaxis de la entrada en la forma en que se muestra en la tabla 2.1. Después de leer la producción (cualquiera de las dos) correspondiente a *expr* debemos emplear (llamar el procedimiento asignado a) la producción *primer-término* sin importar cuál sea el símbolo de preanálisis. Los renglones de la tabla 2.1 muestran el proceso.

Entrada por leer	Preanálisis	Producción
$3 + 4 \times 5$	$-$	*expr* $\rightarrow$ *primer-término más-términos*
$3 + 4 \times 5$	$-$	*primer-término* $\rightarrow -$*término*
$+4 \times 5$	3	*término* $\rightarrow$ *factor más-factores*
$+4 \times 5$	3	*factor* $\rightarrow$ *dígito*
$+4 \times 5$	3	*más-factores* $\rightarrow \epsilon$
4×5	$+$	*más-términos* $\rightarrow$ $+$ *término más-términos*
$\times 5$	4	*término* $\rightarrow$ *factor más-factores*
5	$\times$	*más-factores* $\rightarrow$ $\times$ *factor más-factores*
ϵ	5	*factor* $\rightarrow$ *dígito*
ϵ	ϵ	*más-factores* $\rightarrow \epsilon$

Tabla 2.1: Análisis sintáctico de la expresión "$-3 + 4 \times 5$."

Capítulo 3

Metodología

Para facilitar la administración de posturas y movimientos del robot es necesario encontrar el modelo matemático que describa la relación entre un conjunto de ángulos de unión y las coordenadas del órgano terminal y viceversa, esto es, encontrar la solución al problema de cinemática directa e inversa.

Antes de comenzar el análisis es importante definir algunos de los parámetros del robot, el FANUC S 420-F en concreto, ilustrado en la figura 3.1. Deacuerdo con la notación de Denavit-Hartenberg se obtienen los siguientes parámetros.

Unión	$a_i(mm)$	$b_i(mm)$	α_i	λ_i	μ_i
1	270	1000	90°	0	1
2	900	0	0°	1	0
3	270	0	90°	0	1
4	0	1300	90°	0	1
5	0	0	90°	0	1

Tabla 3.1: Tabla de parámetros de Denavit-Hartenberg del robot FANUC S 420-F.

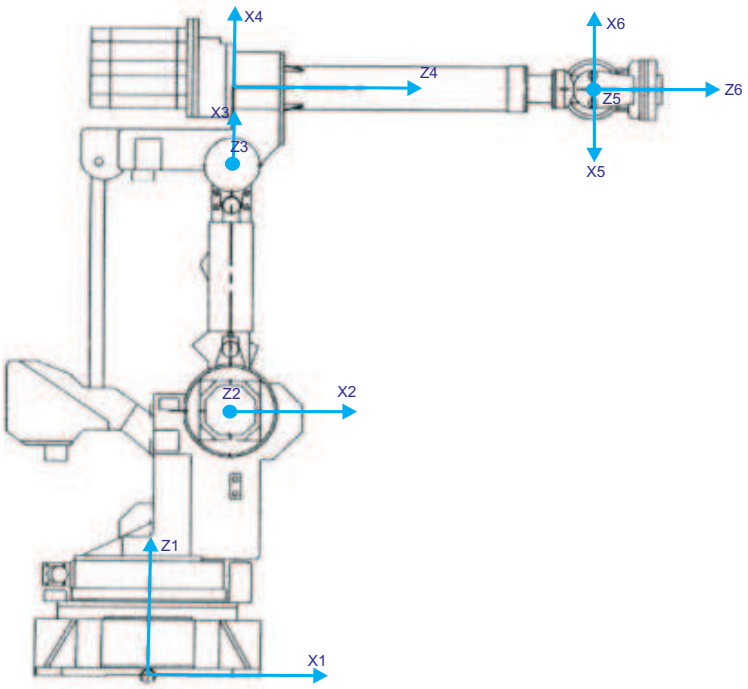


Figura 3.1: Relación de marcos coordenados para el robot FANUC S 420-F acorde a la notación de Denavit-Hartenberg.

3.1. Cinemática directa

El problema de cinemática directa como se ha dicho antes, consiste en encontrar las coordenadas en el espacio del órgano terminal y su orientación con respecto a un marco coordenado fijo dados los ángulos para cada una de sus uniones. La solución a este problema es simple desde un punto de vista geométrico.

3.1.1. Cinemática directa de posición

La posición del órgano terminal del robot en el espacio con respecto al marco base es la suma vectorial de los vectores de posición asociados a cada origen. Dado que estos vectores están descritos en distintos marcos de referencia, nos ayudaremos de las matrices de rotación de cada unión para poder realizar la suma. Es decir,

$$[\mathbf{p}]_1 = [\mathbf{a}_1]_1 + [\mathbf{Q}_1]_1[\mathbf{a}_2]_2 + [\mathbf{Q}_1]_1[\mathbf{Q}_2]_2[\mathbf{a}_3]_3 + [\mathbf{Q}_1]_1[\mathbf{Q}_2]_2[\mathbf{Q}_3]_3[\mathbf{a}_4]_4 + [\mathbf{Q}_1]_1[\mathbf{Q}_2]_2[\mathbf{Q}_3]_3[\mathbf{Q}_4]_4[\mathbf{a}_5]_5 + [\mathbf{Q}_1]_1[\mathbf{Q}_2]_2[\mathbf{Q}_3]_3[\mathbf{Q}_4]_4[\mathbf{Q}_5]_5[\mathbf{a}_6]_6. \tag{3.1}$$

3.1.2. Cinemática directa de orientación

Obtener la orientación a partir de los ángulos de las uniones del robot es aún más sencillo que la posición, ya que se trata del producto de todas las rotaciones hechas a partir de la base hasta la última unión. Esto es,

$$[\mathbf{Q}]_1 = [\mathbf{Q}_1]_1[\mathbf{Q}_2]_2[\mathbf{Q}_3]_3[\mathbf{Q}_4]_4[\mathbf{Q}_5]_5[\mathbf{Q}_6]_6. \tag{3.2}$$

3.2. Cinemática inversa

La solución al problema de cinemática inversa, a diferencia de su contraparte, es mucho más complejo. Este problema consiste en determinar todos

los conjuntos de valores para los ángulos de las uniones tales que, lleven al robot a adoptar una posición y orientación dada. La dificultad adyacente a este problema es que es de caracter no lineal, si es posible o no encontrar una solución y sí es única.

3.2.1. Cinemática inversa de posición

La arquitectura del robot FANUC S 420-F es tal que nos permite tratar el problema de cinemática inversa de posición y orientación de forma independiente. Esto se debe a que se trata de un mecanismo de tipo 6R. Donde los ejes de rotación de las últimas tres articulaciones coinciden en un mismo punto.

Para comenzar el análisis haremos uso de la ecuación 3.1 para obtener las coordenadas del punto [**c**], sobre el cual coinciden los ejes de rotación de las tres últimas uniones. Es importante notar, que dicho punto del robot no depende de los valores de los ángulos de las tres últimas uniones.

$$[\mathbf{c}]_1 = [\mathbf{p}]_1 - [\mathbf{Q}_1]_1[\mathbf{Q}_2]_2[\mathbf{Q}_3]_3[\mathbf{Q}_4]_4[\mathbf{a}_5]_5 - [\mathbf{Q}_1]_1[\mathbf{Q}_2]_2[\mathbf{Q}_3]_3[\mathbf{Q}_4]_4[\mathbf{Q}_5]_5[\mathbf{a}_6]_6, \tag{3.3}$$

siendo el vector $[\mathbf{a}_5]_5 = 0$, lo anterior queda como

$$[\mathbf{c}]_1 = [\mathbf{p}]_1 - [\mathbf{Q}_1]_1[\mathbf{Q}_2]_2[\mathbf{Q}_3]_3[\mathbf{Q}_4]_4[\mathbf{Q}_5]_5[\mathbf{a}_6]_6. \tag{3.4}$$

Si se premultiplica el último término de la parte derecha de la ecuación anterior por $[\mathbf{Q}_6]_6[\mathbf{Q}_6]_6^{-1}$ se obtiene

$$[\mathbf{c}]_1 = [\mathbf{p}]_1 - [\mathbf{Q}_1]_1[\mathbf{Q}_2]_2[\mathbf{Q}_3]_3[\mathbf{Q}_4]_4[\mathbf{Q}_5]_5[\mathbf{Q}_6]_6[\mathbf{Q}_6]_6^{-1}[\mathbf{a}_6]_6. \tag{3.5}$$

De la ecuación 3.2 y los primeros factores que preceden al segundo término del lado derecho de la ecuación anterior, es claro que se trata de la orientación deseada, por lo tanto se puede escribir

$$[\mathbf{c}]_1 = [\mathbf{p}]_1 - [\mathbf{Q}]_1[\mathbf{Q}_6]_6^{-1}[\mathbf{a}_6]_6. \tag{3.6}$$

Otro hecho de particular interés es que $[\mathbf{Q}_6]_6^{-1}[\mathbf{a}_6]_6 = [\mathbf{a}_6]_7$ y aún más importante, es que las entradas de este vector no dependen del valor de la sexta variable de unión. Por lo tanto, es posible encontrar las coordenadas del punto $[\mathbf{c}]$ sin conocer los valores de las últimas tres variables de unión como sigue

$$[\mathbf{c}]_1 = [\mathbf{p}]_1 - [\mathbf{Q}]_1[\mathbf{a}_6]_7. \tag{3.7}$$

Es necesario dejar en claro que aunque las entradas del vector $[\mathbf{a}_6]_7$ no dependen del valor de la última variable de unión; si dependen de la herramienta que lleve el robot. Sin una herramienta en particular para el control del robot FANUC S 420-F, el vector queda descrito como

$$[\mathbf{a}_6]_7 = \begin{bmatrix} a_6 \\ b_6\mu_6 \\ b_6\lambda_6 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -260 \end{bmatrix} \tag{3.8}$$

Deacuerdo con la ecuación 3.1, el vector $[\mathbf{c}]$ puede escribirse en términos de los vectores de posición como sigue

$$[\mathbf{c}]_1 = [\mathbf{a}_1]_1 + [\mathbf{Q}_1]_1[\mathbf{a}_2]_2 + [\mathbf{Q}_1]_1[\mathbf{Q}_2]_2[\mathbf{a}_3]_3 + [\mathbf{Q}_1]_1[\mathbf{Q}_2]_2[\mathbf{Q}_3]_3[\mathbf{a}_4]_4 \tag{3.9}$$

Esto es claro no solo desde un punto de vista algebraico, sino también de forma geométrica, teniendo en cuenta que el vector $[\mathbf{c}]$ es la suma de los primeros cuatro vectores de posición asociados a los orígenes de los marcos coordenados del robot (ver figura 3.1).

Antes de proceder a realizar las operaciones correspondientes para encontrar las entradas del vector $[\mathbf{c}]$, es conveniente realizar algunas simplificaciones como se describe a continuación.

$$[\mathbf{c}]_1 - [\mathbf{a}_1]_1 = [\mathbf{Q}_1]_1[\mathbf{a}_2]_2 + [\mathbf{Q}_1]_1[\mathbf{Q}_2]_2[\mathbf{a}_3]_3 + [\mathbf{Q}_1]_1[\mathbf{Q}_2]_2[\mathbf{Q}_3]_3[\mathbf{a}_4]_4 \quad (3.10)$$

Multiplicando ambos lados de la igualdad por $[\mathbf{Q}_1]_1^{-1}$ se tiene

$$[\mathbf{Q}_1]_1^{-1}([\mathbf{c}]_1 - [\mathbf{a}_1]_1) = [\mathbf{a}_2]_2 + [\mathbf{Q}_2]_2[\mathbf{a}_3]_3 + [\mathbf{Q}_2]_2[\mathbf{Q}_3]_3[\mathbf{a}_4]_4. \quad (3.11)$$

Si se factoriza a $[\mathbf{Q}_2]_2$ del lado derecho de la igualdad obtenemos

$$[\mathbf{Q}_1]_1^{-1}([\mathbf{c}]_1 - [\mathbf{a}_1]_1) = [\mathbf{Q}_2]_2([\mathbf{Q}_2]_2^{-1}[\mathbf{a}_2]_2 + [\mathbf{a}_3]_3 + [\mathbf{Q}_3]_3[\mathbf{a}_4]_4), \quad (3.12)$$

o de forma equivalente

$$[\mathbf{Q}_1]_1^{-1}[\mathbf{c}]_1 - [\mathbf{a}_1]_2 = [\mathbf{Q}_2]_2([\mathbf{a}_2]_3 + [\mathbf{a}_3]_3 + [\mathbf{Q}_3]_3[\mathbf{a}_4]_4). \quad (3.13)$$

Con ayuda de la tabla 3.1 definimos todos lo factores de la ecuación anterior como

$$[\mathbf{Q}_1]_1 = \begin{bmatrix} \cos\theta_1 & 0 & \sin\theta_1 \\ \sin\theta_1 & 0 & -\cos\theta_1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$[\mathbf{Q}_2]_2 = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$[\mathbf{Q}_3]_3 = \begin{bmatrix} \cos\theta_3 & 0 & \sin\theta_3 \\ \sin\theta_3 & 0 & -\cos\theta_3 \\ 0 & 1 & 0 \end{bmatrix}$$

$$[\mathbf{a}_1]_2 = \begin{bmatrix} 270 \\ 1000 \\ 0 \end{bmatrix}$$

$$[\mathbf{a}_2]_3 = \begin{bmatrix} 900 \\ 0 \\ 0 \end{bmatrix}$$

$$[\mathbf{a}_3]_3 = \begin{bmatrix} 270\cos\theta_3 \\ 270\sin\theta_3 \\ 0 \end{bmatrix}$$

$$[\mathbf{a}_4]_4 = \begin{bmatrix} 0 \\ 0 \\ 1300 \end{bmatrix}$$

Sustituyendo lo anterior se obtiene la siguiente igualdad

$$\begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \left(\begin{bmatrix} 900 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 270\cos\theta_3 \\ 270\sin\theta_3 \\ 0 \end{bmatrix} + \begin{bmatrix} \cos\theta_3 & 0 & \sin\theta_3 \\ \sin\theta_3 & 0 & -\cos\theta_3 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1300 \end{bmatrix} \right) =$$

$$\begin{bmatrix} \cos\theta_1 & 0 & \sin\theta_1 \\ \sin\theta_1 & 0 & -\cos\theta_1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} - \begin{bmatrix} 270 \\ 1000 \\ 0 \end{bmatrix} \tag{3.14}$$

$$\begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \left(\begin{bmatrix} 900 + 270\cos\theta_3 \\ 270\sin\theta_3 \\ 0 \end{bmatrix} + \begin{bmatrix} 1300\sin\theta_3 \\ -1300\sin\theta_3 \\ 0 \end{bmatrix} \right) =$$

$$\begin{bmatrix} x_c\cos\theta_1 + y_c\sin\theta_1 \\ z_c \\ x_c\sin\theta_1 + y_c\cos\theta_1 \end{bmatrix} - \begin{bmatrix} 270 \\ 1000 \\ 0 \end{bmatrix} \tag{3.15}$$

$$\begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 900 + 270\cos\theta_3 + 1300\sin\theta_3 \\ 270\sin\theta_3 - 1300\sin\theta_3 \\ 0 \end{bmatrix} =$$

$$\begin{bmatrix} x_c \cos\theta_1 + y_c \sin\theta_1 + 270 \\ z_c - 1000 \\ x_c \sin\theta_1 + y_c \cos\theta_1 \end{bmatrix} \tag{3.16}$$

$$\begin{bmatrix} \cos\theta_2(900 + 270\cos\theta_3 + 1300\sin\theta_3) - \sin\theta_2(270\sin\theta_3 - 1300\sin\theta_3) \\ \sin\theta_2(900 + 270\cos\theta_3 + 1300\sin\theta_3) + \cos\theta_2(270\sin\theta_3 - 1300\sin\theta_3) \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} x_c \cos\theta_1 + y_c \sin\theta_1 + 270 \\ z_c - 1000 \\ x_c \sin\theta_1 + y_c \cos\theta_1 \end{bmatrix} \tag{3.17}$$

Observando la tercera ecuación del sistema 3.17 es claro que se puede obtener a θ_1 como sigue

$$\theta_1 = arctan2(yc, xc), \tag{3.18}$$

y además se trata de una solución única para ese ángulo. Nuestro siguiente paso es obtener del sistema 3.17 una relación tal que no depende de la variabe de unión θ_2. Esto se puede lograr si encontramos la norma euclidiana cuadrada en ambos lados de la ecuación 3.17. Con lo antarior, obtenemos

$$\begin{aligned} 486000\cos\theta_3 + 2340000\sin\theta_3 = x_c^2 + y_c^2 + z_c^2 - \\ 540(x_c\cos\theta_1 + y_c\sin\theta_1) - 2000z_c - 15000. \end{aligned} \tag{3.19}$$

Sabiendo que toda combinación lineal de sinusoidales resulta en otra sinusoidal con magnitud y fase diferentes podemos escribir

$$\begin{aligned} \pm\sqrt{2340000^2 + 486000^2}\sin(\theta_3 + \arctan(\frac{486000}{2340000})) = x_c^2 + y_c^2 + \\ z_c^2 - 540(x_c\cos\theta_1 + y_c\sin\theta_1) - 2000z_c - 15000, \end{aligned} \tag{3.20}$$

o equivalentemente,

$$\pm 2{,}3899 \times 10^6 \sin(\theta_3 + 0{,}2048) = x_c^2 + y_c^2 + z_c^2 - 540(x_c \cos\theta_1 + y_c \sin\theta_1) - 2000 z_c - 15000. \tag{3.21}$$

Al despejar a θ_3 se tiene

$$\theta_3 = \arcsin\left(\frac{x_c^2 + y_c^2 + z_c^2 - 540(x_c \cos\theta_1 + y_c \sin\theta_1) - 2000 z_c - 15000}{\pm 2{,}3899 \times 10^6}\right) - 0{,}2048. \tag{3.22}$$

Un hecho notable en la ecuación anterior es que se tienen dos valores para la variable de unión θ_3.

Para encontrar el valor de θ_2 usaremos las dos primeras ecuaciones del sistema 3.17 formando una matriz de dimensión 2×2.

$$\begin{bmatrix} 900 + 270\cos\theta_3 + 1300\sin\theta_3 & -270\sin\theta_3 + 1300\sin\theta_3 \\ 270\sin\theta_3 - 1300\sin\theta_3 & 900 + 270\cos\theta_3 + 1300\sin\theta_3 \end{bmatrix} \begin{bmatrix} \cos\theta_2 \\ \sin\theta_2 \end{bmatrix} = \begin{bmatrix} x_c \cos\theta_1 + y_c \sin\theta_1 + 270 \\ z_c - 1000 \end{bmatrix}. \tag{3.23}$$

Multiplicando por la izquierda ambos lados del sistema anterior por la inversa de la matriz de la derecha obtenemos las siguientes ecuaciones

$$\cos\theta_2 = \frac{(900 + 270\cos\theta_3 + 1300\sin\theta_3)(x_c \cos\theta_1 + y_c \sin\theta_1 + 270)}{486000\cos\theta_3 + 2340000\sin\theta_3 + 2572900} + \frac{(270\sin\theta_3 - 1300\sin\theta_3)(z_c - 1000)}{486000\cos\theta_3 + 2340000\sin\theta_3 + 2572900}, \tag{3.24}$$

y

$$\sin\theta_2 = \frac{(-270\sin\theta_3 + 1300\sin\theta_3)(x_c \cos\theta_1 + y_c \sin\theta_1 + 270)}{486000\cos\theta_3 + 2340000\sin\theta_3 + 2572900} +$$

$$\frac{(900 + 270\cos\theta_3 + 1300\sin\theta_3)(z_c - 1000)}{486000\cos\theta_3 + 2340000\sin\theta_3 + 2572900}. \tag{3.25}$$

De las ecuaciones 3.24 y 3.25 se obtiene una única solución para θ_2 dada como

$$\theta_2 = \arctan_2(\sin\theta_2, \cos\theta_2). \tag{3.26}$$

Con lo anterior, dado la posición y orientación del robot, es posible encontrar los tres primeros ángulos de las variables de unión y ubicar el robot en un punto tal, que con la orientación adecuada, alcanzará la posición deseada.

3.2.2. Cinemática inversa de orientación

Antes de comenzar el análisis para dar solución al problema asociado a la cinemática inversa de orientación se deben tener en cuenta algunas cuestiones. Primero, la orientación deseada nos dice que orientación ha de tener la herramienta o en nuestro caso, el último eslabón del robot, entoncés, si se conocen las tres primeras variables de unión y la orientación deseada, solo resta encontrar las tres últimas variables de unión y además, podemos encontrar una relación directa entre ellas como sigue

$$[\mathbf{R}]_4 = [\mathbf{Q}_4]_4[\mathbf{Q}_5]_5[\mathbf{Q}_6]_6 = [\mathbf{Q}_3]_3^T[\mathbf{Q}_2]_2^T[\mathbf{Q}_1]_1^T[\mathbf{Q}]_1. \tag{3.27}$$

El segundo punto a considerar es el hecho de que los vectores Z_5 y Z_6 descritos en el marco $\mathcal{F}_4$ por la construcción propia del robot, siempre forman un ángulo recto (ver figura 3.2).

Nuestro objetivo es describir el vector e_6 asociado al eje de rotación Z_6 en términos del marco $\mathcal{F}_7$ para después cambiar su base al marco $\mathcal{F}_4$. Esto lo podemos conseguir con ayuda de la matriz $[\mathbf{R}]$ descrita en la ecuación 3.27.

El vector e_6 en el marco $\mathcal{F}_7$ tiene la forma

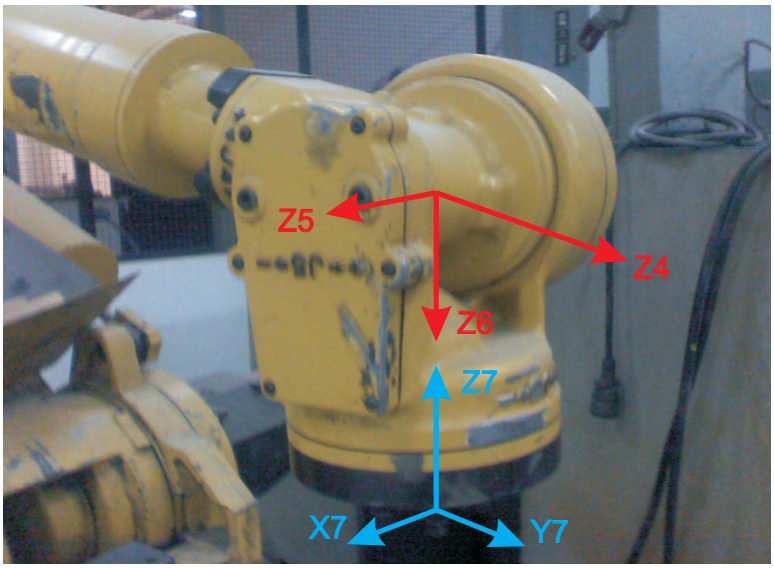


Figura 3.2: Ubicación y dirección de los tres últimos eslabones del robot.

$$[\mathbf{e}_6]_7 = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}. \tag{3.28}$$

De lo anterior, el mismo vector e_6 en el marco $\mathcal{F}_4$ está dado como

$$[\mathbf{e}_6]_4 = [\mathbf{R}]_4[\mathbf{e}_6]_7 = \begin{bmatrix} \xi \\ \eta \\ \zeta, \end{bmatrix} \tag{3.29}$$

donde

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, \tag{3.30}$$

y el vector e_5 como

$$[\mathbf{e}_5]_4 = \begin{bmatrix} \cos\theta_4 & 0 & \sin\theta_4 \\ \sin\theta_4 & 0 & -\cos\theta_4 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \sin\theta_4 \\ -\cos\theta_4 \\ 0 \end{bmatrix}. \tag{3.31}$$

Sabiendo que el producto escalar entre dos vectores es igual al ángulo que forman entre ellos, y sabiendo que el ángulo que forman los vectores $[\mathbf{e}_6]_4$ y $[\mathbf{e}_5]_4$ es siempre de $\frac{\pi}{2}$ radianes, entoncés

$$\xi \sin\theta_4 - \eta \cos\theta_4 = 0, \tag{3.32}$$

se tiene

$$\theta_4 = \arctan_2(\eta, \xi). \tag{3.33}$$

Para calcular el valor de θ_5 hacemos uso nuevamente de la ecuación 3.27 como sigue

$$[\mathbf{Q}_5]_5 = [\mathbf{Q}_4]_4^T [\mathbf{R}]_4 [\mathbf{Q}_6]_6^T \tag{3.34}$$

es decir,

$$[\mathbf{Q}_5]_5 = \begin{bmatrix} \cos\theta_4 & \sin\theta_4 & 0 \\ 0 & 0 & 1 \\ \sin\theta_4 & -\cos\theta_4 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} \cos\theta_6 & \sin\theta_6 & 0 \\ \sin\theta_6 & -\cos\theta_6 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \tag{3.35}$$

o equivalentemente,

$$\begin{bmatrix} \cos\theta_5 & 0 & \sin\theta_5 \\ \sin\theta_5 & 0 & -\cos\theta_5 \\ 0 & 1 & 0 \end{bmatrix} =$$

$$\begin{bmatrix} r_{11}\cos\theta_4 + r_{21}\sin\theta_4 & r_{12}\cos\theta_4 + r_{22}\sin\theta_4 & r_{13}\cos\theta_4 + r_{23}\sin\theta_4 \\ r_{31} & r_{32} & r_{33} \\ r_{11}\sin\theta_4 - r_{21}\cos\theta_4 & r_{12}\sin\theta_4 - r_{22}\cos\theta_4 & r_{13}\sin\theta_4 - r_{23}\cos\theta_4 \end{bmatrix} \times$$

$$\begin{bmatrix} \cos\theta_6 & \sin\theta_6 & 0 \\ \sin\theta_6 & -\cos\theta_6 & 0 \\ 0 & 0 & -1 \end{bmatrix}. \tag{3.36}$$

De las entradas (1,3) y (2,3) de la matriz resultante obtenemos las siguientes igualdades

$$\sin\theta_5 = -r_{13}\cos\theta_4 - r_{23}\sin\theta_4, \tag{3.37}$$

y

$$\cos\theta_5 = r_{33}. \tag{3.38}$$

Así, el valor de θ_5 queda como

$$\theta_5 = \arctan_2(\sin\theta_5, \cos\theta_5). \tag{3.39}$$

Conociendo los valores de las primeras cinco variables de unión y con ayuda de la ecuación 3.27 podemos asumir que

$$[\mathbf{Q}_6]_6 = [\mathbf{Q}_5]_5^T [\mathbf{Q}_4]_4^T [\mathbf{R}]_4 \tag{3.40}$$

Sustituyendo los valores correspondientes se tiene

$$\begin{bmatrix} \cos\theta_6 & \sin\theta_6 & 0 \\ \sin\theta_6 & -\cos\theta_6 & 0 \\ 0 & 0 & -1 \end{bmatrix} = \begin{bmatrix} \cos\theta_5 & \sin\theta_5 & 0 \\ 0 & 0 & 1 \\ \sin\theta_5 & -\cos\theta_5 & 0 \end{bmatrix} \begin{bmatrix} \cos\theta_4 & \sin\theta_4 & 0 \\ 0 & 0 & 1 \\ \sin\theta_4 & -\cos\theta_4 & 0 \end{bmatrix} \times$$

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, \tag{3.41}$$

o bien,

$$\begin{bmatrix} \cos\theta_6 & \sin\theta_6 & 0 \\ \sin\theta_6 & -\cos\theta_6 & 0 \\ 0 & 0 & -1 \end{bmatrix} = \begin{bmatrix} \cos\theta_5\cos\theta_4 & \cos\theta_5\sin\theta_4 & \sin\theta_5 \\ \sin\theta_4 & -\cos\theta_4 & 0 \\ \sin\theta_5\cos\theta_4 & \sin\theta_5\sin\theta_4 & \cos\theta_5 \end{bmatrix} \times$$

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}. \tag{3.42}$$

Tomando las entradas (1,1) y (2,1) de ambos lados de la ecuación anterior se tiene

$$\cos\theta_6 = r_{11}\cos\theta_5\cos\theta_4 + r_{21}\cos\theta_5\sin\theta_4 + r_{31}\sin\theta_5, \tag{3.43}$$

y

$$\cos\theta_6 = r_{11}\sin\theta_4 - r_{21}\cos\theta_4. \tag{3.44}$$

Con lo anterior se obtiene

$$\theta_6 = \arctan_2(\sin\theta_6, \cos\theta_6). \tag{3.45}$$

Con los resultados anteriores se puede representar cualquier postura del robot como un conjunto de valores que representan los ángulos a los cuales debe ser rotada cada unión para alcanzar dicha postura.

Cuando se trabaja con robots, siempre es deseable darle órdenes a un robot en un lenguaje más apropiado y mucho más fácil de comprender para el operador.

Al igual que en la programación de computadoras, han surgido muchos lenguajes de programación de robots. Uno en particular es el lenguaje VAL (Variable Assembly Language), es un lenguaje propio para la línea PUMA de Unimate, y por ser de los primeros en aparecer, su síntaxis permanece en lenguajes más recientes.

3.3. Lenguaje de programación VAL

El lenguaje de programación de robots VAL se almacena permanentemente como una parte del sistema VAL. Esto incluye el lenguaje de programación utilizado para dirigir el sistema en aplicaciones individuales. El lenguaje VAL tiene una sintaxis fácil de entender. Utiliza un conjunto de instrucciones claro, conciso y generalmente intuitivo. Todos los comandos y comunicaciones con el robot están compuestos por palabras fáciles de entender. Programas de control se escriben en el mismo equipo que controla el robot.

Con la técnica "Traducción dirigida por la sintaxis" describiremos la sintaxis del lenguaje independientemente de su semántica, es decir, decribiremos la forma de los programas sin tomar en cuenta su significado.

3.4. Interfaz gráfica de usuario

La interfaz gráfica de usuario es una aplicación para *Windows* escrita en C# y se ejecuta sobre la plataforma *Net Framwork*. La aplicación consta de cuatro modos de ejecución: monitoreo, edición, ejecución automática, y modo manual.

3.4.1. Monitor

La máscara de usuario para el monitoreo del sistema tiene cajas de texto de modo sólo lectura para la visualización del número de cuentas de cada eje, el ángulo de cada unión, posición en el espacio del órgano terminal, estado de los limit-switches y una ventana para la visualización de mensajes emitidos por el controlador (ver figura 3.5). La información proveniente del controlador se actualiza cada 50 milisegundos.

program	→	listinstr
	|	ϵ
listinstr	→	listinstr instr
instr	→	POINT **id** = <location>
	|	MOVE location
	|	MOVET location, expr
	|	MOVES location
	|	MOVEST location, expr
	|	JMOVE location
	|	DRAW expr, expr, expr
	|	APPRO location, expr
	|	APPROS location, expr
	|	DRIVE joint, expr, expr
	|	SPEED expr
	|	OPEN channel
	|	CLOSE channel
	|	WAIT channel
	|	ϵ
joint	→	1 | 2 | 3 | 4 | 5 | 6
channel	→	1 | 2 | 3 | 4 | 5 | 6 | 7 | 8
location	→	<expr, expr, expr, expr, expr, expr>
	|	**id**
expr	→	expr + term
	|	expr - term
	|	term
term	→	term * fact
	|	term / fact
	|	fact
fact	→	fact v̂alue
	|	value
value	→	-exp
	|	exp
exp	→	(expr)
	|	num

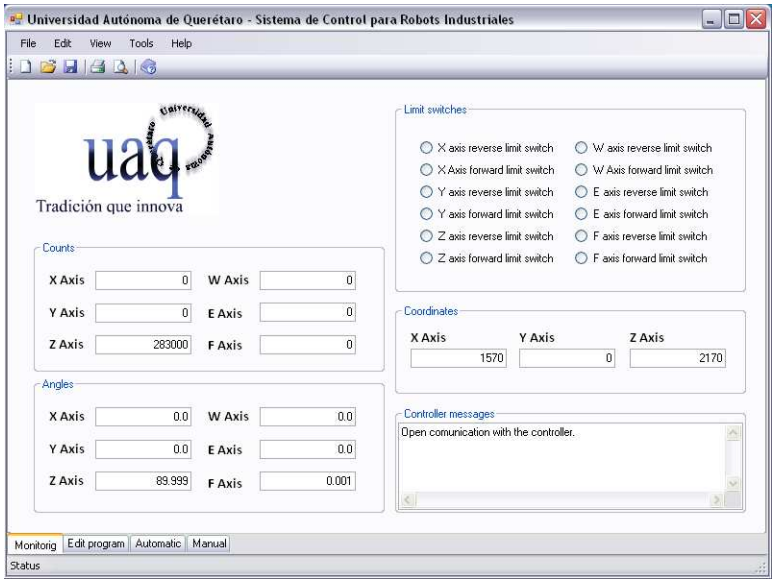


Figura 3.3: Máscara de usuario de la interfaz gráfica de usuario, modo de monitoreo.

3.4.2. Editor

En el modo edición de la aplicación se encuentra un editor de texto para la escritura de programas. Este editor de texto posee todas las características de un editor tradicional, es decir, se puede abrir un archivo, modificar, guardar, imprimir. Además de las herramientas nativas para seleccionar, copiar, cortar y pegar text.

La característica fundamental de esta parte de la aplicación es la compilación del programa escrito a las instrucciones que el controlador entiende y puede llevar a cabo.

3.4.3. Ejecución automática

En el modo de ejecución automática se abren los programas ya compilados para su ejecución cíclica. Algunas de las características de este modo de operación es la visualización de la línea actual en ejecución, el nombre del

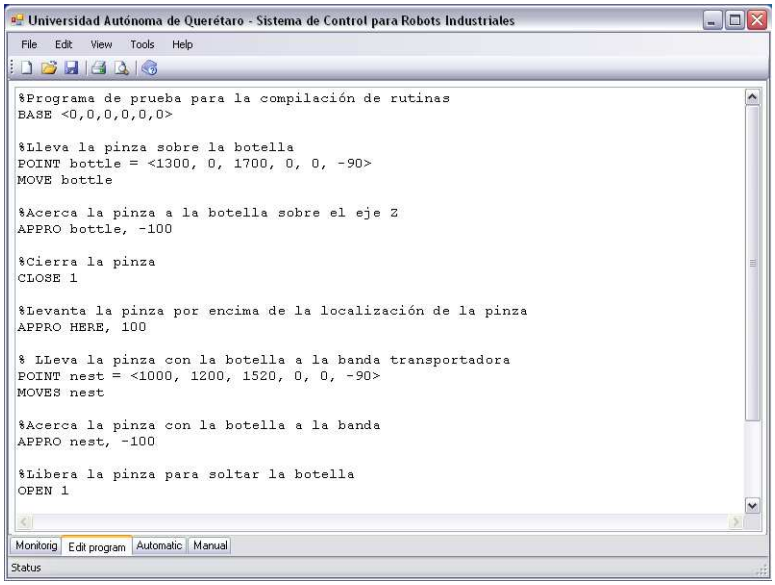


Figura 3.4: Máscara de usuario de la interfaz gráfica de usuario, modo edición.

archivo, si su ejecución se detiene en presencia de peligros de colisión y el despliegue de información proveniente del controlador.

Otra ventaja de la separación de tareas en esta aplicación es que se puede cambiar de tarea mientras se está ejecutando un programa, es decir, el usuario puede ejecutar una rutina mientras está monitoreando el estado general del sistema.

3.4.4. Modo manual

En este modo de operación el usario puede llevar la herramienta del robot de forma manual a su destino. Existen tres formas de mover un robot en la mayoria de los controladores comerciales: *joint*, *world* y *tool* (ver figura 3.6).

El primer tipo de movimiento *joint* hace girar cada unión un determinado número de cuentas, sin embargo, este tipo de movimiento no es muy común en la programción de robots.

El movimiento de tipo *world* es el más utilizado y se basa en el movimiento

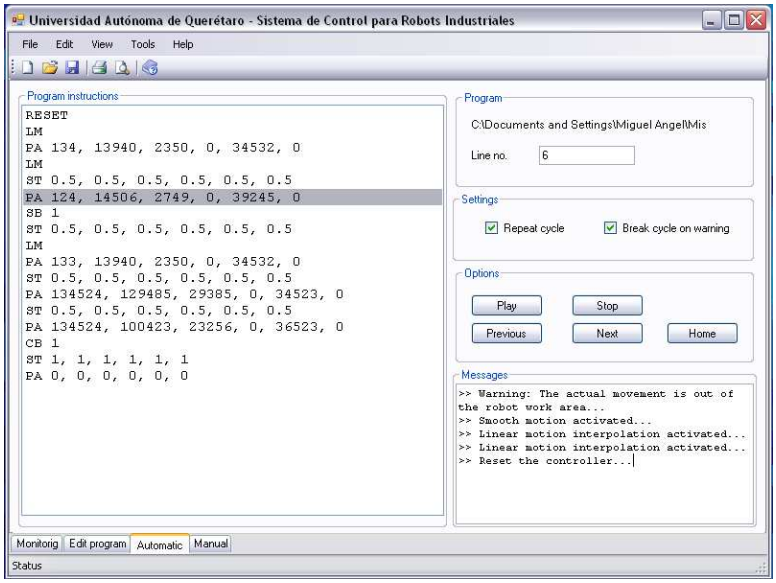


Figura 3.5: Máscara de usuario de la interfaz gráfica de usuario, modo de ejecución.

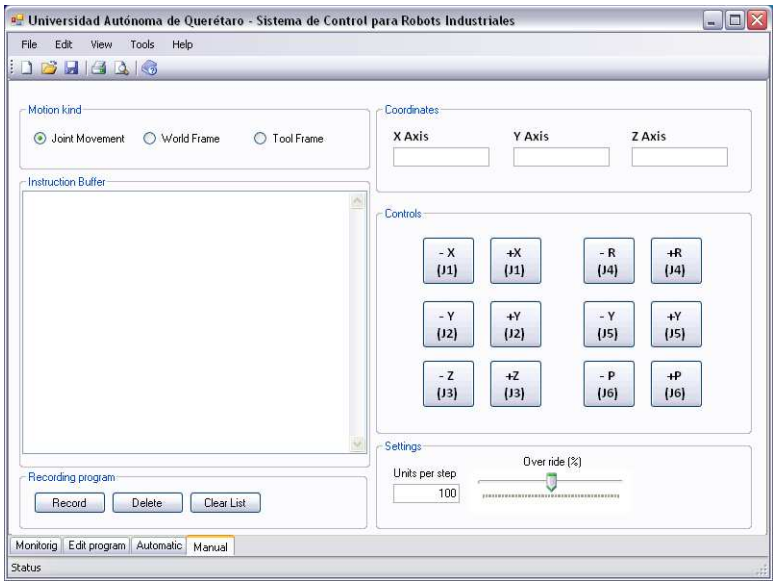


Figura 3.6: Máscara de usuario de la interfaz gráfica de usuario, modo manual.

relativo en el marco coordenado de la base y su orientación con respecto al mismo. El sistema es capaz de resolver el problema de cinemática inversa en cada movimiento para determinar las cuentas de encoder a las que debe ser rotado cada motor.

El movimiento de tipo *tool* es similar al anterior, con la diferencia que el marco coordenado con respecto al cual se realiza el movimiento es el marco coordenado sujeto a la herramienta.

El panel de control está compuesto de un par de gabinetes; uno de ellos tiene todos los componentes eléctricos de alta potencia: contactores, contactores de protección, fuentes y servoamplificadores. En el otro se encuentran los circuitos electrónicos de control: PC, FPGA y tarjetas electrónicas.

3.5. Etapa de potencia

La etapa de potencia está constituida básicamente por los servoamplificadores. Los servoamplificadores son los dispositivos encargados de alimentar los motores según sea requerido por el controlador.

Los servoamplificadores utilizados en el panel son de corriente alterna para motores sin escobillas. Están alimentados por tres fases de 110V cada una con respecto a tierra y sus salidas están conectadas a los motores del robot. Requiere de las señales de efecto Hall y su referencia está conectada al controlador. La figura 3.7 ilustra el diagrama eléctrico de dicha parte del panel.

3.6. Emulación del sensor de efecto Hall

Uno de los requerimientos de los servoamplificadores para el control de los motores del robot son las señales de efecto Hall. El problema de los motores del robot es que no tiene un sensor de efecto Hall acoplado a la flecha, únicamente se cuenta con un encoder relativo de 2048 líneas.

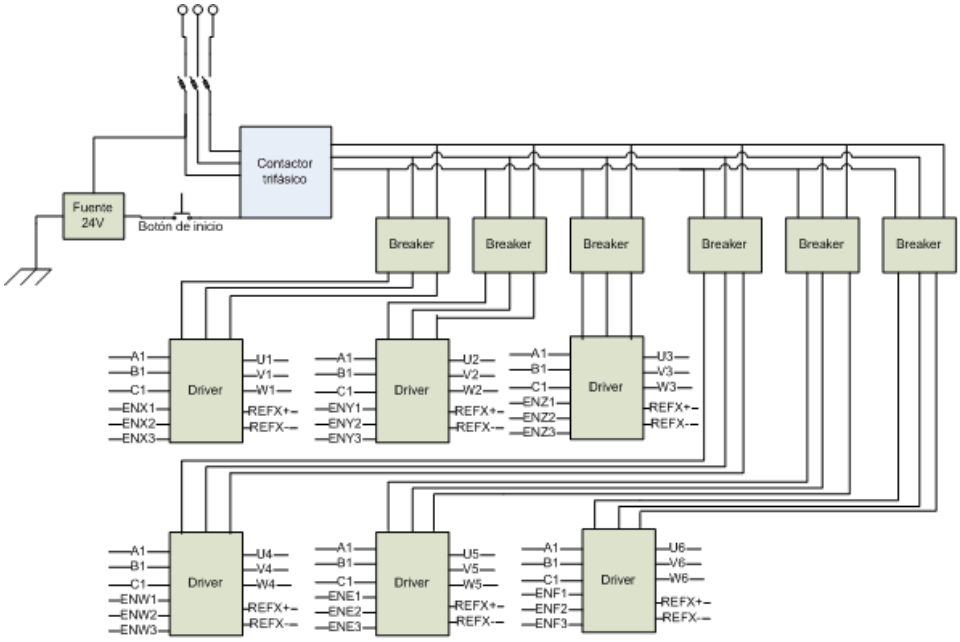


Figura 3.7: Diagrama eléctrico de la etapa de potencia del panel de control.

Para solucionar dicho problema se utilizó un FPGA para la lectura del encoder relativo. A partir de la posición absoluta y en base al número de polos del motor, se generan las señales de efecto Hall (ver figura 3.8).

La descripción de la interfaz de cuadratura de encoder y la visualización de las cuentas de encoder se realizó en VHDL y la generación de las señales de encoder y administración de límites de movimiento en Handel-C.

3.7. Panel de electrónica de control

El otro gabinete armado consta de los circuitos electrónicos para el control de posición, generación de señales de efecto Hall y la interfaz de usuario.

Se instaló una tarjeta madre con sus periféricos, incluyendo la tarjetas de control de movimiento Galil DMC-1800. A esta tarjeta también se conectan las señales de encoder por medio del módulo de conexiones. Este mismo

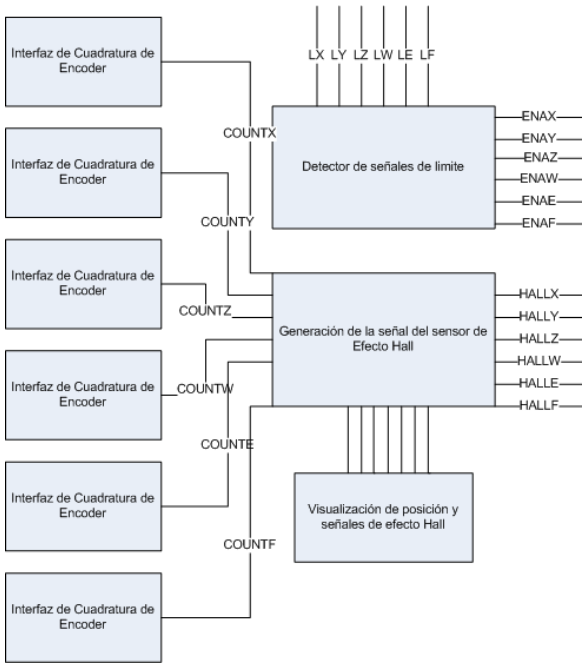


Figura 3.8: Diagrama a bloques de la descripción en hardware de la generación de señales de efecto Hall.

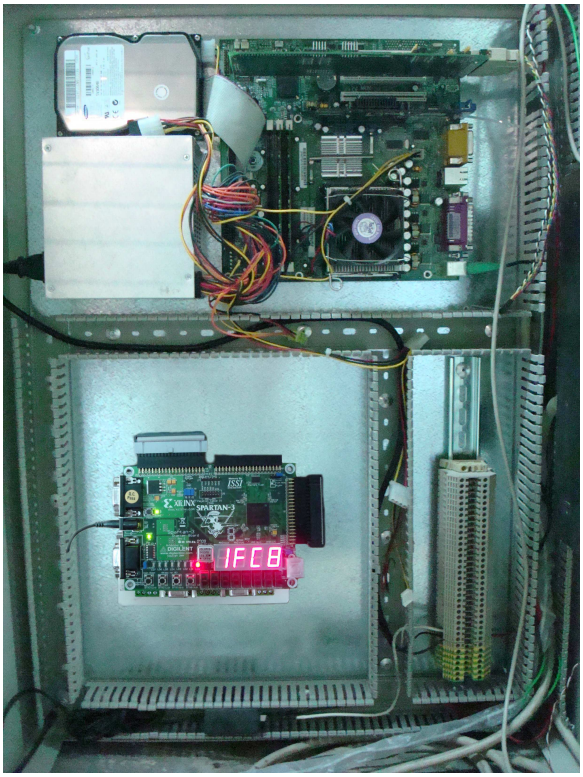


Figura 3.9: Gabinete de electrónica de control.

módulo provee las señales de control que han de conectarse a los servoamplificadores (ver figura 3.9).

Teniendo en consideración que los servoamplificadores manejan voltajes superiores a los 200V y que debe recibir señales del FPGA, se diseñó una tarjeta optoacopladores. También, tomando en cuenta que la frecuencia de las señales puede superar los 100 kHz, se utilizaron opto-acopladores rápidos. La figura 3.10 ilustra el disño esquemático del circuito.

También se diseñó una tarjeta para aislar las señales de recepción de los encoder y dado que cada encoder consume cerca de 180 mA. La figura 3.11

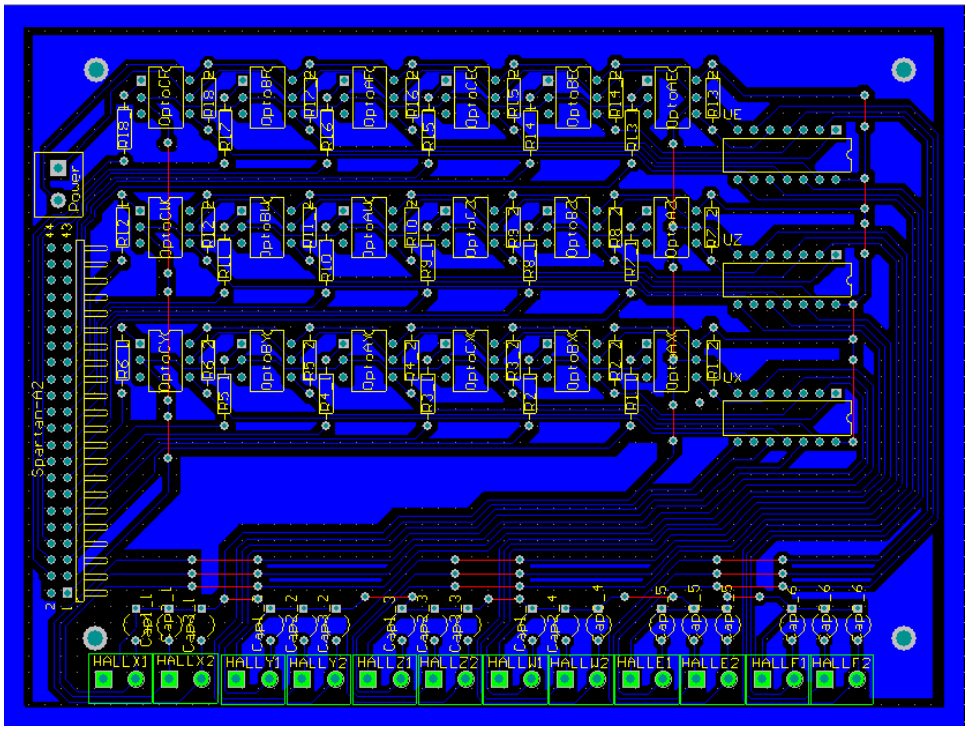


Figura 3.10: Circuito PCB de la tarjeta para aislar la generación de señales de efecto Hall.

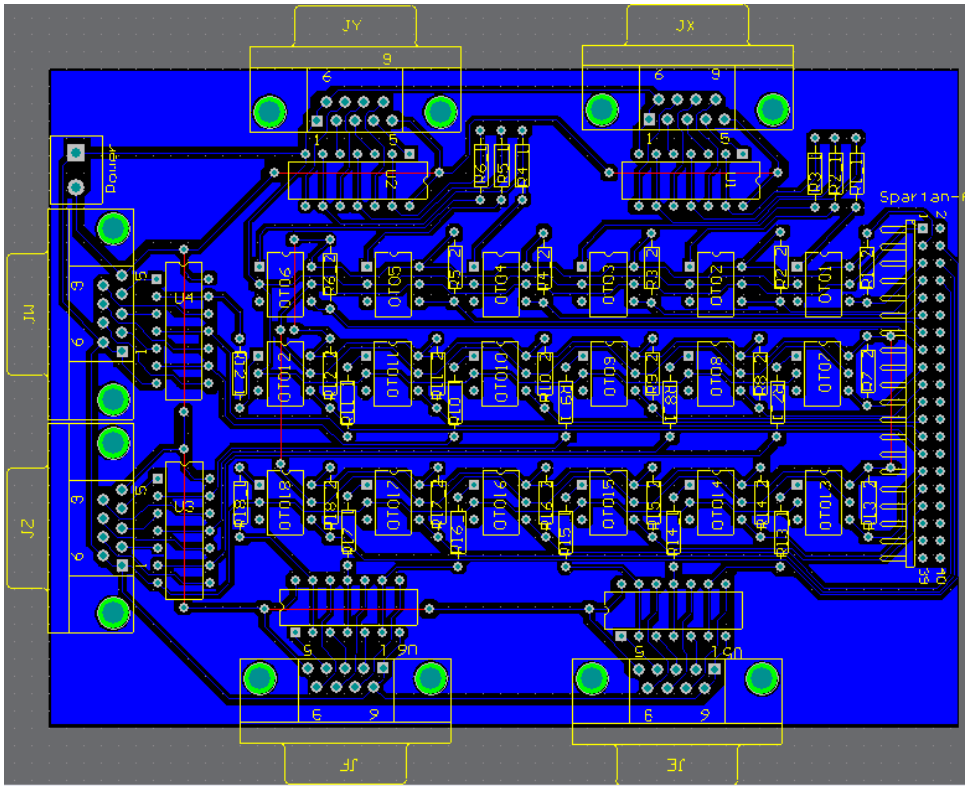


Figura 3.11: Circuito PCB de la tarjeta para aislar las señales de encoder.

Capítulo 4

Resultados y Conclusiones

El prototipo final del sistema de control para robots en cuestión, ilustrado en la figura 4.1, cuenta actualmente con los dispositivos necesarios para manipular tres eslabones de los robots FANUC S 420-F. Aunque se espera que eventualmente se adquiera el material restante para poder manipular el robot completo.

El único problema sujeto al sistema es la magnitud de ruido eléctrico en el panel de potencia. Dicho ruido contamina notablemente las señales de control, provocando pérdidas de información para el controlador. Sin embargo, también se observa que el FPGA que se encarga de emular las señales de efecto Hall no se ve perjudicado por estos efectos; por lo que se puede asumir que un diseño de controlador basado en FPGA aportaría mayor robustez al sistema.

El hardware que se ha diseñado y construido fué sometido a constantes pruebas sin presentar problema alguno. Las pruebas realizadas al control de movimiento de los ejes han sido escalones de alrededor de 100000 cuentas de encoder; generando un error de ± 50 cuentas; es decir, se trata de un 0,0005 % de error, lo cual es relativamente despreciable.

En cuestión a costos, el sistema presentado en este trabajo se encuentra por debajo de una quinta parte del valor de un controlador comercial

Figura 4.1: Prototipo de sistema de control para robots industriales.

para el mismo robot. Además hay que tener en cuenta que se puede adaptar fácilmente a otros robots de la arquitectura.

Bibliografía

[Angeles, 1997] Jorge Angeles, *Fundamentals of Robotic Mechanical Systems.* Springer, 1997.

[Aho, 1998] Alfred V. Aho Ravi Sethi, Jeffrey D. Ullman, *Compiladores, Principios, técnicas y herramientas.* Pearson, 1998.

[Booch, 2006] Grady Booch, James Rumbaugh, Ivar Jacobson, *El Lenguaje Unificado de Modelado.* Pearson, 2006.

[Petzold, 2002] Cherles Petzold, *Programación en Microsoft Windows con C#.* Mc Graw Hill, 2002.

[Belforte, 2006] Belforte, R. Deboli, P. Piccarolo, *Robot design and testing for greenhouse applications* Elsevier Ltd, 2006.

[Woo, 2006] Sangkyun Woo, Daehie Hong, Woo-Chang Lee, Jae-Hun Chung, Tae-Hyung Kim, *A robotic system for road lane painting.* Elsevier Ltd, 2006.

[Abele, 2007] E. Abele, M. Weigold1 y S. Rothenbücher. *Modeling and identification of an industrial robot for machining applications.* Elsevier Ltd, 2007.

[Watanabe, 2007] A.Watanabe1, S. Sakakibara, K. Ban, M. Yamada, G. Shen, *A kinematic calibration method for industrial*

robots using autonomous visual measurement. FANUC Ltd, 2007.

[Chen, 2007] Chen Ying, Zhan Jia-fan, Yan Can-jun, Niu Bin, *Design and hybrid control of the pneumatic force-feedback systems for Arm-Exoskeleton by using on/off valve*. Elsevier Ltd, 2007.

[Dolinsky, 2007] J.U. Dolinsky, I.D. Jenkinson, G.J. Colquhoun, *Application of genetic programming to the calibration of industrial robots*. Elsevier Ltd, 2006.

[Ziliani, 2007] G. Ziliani, A. Visioli, G. Legnani, *A Mechatronic Approach for Robotic Reburring*. Elsevier Ltd, 2006.

[Pashkevich, 2002] Anatoly P. Pashkevich, Alexandre B. Dolgui, Konstantin I. Semkin, *Kinematic aspects of a robot-positioner in an arc welding application*. Elsevier Ltd, 2006.

[Flordal, 2002] H. Flordal, M. Fabian, K. Akesson, D. Spensieri, *Automatic model generation and PLC-code implementation for interlocking policies in industrial robot cells*. Elsevier Ltd, 2007.

[Hakvoort, 2007] W.B.J. Hakvoort, R.G.K.M. Aarts, J. van Dijk, J.B. Jonker *Lifted system iterative learning control applied to an industrial robot*. Elsevier Ltd, 2007.

Listado del generador de señales de efecto Hall

El siguiente listado es utilizado para la sincronización de señales de entrada en el FPGA. Se recomienda el uso de la sincronización en dispositivos cuyo comportamiento se da de forma secuencial.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity Synchronizer is
  port(
  RST : in std_logic;
  CLK : in std_logic;
  A : in std_logic ;
  ASS : out std_logic
  );
end Synchronizer;

architecture Simple of Synchronizer is
signal ASX : std_logic;
begin
  Combinacional : process(ASX)
  begin
    ASS <= ASX;
  end process Combinacional;
  Synchronizing : process(RST, CLK, A)
  begin
    if RST = '1' then
      ASX <= '0';
    elsif CLK'event and CLK = '1' then
      ASX <= A;
    end if;
```

```
  end process Synchronizing;
end Simple;
```

El listado mostrado a continuación determina el sentido de giro del motor en base a una máquina de estados de tipo Moore.

```
Library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity DirectionDetector is
  port(
  RST : in std_logic ;
  CLK : in std_logic ;
  ASS : in std_logic ;
  BSS : in std_logic ;
  DIR : out std_logic
  );
end DirectionDetector;

architecture Simple of DirectionDetector is
signal AB : std_logic_vector(1 downto 0);
signal Qp, Qn : std_logic_vector(2 downto 0);
begin
  Comparator : process(ASS, BSS, Qp, Qn)
  begin
    AB <= ASS & BSS;
    case Qp is
      when "000" => case AB is
        when "01" => Qn <= "001";
        when "10" => Qn <= "100";
        when others => Qn <= Qp;
        end case;
        DIR<='0';
      when "001" => case AB is
        when "11" => Qn <= "010";
          when "00" => Qn <= "111";
        when others => Qn <= Qp;
        end case;
        DIR<='0';
      when "010" => case AB is
        when "01" => Qn <= "110";
        when "10" => Qn <= "011";
        when others => Qn <= Qp;
        end case;
        DIR<='0';
```

```
      when "011" => case AB is
        when "11" => Qn <= "101";
        when "00" => Qn <= "000";
        when others => Qn <= Qp;
        end case;
          DIR<='0';
      when "100" => case AB is
        when "11" => Qn <= "101";
        when "00" => Qn <= "000";
        when others => Qn <= Qp;
        end case;
        DIR<='1';
      when "101" => case AB is
        when "10" => Qn <= "011";
        when "01" => Qn <= "110";
        when others => Qn <= Qp;
        end case;
        DIR<='1';
      when "110" => case AB is
        when "00" => Qn <= "111";
        when "11" => Qn <= "010";
        when others => Qn <= Qp;
        end case;
        DIR<='1';
      when others => case AB is
        when "01" => Qn <= "001";
        when "10" => Qn <= "100";
        when others => Qn <= Qp;
        end case;
        DIR<='1';
    end case;
  end process Comparator;
  Secuencial : process(RST, CLK)
  begin
    if RST = '1' then
      Qp <= "000";
    elsif CLK'event and CLK = '1' then
      Qn <= Qp;
    end if;
  end process Secuencial;
end Simple;
```

El siguiente listado describe un contador de 13 bits con habilitación de entrada ascendente, descendente. El número de bits se debe al máximo número de cuentas que se puede obtener en una revolución. De esta forma, el desbor-

damiento del número de cuentas se da de forma natural en cada revolución.

```
Library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity Counter32Bits is
  port(
  RST : in std_logic ;
  CLK : in std_logic ;
  DIR : in std_logic ;
  ENA : in std_logic ;
  CTS : out std_logic_vector(12 downto 0)
  );
end Counter32Bits;

architecture Counter of Counter32Bits is
signal Qp, Qn : std_logic_vector(12 downto 0);
begin
  Counting : process(DIR, ENA, Qp)
  begin
    if ENA = '1' then
      if DIR = '0' then
        Qn <= Qp + '1';
      else
        Qn <= Qp - '1';
      end if;
    else
      Qn <= Qp;
    end if;
    CTS <= Qp;
  end process Counting;
  Secuencial : process(RST, CLK)
  begin
    if RST = '1' then
      Qp <= (others => '0');
    elsif CLK'event and CLK = '1' then
      Qp <= Qn;
    end if;
  end process Secuencial;
end Counter;
```

El archivo enlistado a continuación se encarga de la detección de un cambio en las los canales de entrada del encoder para la habilitación del contador.

```
Library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity MotionDetector is
  port(
  RST : in std_logic ;
  CLK : in std_logic ;
  AXB : in std_logic ;
  ENA : out std_logic
  );
end MotionDetector;

architecture Simple of MotionDetector is
signal AXBP : std_logic ;
begin
  Comparator : process(AXB, AXBP)
  begin
    ENA <= AXB xor AXBP;
  end process Comparator;
  Secuencial : process(RST, CLK)
  begin
    if RST = '1' then
      AXBP <= '0';
    elsif CLK'event and CLK = '1' then
      AXBP <= AXB;
    end if;
  end process Secuencial;
end Simple;
```

El siguiente listado es la entidad de más alta jerarquía en la interfaz de cuadratura de encoder. En este archivo se incluyen como componentes el resto de las entidades antes descritas.

```
Library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity EncoderQuadInterface is
  port(
  RST : in std_logic ;
  CLK : in std_logic ;
  CHA : in std_logic ;
  CHB : in std_logic ;
  CTS : out std_logic_vector(12 downto 0)
  );
```

```
end EncoderQuadInterface;

architecture Composed of EncoderQuadInterface is
component Synchronizer is
  port(
  RST : in std_logic;
  CLK : in std_logic;
  A : in std_logic ;
  ASS : out std_logic
  );
end component;
component DirectionDetector is
  port(
  RST : in std_logic ;
  CLK : in std_logic ;
  ASS : in std_logic ;
  BSS : in std_logic ;
  DIR : out std_logic
  );
end component;
component MotionDetector is
  port(
  RST : in std_logic ;
  CLK : in std_logic ;
  AXB : in std_logic ;
  ENA : out std_logic
  );
end component;
component Counter32Bits is
  port(
  RST : in std_logic ;
  CLK : in std_logic ;
  DIR : in std_logic ;
  ENA : in std_logic ;
  CTS : out std_logic_vector(12 downto 0)
  );
end component;
signal ASS, BSS, AXB, ENA, DIR : std_logic ;
begin
  AChannelSynchronization : Synchronizer port map(RST, CLK, CHA, ASS);
  BChannelSynchronization : Synchronizer port map(RST, CLK, CHB, BSS);
  MotionDirectionDetection : DirectionDetector port map(RST, CLK, ASS, BSS, DIR);
  MotionDetection : MotionDetector port map(RST, CLK, AXB, ENA);
  CounterUpdate : Counter32Bits port map(RST, CLK, DIR, ENA, CTS);
  AXB <= ASS xor BSS;
end Composed;
```

El siguiente código es el módulo principal del diseño para la generación de las señales de efecto Hall. Está descrito en el lenguaje Handel-C y su implementación está dirigida para un FPGA de la familia Xilinx Spartan 3.

```
set family = XilinxSpartan3;
set clock = external "D9";
set reset = external "L14";

extern chan <unsigned int 16> Data;
extern chan <unsigned int 13> EncoderChanX;
extern chan <unsigned int 3> HallChanX;
extern chan <unsigned int 13> EncoderChanY;
extern chan <unsigned int 3> HallChanY;
extern chan <unsigned int 13> EncoderChanZ;
extern chan <unsigned int 3> HallChanZ;
extern chan <unsigned int 13> EncoderChanW;
extern chan <unsigned int 3> HallChanW;
extern chan <unsigned int 13> EncoderChanE;
extern chan <unsigned int 3> HallChanE;
extern chan <unsigned int 13> EncoderChanF;
extern chan <unsigned int 3> HallChanF;

void main(void)
{
    unsigned int 3 hallX, hallY, hallZ;
    unsigned int 3 hallW, hallE, hallF;
    unsigned int 3 Leds, SwLedHall;
    unsigned int 13 encoderX, encoderY, encoderZ;
    unsigned int 13 encoderW, encoderE, encoderF;
    interface bus_in(unsigned int 3 Input) SwitchAxis()
    with {data = {"H14", "G12", "F12"}};
    interface bus_out() HallLightsInterface(unsigned int 3 Output = Leds)
    with {data = {"L12", "P14", "K12"}};
    par
    {
        hallX = 0; hallY = 0; hallZ = 0; hallW = 0; hallE = 0; hallF = 0;
        encoderX = 0; encoderY = 0; encoderZ = 0; encoderW = 0;
        encoderE = 0; encoderF = 0; Leds = 0; SwLedHall = 0;
    }
    par
    {
        while(1) { EncoderChanX ? encoderX; }
        while(1) { EncoderChanY ? encoderY; }
        while(1) { EncoderChanZ ? encoderZ; }
        while(1) { EncoderChanW ? encoderW; }
```

```
        while(1) { EncoderChanE ? encoderE; }
        while(1) { EncoderChanF ? encoderF; }
        while(1) { HallChanX ? hallX; }
        while(1) { HallChanY ? hallY; }
        while(1) { HallChanZ ? hallZ; }
        while(1) { HallChanW ? hallW; }
        while(1) { HallChanE ? hallE; }
        while(1) { HallChanF ? hallF; }
        while(1) { SwLedHall = SwitchAxis.Input; }
        while(1)
        {
            switch(SwLedHall)
            {
            case 0: par{ Data ! (unsigned int 3)0b000 @ encoderX; Leds = hallX; }break;
            case 1: par{ Data ! (unsigned int 3)0b000 @ encoderY; Leds = hallY; }break;
            case 2: par{ Data ! (unsigned int 3)0b000 @ encoderZ; Leds = hallZ; }break;
            case 3: par{ Data ! (unsigned int 3)0b000 @ encoderW; Leds = hallW; }break;
            case 4: par{ Data ! (unsigned int 3)0b000 @ encoderE; Leds = hallE; }break;
            case 5: par{ Data ! (unsigned int 3)0b000 @ encoderF; Leds = hallF; }break;
            default: par{ Data ! 0xABCD; Leds = 0; } break;
            }
        }
    }
}
```

En el siguiente archivo se describe el intercambio de señales entre el código escrito en VHDL para la interfaz de cuadratura de encoder y la generación de las señales del sensor de efecto Hall.

```
ï»¿set family = XilinxSpartan3;
set clock = external "D9";
set reset = external "L14";

chan <unsigned int 13> EncoderChanX;
chan <unsigned int 3> HallChanX;

void main(void)
{
    unsigned int 1 IN, ENCA, ENCB;
    unsigned int 3 HallSignals;
    unsigned int 13 COUNT;
    interface bus_clock_in(unsigned int 3 Input) EncoderXInterface()
    with {data = {"T3", "N11", "P10"}};
    interface bus_out() HallXInterface(unsigned int Output = HallSignals)
```

```
with {data = {"D5", "D6", "E7"}};
interface bus_out() AuxEncoderX(unsigned int 3 Output = ENCA @ ENCB @ IN)
with {data = {"N7", "T8", "R6"}};
interface EncoderQuadInterface (unsigned int 13 CNT)
HDLEncoderXInterface(unsigned int 1 RST = __reset, unsigned 1 CLK = __clock,
unsigned int 1 CHA = ENCA, unsigned int 1 CHB = ENCB, unsigned int 1 INX = IN);
par
{
    COUNT = 0; HallSignals = 0; ENCA = 0; ENCB = 0; IN = 0;
}
par
{
    while(1) { IN = EncoderXInterface.Input[0]; }
    while(1) { ENCB = EncoderXInterface.Input[1]; }
    while(1) { ENCA = EncoderXInterface.Input[2]; }
    while(1) { COUNT = HDLEncoderXInterface.CNT; }
    while(1) { EncoderChanX ! COUNT; }
    while(1) { HallChanX ! HallSignals; }
    while(1)
    {
        if (COUNT >= 0 && COUNT < 341)
            HallSignals = (unsigned int 3)0b000;
        else if (COUNT >= 341 && COUNT < 683)
            HallSignals = (unsigned int 3)0b100;
        else if (COUNT >= 683 && COUNT < 1024)
            HallSignals = (unsigned int 3)0b110;
        else if (COUNT >= 1024 && COUNT < 1365)
            HallSignals = (unsigned int 3)0b111;
        else if (COUNT >= 1365 && COUNT < 1707)
            HallSignals = (unsigned int 3)0b011;
        else if (COUNT >= 1707 && COUNT < 2048)
            HallSignals = (unsigned int 3)0b001;
        else if (COUNT >= 2048 && COUNT < 2389)
            HallSignals = (unsigned int 3)0b000;
        else if (COUNT >= 2389 && COUNT < 2731)
            HallSignals = (unsigned int 3)0b100;
        else if (COUNT >= 2731 && COUNT < 3072)
            HallSignals = (unsigned int 3)0b110;
        else if (COUNT >= 3072 && COUNT < 3413)
            HallSignals = (unsigned int 3)0b111;
        else if (COUNT >= 3413 && COUNT < 3754)
            HallSignals = (unsigned int 3)0b011;
        else if (COUNT >= 3754 && COUNT < 4096)
            HallSignals = (unsigned int 3)0b001;
        else if (COUNT >= 4096 && COUNT < 4437)
            HallSignals = (unsigned int 3)0b000;
```

```
            else if (COUNT >= 4437 && COUNT < 4779)
                HallSignals = (unsigned int 3)0b100;
            else if (COUNT >= 4779 && COUNT < 5120)
                HallSignals = (unsigned int 3)0b110;
            else if (COUNT >= 5120 && COUNT < 5461)
                HallSignals = (unsigned int 3)0b111;
            else if (COUNT >= 5461 && COUNT < 5803)
                HallSignals = (unsigned int 3)0b011;
            else if (COUNT >= 5803 && COUNT < 6144)
                HallSignals = (unsigned int 3)0b001;
            else if (COUNT >= 6144 && COUNT < 6485)
                HallSignals = (unsigned int 3)0b000;
            else if (COUNT >= 6485 && COUNT < 6826)
                HallSignals = (unsigned int 3)0b100;
            else if (COUNT >= 6826 && COUNT < 7168)
                HallSignals = (unsigned int 3)0b110;
            else if (COUNT >= 7168 && COUNT < 7509)
                HallSignals = (unsigned int 3)0b111;
            else if (COUNT >= 7509 && COUNT < 7850)
                HallSignals = (unsigned int 3)0b011;
            else
                HallSignals = (unsigned int 3)0b001;
        }
    }
}
```

Código fuente del compilador VAL

El siguiente código está escrito en el lenguaje C#. Se trata de una clase dedicada al desglose de palabras en el flujo de entrada del compilador, además de separar palabras clave, valores e identificadores.

```
ï»¿using System;
using System.Collections.Generic;
using System.Text;

namespace VALCompiler
{
    public class Scanner
    {
        //Enumeracion de los tipos de objetos (tokens)
        enum Types { NONE, DELIMITER, VARIABLE, NUMBER, LOCATION };

        public string exp;     //Hace referencia a la cadena de expresion
        public int expIndex;   //Indice actual de la expresion
        public string token;   //Contiene el token actual
        public Types tokType;  //Contiene el tipo de token

        //Constructor
        public Scanner()
        {
            exp = "";
            expIndex = 0;
            token = "";
            tokType = Types.NONE;
        }

        //Devolver true si c es un delimitador
```

```
private bool IsDelimiter(char c)
{
    if (("+-/*%^=()<>,".IndexOf(c) != -1))
        return true;
    return false;
}

public void GetToken()
{
    tokType = Types.NONE;
    token = "";

    if (expIndex == exp.Length) return;

    //Pasar por alto los espacios en blanco
    while (expIndex < exp.Length && Char.IsWhiteSpace(exp[expIndex]))
        ++expIndex;

    //cuando se alcance el espacio en blanco final, termina la expresion
    if (expIndex == exp.Length)
        return;

    if (IsDelimiter(exp[expIndex])) // es operador
    {
        token += exp[expIndex];
        expIndex++;
        tokType = Types.DELIMITER;
    }
    else if (Char.IsLetter(exp[expIndex])) //es variable
    {
        while (Char.IsLetterOrDigit(exp[expIndex])) //Es alfanumerico
        {
            token += exp[expIndex];
            expIndex++;
            if (expIndex >= exp.Length) break;
        }
        tokType = Types.VARIABLE;
    }
    else if (Char.IsDigit(exp[expIndex])) //es un numero
    {
        while (!IsDelimiter(exp[expIndex]))
        {
            token += exp[expIndex];
            expIndex++;
            if (expIndex >= exp.Length)
                break;
```

```
                }
                tokType = Types.NUMBER;
            }
        }
    }
}
```

La clase *Compiler* describe los métodos para la traducción de un comando. Su resultado es la instrucción que habrá de enviarse al controlador.

```
ï»¿using System;
using System.Collections.Generic;
using System.Text;

namespace VALCompiler
{
    class CompilerException : ApplicationException
    {
        public CompilerException(string msg) : base(msg) { }

        public override string ToString()
        {
            return Message;
        }
    }

    class Compiler
    {
        private Scanner scanner;
        private Kinematics kinematics;
        SortedList<string, Location> idTable;

        enum Errors { SYNTAX, UNBALPARENS, NOEXP, DIVBYZERO };

        public Compiler()
        {
            scanner = new Scanner();
            idTable = new SortedList<string, Location>();
        }

        private Location FindVar(string varName)
        {
            if (!Char.IsLetter(varName[0]))
            {
                SyntaxError(Errors.SYNTAX);
```

```
        return new Location();
    }
    return idTable[varName];
}

private void SyntaxError(Errors error)
{
    string[] err = {
    "Error de sintaxis",
    "ParÃ©ntesis no coincidentes",
    "No existe ninguna expresion",
    "Division por cero"};
    throw new CompilerException(err[(int)error]);
}

public string Evaluate(string expstr)
{
    Location location;
    string tokenaux, res = "";
    scanner.exp = expstr;
    scanner.expIndex = 0;
    try
    {
        scanner.GetToken();
        if (scanner.token == "")    //No hay ninguna expresion
        {
            SyntaxError(Errors.NOEXP);
            res = "ERROR";
            return res;
        }
        else if (scanner.token == "POINT")
        {
            scanner.GetToken();
            if (scanner.tokType == Scanner.Types.VARIABLE)
            {
                tokenaux = scanner.token;
                scanner.GetToken();
                if (scanner.token == "=")
                {
                    scanner.GetToken();
                    EvalLocation(out location);
                    idTable.Add(tokenaux, location);
                }
            }
            else
            {
```

```
        res = "SYNTAX ERROR";
        return res;
    }
}
else if (scanner.token == "MOVE")
{
    res = "ST 1, 1, 1, 1, 1, 1;";
    scanner.GetToken();
    EvalLocation(out location);
    Location joints = kinematics.InverseKinematics(location);
    res += "LM; PA " + joints[0] + ", " + joints[1] + ", " + joints[2] +
    ", " + joints[3] + ", " + joints[4] + ", " + joints[5] + "; BG";
    return res;
}
else if (scanner.token == "MOVET")
{
    res = "ST 1, 1, 1, 1, 1, 1;";
    scanner.GetToken();
    EvalLocation(out location);
    Location joints = kinematics.InverseKinematics(location);
    res += "LM; PA " + joints[0] + ", " + joints[1] + ", " + joints[2] +
    ", " + joints[3] + ", " + joints[4] + ", " + joints[5] + "; BG";
    scanner.GetToken();
    if (scanner.tokType == Scanner.Types.DELIMITER)
    {
        scanner.GetToken();
        double val;
        EvalExpression(out val);
        if (val != 0)
            res += "SB 1";
        else
            res += "CB 1";
    }
}
else if (scanner.token == "MOVES")
{
    res = "ST 0.5, 0.5, 0.5, 0.5, 0.5, 0.5;";
    scanner.GetToken();
    EvalLocation(out location);
    Location joints = kinematics.InverseKinematics(location);
    res += "LM; PA " + joints[0] + ", " + joints[1] + ", " + joints[2] +
    ", " + joints[3] + ", " + joints[4] + ", " + joints[5] + "; BG";
    return res;
}
else if (scanner.token == "MOVEST")
{
```

```
    res = "ST 0.5, 0.5, 0.5, 0.5, 0.5, 0.5;";
    scanner.GetToken();
    EvalLocation(out location);
    Location joints = kinematics.InverseKinematics(location);
    res += "LM; PA " + joints[0] + ", " + joints[1] + ", " + joints[2] +
    ", " + joints[3] + ", " + joints[4] + ", " + joints[5] + "; BG";
    scanner.GetToken();
    if (scanner.tokType == Scanner.Types.DELIMITER)
    {
        scanner.GetToken();
        double val;
        EvalExpression(out val);
        if (val != 0)
            res += "SB 1";
        else
            res += "CB 1";
    }
}
else if (scanner.token == "JMOVE")
{
    res = "ST 1, 1, 1, 1, 1, 1;";
    scanner.GetToken();
    EvalLocation(out location);
    res += "LM; PA " + location[0] + ", " + location[1] + ", " + location[2] +
    ", " + location[3] + ", " + location[4] + ", " + location[5] + "; BG";
}
else if (scanner.token == "APPRO")
{
    res = "ST 1, 1, 1, 1, 1, 1;";
    scanner.GetToken();
    EvalLocation(out location);
    scanner.GetToken();
    if (scanner.tokType == Scanner.Types.DELIMITER)
    {
        scanner.GetToken();
        double val;
        EvalExpression(out val);
        location[2] += val;
        Location joints = kinematics.InverseKinematics(location);
        res += "LM; PA " + joints[0] + ", " + joints[1] + ", " + joints[2] +
        ", " + joints[3] + ", " + joints[4] + ", " + joints[5] + "; BG";
    }
}
else if (scanner.token == "APPROS")
{
    res = "ST 0.5, 0.5, 0.5, 0.5, 0.5, 0.5;";
```

```
    scanner.GetToken();
    EvalLocation(out location);
    scanner.GetToken();
    if (scanner.tokType == Scanner.Types.DELIMITER)
    {
        scanner.GetToken();
        double val;
        EvalExpression(out val);
        location[2] += val;
        Location joints = kinematics.InverseKinematics(location);
        res += "LM; PA " + joints[0] + ", " + joints[1] + ", " + joints[2] +
        ", " + joints[3] + ", " + joints[4] + ", " + joints[5] + "; BG";
    }
}
else if (scanner.token == "DRIVE")
{
    res = "ST 1, 1, 1, 1, 1, 1;";
    double joint, change, speed;
    scanner.GetToken();
    EvalExpression(out joint);
    scanner.GetToken();
    if (scanner.tokType != Scanner.Types.DELIMITER)
    {
        res = "SYNTAX ERROR";
        return res;
    }
    scanner.GetToken();
    EvalExpression(out change);
    scanner.GetToken();
    if (scanner.tokType != Scanner.Types.DELIMITER)
    {
        res = "SYNTAX ERROR";
        return res;
    }
    scanner.GetToken();
    EvalExpression(out speed);
    switch ((int)joint)
    {
        case 1:
            res += "SP " + speed + ";";
            res += "PR " + change + "; BG;";
            break;
        case 2:
            res += "SP ," + speed + ";";
            res += "PR ," + change + "; BG;";
            break;
```

```
                case 3:
                    res += "SP ,," + speed + ";";
                    res += "PR ,," + change + "; BG;";
                    break;
                case 4:
                    res += "SP ,,," + speed + ";";
                    res += "PR ,,," + change + "; BG;";
                    break;
                case 5:
                    res += "SP ,,,," + speed + ";";
                    res += "PR ,,,," + change + "; BG;";
                    break;
                case 6:
                    res += "SP ,,,,," + speed + ";";
                    res += "PR ,,,,," + change + "; BG;";
                    break;
                default: res = "";
                    break;
            }
            return res;
        }
        else if (scanner.token == "SPEED")
        {
            double speed;
            scanner.GetToken();
            EvalExpression(out speed);
            res = "SP " + speed + "," + speed + "," + speed + "," + speed +
                "," + speed + "," + speed + ";";
            return res;
        }
        if (scanner.token != "")    //El Ã°ltimo token debe ser nulo
        {
            SyntaxError(Errors.SYNTAX);
            res = "SYNTAX ERROR";
        }
        return res;
    }
    catch (CompilerException exc)
    {
        System.Windows.Forms.MessageBox.Show(exc.Message);
        return "SYNTAX ERROR";
    }
}

private void EvalLocation(out Location location)
{
```

```
double val;
location = new Location();
scanner.GetToken();
if (scanner.tokType == Scanner.Types.DELIMITER)
{
    if (scanner.token != "<")
    {
        SyntaxError(Errors.SYNTAX);
        return;
    }
    scanner.GetToken();
    EvalExpression(out val);
    location[0] = val;
    scanner.GetToken();
    if (scanner.token != ",")
    {
        SyntaxError(Errors.SYNTAX);
        return;
    }
    scanner.GetToken();
    EvalExpression(out val);
    location[1] = val;
    scanner.GetToken();
    if (scanner.token != ",")
    {
        SyntaxError(Errors.SYNTAX);
        return;
    }
    scanner.GetToken();
    EvalExpression(out val);
    location[2] = val;
    scanner.GetToken();
    if (scanner.token != ",")
    {
        SyntaxError(Errors.SYNTAX);
        return;
    }
    scanner.GetToken();
    EvalExpression(out val);
    location[3] = val;
    scanner.GetToken();
    if (scanner.token != ",")
    {
        SyntaxError(Errors.SYNTAX);
        return;
    }
```

```
            scanner.GetToken();
            EvalExpression(out val);
            location[4] = val;
            scanner.GetToken();
            if (scanner.token != ",")
            {
                SyntaxError(Errors.SYNTAX);
                return;
            }
            scanner.GetToken();
            EvalExpression(out val);
            location[5] = val;
            scanner.GetToken();
            if (scanner.token != ">")
            {
                SyntaxError(Errors.SYNTAX);
                return;
            }
        }
        else if (scanner.tokType == Scanner.Types.VARIABLE)
        {
            location = FindVar(scanner.token);
        }
        else
        {
            SyntaxError(Errors.SYNTAX);
            location = new Location();
        }
    }

    private void Atom(out double result)
    {
        switch (scanner.tokType)
        {
            case Scanner.Types.NUMBER:
                try
                {
                    result = Double.Parse(scanner.token);
                }
                catch (FormatException)
                {
                    result = 0.0;
                    SyntaxError(Errors.SYNTAX);
                }
                scanner.GetToken();
                return;
```

```
        default:
            result = 0.0;
            SyntaxError(Errors.SYNTAX);
            break;
    }
}

private void EvalExpression(out double result)
{
    string op;
    double partialResult;

    EvalExp3(out result);
    while ((op = scanner.token) == "+" || op == "-")
    {
        scanner.GetToken();
        EvalExp3(out partialResult);
        switch (op)
        {
            case "-":
                result = result - partialResult;
                break;
            case "+":
                result = result + partialResult;
                break;
        }
    }
}

private void EvalExp3(out double result)
{
    string op;
    double partialResult = 0.0;

    EvalExp4(out result);
    while ((op = scanner.token) == "*" || op == "/" || op == "%")
    {
        scanner.GetToken();
        EvalExp4(out partialResult);
        switch (op)
        {
            case "*":
                result = result * partialResult;
                break;
            case "/":
                if (partialResult == 0.0)
```

```
                    SyntaxError(Errors.DIVBYZERO);
                result = result / partialResult;
                break;
            case "%":
                if (partialResult == 0.0)
                    SyntaxError(Errors.DIVBYZERO);
                result = (int)result % (int)partialResult;
                break;
        }
    }
}

private void EvalExp4(out double result)
{
    double partialResult, ex;
    int t;
    EvalExp5(out result);
    if (scanner.token == "^")
    {
        scanner.GetToken();
        EvalExp4(out partialResult);
        ex = result;
        if (partialResult == 0.0)
        {
            result = 1.0;
            return;
        }
        for (t = (int)partialResult - 1; t > 0; t--)
            result = result * (double)ex;
    }
}

private void EvalExp5(out double result)
{
    string op;
    op = "";
    if ((scanner.tokType == Scanner.Types.DELIMITER) &&
        scanner.token == "+" || scanner.token == "-")
    {
        op = scanner.token;
        scanner.GetToken();
    }
    EvalExp6(out result);
    if (op == "-")
        result = -result;
}
```

```
        private void EvalExp6(out double result)
        {
            if ((scanner.token == "("))
            {
                scanner.GetToken();
                EvalExpression(out result);
                if (scanner.token != ")")
                    SyntaxError(Errors.UNBALPARENS);
                scanner.GetToken();
            }
            else
                Atom(out result);
        }

        private void PutBack()
        {
            for (int i = 0; i < scanner.token.Length; i++)
                scanner.expIndex--;
        }
    }
}
```