UNIVERSIDAD RICARDO PALMA

FACULTAD DE INGENIERÍA

ESCUELA PROFESIONAL DE INGENIERÍA MECATRÓNICA

“Diseño del sistema de control de un Robot Tipo PUMA utilizando LÓGICA DIFUSA”

TESIS PARA OPTAR EL TÍTULO PROFESIONAL DE INGENIERO MECATRÓNICO

PRESENTADO POR ANDRÉ FRANCISCO GONZÁLES CHÁVEZ

LIMA – PERÚ

2013

DEDICATORIA

Dedico el esfuerzo de mi trabajo, a quienes han fortalecido mi espíritu, mente y cuerpo cada día, regalándome alegrías y demostrándome que tenemos la vida para dar todo de nosotros y predicar que lo más importante es hacer, lo que tú realmente deseas hacer. Las personas más importantes de mi vida, mi increíble familia.

ÍNDICE

CAPÍTULO I : INTRODUCCIÓN

La terminología “Robótica” es utilizada por primera vez por el escritor Isaac Asimov en 1942, en su cuento titulado como “Runaround”, cuando definió esta palabra como “el estudio sistemático de robots, su construcción, mantenimiento y comportamiento”.

Desde ese entonces se ha internacionalizado este concepto, siendo esta rama de la ciencia la que se ocupa de todo el desarrollo respectivo al tema de los robots.

Después de la aparición del nombre por primera vez en 1942, sucedieron un sinnúmero de hitos que sirvieron para el desarrollo de la robótica. A continuación mencionamos algunos hechos en orden cronológico para tener una mejor perspectiva del desarrollo de la robótica.

En 1948 se publica el libro llamado Cybernetics, por Norbert Wiener, en el cual, por primera vez se incluyen conceptos para la operación de robots.

En 1954 se patenta, por primera vez en la historia, el diseño de un robot y fue por George C. Devol en Estados Unidos. Esto sirve como ejemplo y es seguido por otras personas en todo el mundo como es el caso de C.W. Kenward del Reino Unido.

En 1961 vuelve a aparecer George C. Devol quien con la ayuda de J.F. ENGELBERG, construyen el primer robot industrial denominado “Unimate-2000”. Al año siguiente, este robot se instala en las fábricas de General Motors. Por esta razón a estos dos investigadores se les considera como padres de la robótica industrial.

A partir del año 1967 aparecerían los primeros robots móviles. Primero con General Electric Corporation, que construiría el primer robot caminante de cuatro patas. Un año más tarde el Stanford Research Institute presenta un robot móvil con ruedas, pero a diferencia de sus antecesores, este disponía de sensores de visión, contacto y medidores de distancia.

El año 1970 sería un gran año para la Robótica, primero porque la Universidad de Stanford presenta por primera vez un robot manipulador accionado con motores eléctricos el T^3 (The Tomorrow Tool Today), que sería el primer robot manipulador industrial controlado con un microcomputador.

En 1973 se funda la empresa ASEA, que sería la primera de muchas empresas dedicadas a la comercialización de robots industriales. Años después cambió su nombre por el que actualmente conocemos ABB.

En 1976 la robótica dejaría de ser utilizada únicamente para aplicaciones industriales o de investigación, sino para otras ramas de la sociedad. Por ejemplo, en ese mismo año la NASA utilizó los primeros manipuladores Viking 1 y 2, robots móviles exploradores, para recoger muestras de la superficie de Marte.

En 1978, Unimation desarrolla el robot PUMA (Programmable Universal Machine for Assembly) con las especificaciones de General Motors. Este robot ha sido uno de los más populares en la industria por años y su modelo ha servido como base para muchos posteriores diseños. Además es el objeto de numerosos estudios teóricos.

El profesor Makino de la Universidad de Yamashi de Japón, presenta en 1982, el concepto del robot SCARA (Selective Compliance Assembly Robot Arm), que permite tener un perfecto manipulador para el ensamblado de piezas.

La presente tesis muestra el diseño de controladores para un brazo robótico tipo PUMA de cuatro grados de libertad (GDL). Se desarrolla un análisis de tres tipos de controladores: controladores clásicos tipo proporcional, integral y derivativo (PID), controladores por Torque computado y finalmente un controlador basado en lógica difusa (Fuzzy) basado en 9 reglas.

La tesis se encuentra dividida en 5 capítulos, conclusiones, recomendaciones, las referencias bibliográficas y 03 anexos.

La estructura de la tesis responde a la resolución secuencial del problema de diseño del sistema de control de un robot tipo PUMA utilizando lógica difusa. A continuación se describe la composición de los cinco capítulos y un breve resumen de los mismos:

Capítulo 1. Introducción

En este capítulo se expone el planteamiento del problema general, problema específico, objetivo general, objetivo específico, la justificación y la estructura de la tesis.

Capítulo 2. Marco teórico

En este capítulo se describe los antecedentes y las bases teóricas que enmarcan la investigación. Así como se hace referencia a la cinemática y dinámica del robot tipo PUMA. Por otro lado, se estudia el marco teórico de la lógica difusa y sus aplicaciones a la robótica.

Capítulo 3. Diseño de la Investigación

En este capítulo de la tesis se describe el diseño del control PID clásico, controlador por torque computado y controlador difuso aplicados al robot PUMA.

Capítulo 4. Resultados

En este capítulo se describen los resultados obtenidos durante la investigación.

Capítulo 5. Discusión de resultados

En este capítulo se presentan las observaciones producto del desarrollo de la tesis.

Finalmente, se presenta las conclusiones, recomendaciones y las futuras líneas de investigación que se pueden generar con esta tesis.

1.1 PLANTEAMIENTO DEL PROBLEMA

1.1.1. Problema general:

¿Cómo se diseña el sistema de control de un robot tipo PUMA de cuatro grados de libertad (GDL) utilizando lógica difusa?

1.1.2 Problema específico:

El problema específico que se resolvió fue la optimización del controlador proporcional, integral y derivativo del robot manipulador PUMA (Programmable Universal Machine for Assembly or Programmable Universal Manipulation Arm) utilizando la lógica difusa. Este robot tiene múltiples aplicaciones en la industria y en sector médico.

1.2. OBJETIVOS

1.2.1 Objetivo general

Diseñar el sistema de control de un robot tipo PUMA de cuatro grados de libertad (GDL) utilizando lógica difusa.

1.2.2 Objetivo específico

Determinar cuál de los siguientes controladores: proporcional, integral y derivativo (PID) clásico, controlador por torque computado y controlador basado en lógica difusa (Fuzzy) de 9 reglas básicas, es más óptimo para el control del robot tipo PUMA.

1.3. JUSTIFICACIÓN

El robot PUMA (Programmable Universal Machine for Assembly or Programmable Universal Manipulation Arm) es ampliamente utilizado en aplicaciones de ensamblaje y manipulación en el sector industrial y el sector biomédico. Cabe mencionar que el hecho de diseñar un controlador Proporcional, Integral y Derivativo (PID) basado en lógica difusa para el robot PUMA, nos permitió encontrar los parámetros K_p, K_i, K_d, que optimizan la señal de control aplicada a los servomotores del brazo robótico, en comparación con los parámetros del controlador PID clásico y el controlador por torque calculado.

A nivel industrial el robot tipo PUMA permite: incrementar la productividad, evitar la realización de trabajos pesados y repetitivos para el ser humano, amortizarse rápidamente por sustitución de la mano de obra obteniendo así una mayor duración de las herramientas con más precisión de los trabajos realizados y menos pérdida de material y reducido

mantenimiento, realización de tareas en condiciones y ambientes peligrosos para el ser humano (hostiles, a muy altas o muy bajas temperaturas).

Finalmente la ejecución de la presente investigación es un aporte científico desarrollado en la Escuela Académico Profesional de Ingeniería Mecatrónica de la Facultad de Ingeniería de la Universidad Ricardo Palma.

El trabajo de tesis permitirá ampliar conocimientos en técnicas de inteligencia artificial empleadas en el sistema de control de robots industriales con fines de aprendizaje para los alumnos de la carrera.

CAPÍTULO II : MARCO TEÓRICO

2.1 ANTECEDENTES

Los robots fueron diseñados y construidos desde hace varias décadas con el objetivo de prestar servicios a los humanos en tareas repetitivas y garantizar su seguridad en algunas tareas extremas; el robot tipo PUMA tiene su origen en la manufactura de automóviles. El desarrollo de los controladores para robots manipuladores, ha sido desde los controladores tradicionales (PD, PID), controladores del tipo adaptable hasta los controladores del tipo robusto.

A continuación se presentan algunas instituciones científicas internacionales que desarrollaron proyectos de investigación afines a la tesis:

Tibaduiza, D., Amaya, I., Rodríguez, S., Mejía, N., Flores, M. (2011). Implementación de un control fuzzy para el control cinemático directo en un robot manipulador. Departamento de Matemática Aplicada III de la Universidad Politécnica de Cataluña, Barcelona, España y la Facultad de Ingeniería Mecatrónica de la Universidad Autónoma de Bucaramanga, Colombia. En este artículo de investigación se muestra el desarrollo e implementación de la lógica difusa como herramienta de control de posición para cada una de las articulaciones de un robot tipo PUMA. Se hace una descripción general del robot y se muestra el cálculo del volumen de trabajo, el cual es usado para la fuzzificación en el desarrollo del controlador. Finalmente es mostrado el desarrollo y la simulación del controlador usando el toolbox fuzzy de Matlab, así como la descripción de una implementación realizada en un PLC.

Torres, S., Méndez, J. (2009). Seguimiento de trayectorias en robots manipuladores revisión de soluciones y nuevas propuestas. Departamento de Ingeniería de Sistemas y Automática y Arquitectura y Tecnología de Computadores de la Universidad de La Laguna – Tenerife, España. El problema del seguimiento de trayectoria en robots manipuladores ha sido abordado aplicando una gran variedad de controladores, desde estructuras sencillas basadas en PD hasta otras más complejas basadas en controladores adaptativos y robustos. Estas últimas técnicas presentan inconvenientes como la presunción de ciertas cotas en los términos de la ecuación

dinámica del robot o la no inclusión de las ligaduras del sistema en el algoritmo de control. En el presente trabajo se hace una revisión de las técnicas clásicas de control de manipuladores y se introduce un conjunto de técnicas novedosas de control robusto y control predictivo, con las que se evitan los problemas comentados. En particular se describe un controlador con una acción robusta autoadaptativa, necesaria para evitar los errores en la cancelación de términos no lineales de la dinámica del robot. Este esquema es mejorado mediante técnicas predictivas que permiten la inclusión de las ligaduras de movimiento del robot en el algoritmo de control. Se incluyen resultados reales y en simulación en un robot tipo PUMA de la Unimation que prueban la bondad de dichos controladores.

2.2 BASES TEÓRICAS

2.2.1 Robot Industrial

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

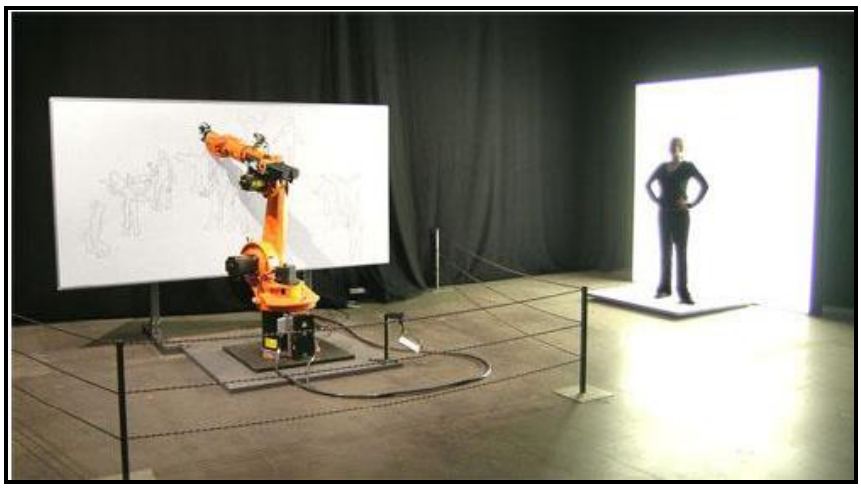


Figura 2.1: Ejemplo de robot dibujante

Un robot se compone principalmente de tres partes: parte mecánica (forma y tamaño de la carcasa), parte electrónica (compuesta por el circuito de control y la sensórica del robot) y la parte de potencia (control de motores del robot). La carcasa constituye la parte mecánica, la forma de esta es uno de los factores determinantes en el éxito del desarrollo de una determinada tarea. La parte electrónica está constituida por el circuito de control que a su vez contiene un microcontrolador el cual se programa empleando un determinado algoritmo, en dicho algoritmo radica todas las decisiones que va a efectuar el robot ante determinados casos dados de acuerdo al problema.

Un robot es un dispositivo generalmente mecánico, que desempeña tareas automáticamente, ya sea de acuerdo a supervisión humana directa, a través de un programa predefinido o siguiendo un conjunto de reglas generales, utilizando técnicas de inteligencia artificial. Generalmente estas tareas reemplazan, asemejan o extienden el trabajo humano, como ensamble en líneas de manufactura, manipulación de objetos pesados o peligrosos, trabajo en el espacio, etc.

Un robot también se puede definir como una entidad hecha por el hombre con un cuerpo y una conexión de retroalimentación inteligente entre el sentido y la acción (no bajo la acción directa del control humano). Usualmente, la inteligencia es una computadora o un microcontrolador ejecutando un programa. Sin embargo, se ha avanzado mucho en el campo de los robots con inteligencia alámbrica. Las acciones de este tipo de robots son generalmente llevadas a cabo por motores o actuadores que mueven extremidades o impulsan al robot.

La RIA (Robot Industries Association) lo define así: un robot es un manipulador reprogramable y multifuncional, diseñado para mover cargas, piezas, herramientas o dispositivos especiales, según trayectorias variadas y programadas. En resumen se puede decir:

Su característica fundamental es poder manejar objetos (o sea, manipulador). Un robot se diseña con este fin, teniendo en cuenta que ha de ser muy versátil a la hora de utilizar herramientas y manejarlas.

La segunda peculiaridad que a diferencia de otras máquinas automáticas es su capacidad para realizar trabajos completamente diferentes adaptándose al medio, e incluso pudiendo tomar decisiones. A eso es a lo que se refiere lo de multifuncional y reprogramable.

Los Web bots son conocidos como robots, pero existen solamente en código, y se mueven a través de páginas Web obteniendo información. Tales entidades son normalmente llamadas agentes de software para ser distinguidos de un robot que posee cuerpo.

Arquitectura de un robot

Todo sistema robótico está compuesto por un brazo y otras cuatro partes esenciales que son las siguientes: el controlador, los actuadores y reguladores, el elemento terminal y los sensores.

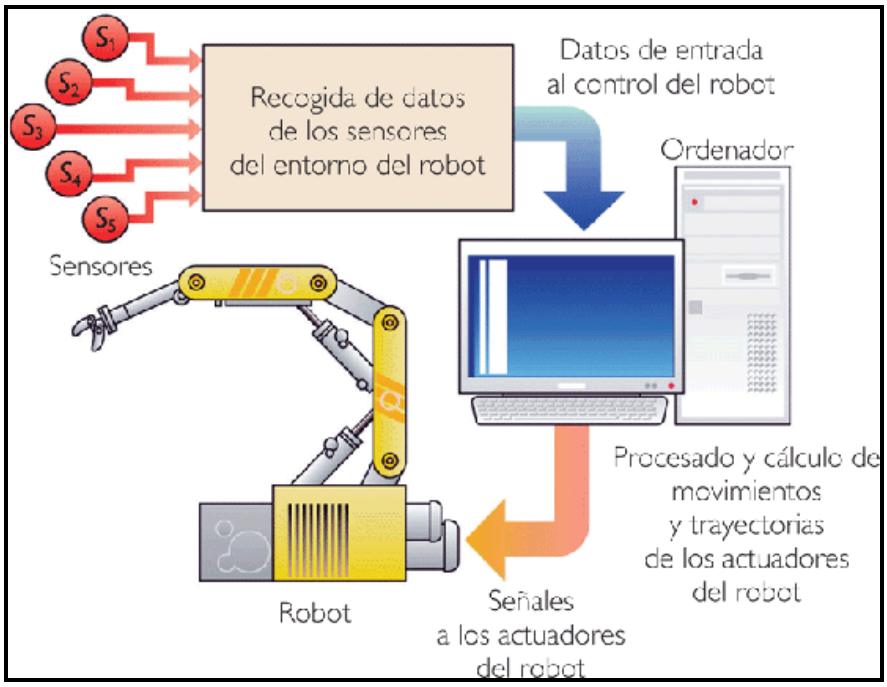


Figura 2.2: Parte de un sistema robótico.

En definitiva, un robot ha evolucionado como una réplica de sus creadores, salvando las distancias. El conjunto guarda cierta similitud con nuestro propio cuerpo.

Manos y brazos se ven reflejados en las partes mecánicas: el manipulador y la herramienta. Los músculos serían los actuadores y las terminaciones nerviosas, los reguladores.

El cerebro (equivalente del controlador) es el encargado de enviar las órdenes a los músculos a través de las terminaciones nerviosas y de recibir información a mediante los sentidos (sensores).

Finalmente, la manera de pensar y actuar vendría determinada por el software de control residente en la computadora.

Sistemas realimentados

Nuestros sentidos toman información, que aprovecha el cerebro para dirigirnos correctamente a través de la calle. Este esquema es válido también para un sistema robotizado.

En cambio, un sistema no sensorizado daría lugar a un control no realimentado y, por tanto, en lazo abierto. Éstos se caracterizan por la falta de adaptabilidad al medio; o, lo que es lo mismo, ante las mismas órdenes de entrada su comportamiento será el mismo, sin tener en cuenta lo que le rodea en esos momentos.

El brazo o manipulador

La estructura mecánica del manipulador puede ser tan variada como los fabricantes que las hacen. Pero generalmente se pueden distinguir cuatro partes principales en el manipulador: el pedestal, el cuerpo, el brazo y el antebrazo.

Las articulaciones entre las distintas partes rígidas del brazo pueden ser giratorias (como las del brazo humano) o deslizantes (si hay traslación de las partes). El número de elementos del brazo y sus articulaciones determinan una característica propia de cada robot. Al número de movimientos espaciales independientes entre sí se le denomina grados de libertad (GDL).

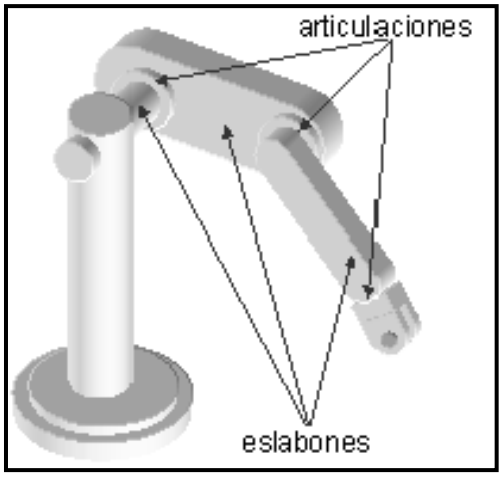


Figura 2.3: Eslabones y articulaciones de un robot

Campo de acción

Debido a la estructura de las articulaciones y al número de ellas existente, el brazo del robot puede llegar a alcanzar ciertos puntos del espacio, pero nunca todos. Al conjunto de los puntos del espacio que el robot puede alcanzar con su herramienta se le denomina campo de acción, y es una característica propia de cada robot.

Los fabricantes de robots ofrecen una variedad de ilustraciones en sus catálogos para poder observar los volúmenes de trabajo del robot.

Más características

Hay otras tres características que definen la calidad del movimiento de un robot:

Resolución (o precisión). Es el mínimo movimiento que puede realizar el robot expresado en milímetros.

Repetitividad. Es una medida estadística del error que comete un robot al colocarse repetidas veces en un mismo punto.

Exactitud. Es una medida de la distancia que hay entre el punto donde se ha colocado el extremo del brazo y el punto real donde debería haberlo hecho.

Aplicaciones industriales

Un robot industrial es un manipulador automático reprogramable y multifuncional, que posee ejes capaces de agarrar materiales, objetos, herramientas, mecanismos especializados a través de operaciones programadas para la ejecución de una variedad de tareas como se puede apreciar, estas definiciones se ajustan a la mayoría de las aplicaciones industriales de robots salvo para las aplicaciones de inspección y para los robots móviles (autónomos) o robots personales.

Para Firebaugh un robot es una computadora con el propósito y la capacidad de movimiento.

Un robot industrial es una máquina que puede efectuar un número diverso de trabajos automáticamente mediante una programación informática previa. Se caracteriza por tener una estructura en forma de brazo mediante el cual puede usar diferentes herramientas o aprehensores situados como elemento terminal de éste. Además, es capaz de tomar decisiones en función de la información procedente del exterior.

El robot industrial forma parte del progresivo desarrollo de la automatización industrial, favorecido notablemente por el avance de las técnicas de control por computadora, y contribuye de manera decisiva la automatización en los procesos de fabricación de series de mediana y pequeña escala.

La fabricación en series pequeñas había quedado hasta ahora fuera del alcance de la automatización, debido a que requiere una modificación rápida de los equipos producción.

El robot, como manipulador reprogramable y multifuncional, puede trabajar de forma continua y con flexibilidad. El cambio de herramienta o dispositivo especializado y la facilidad de variar el movimiento a realizar permiten que, al incorporar al robot en el proceso productivo, sea posible y rentable la automatización en procesos que trabajan con series más reducidas y gamas más variadas de productos.

Estructura mecánica de un Robot Industrial:

Manipulador o brazo mecánico.

El manipulador consta de un conjunto de herramientas interrelacionadas que permiten los movimientos del elemento terminal del brazo del robot. Consta de una base para sujetarse a una plataforma rígida (como el suelo), un cuerpo donde se suele integrar la mayor parte del hardware interno que lo hará funcionar (circuitería, placas impresas, etc.), un brazo para permitir un gran movimiento en 3 dimensiones y un antebrazo para hacer también movimientos en 3 dimensiones aunque diferenciales (muy pequeños) y de mucha precisión, tal que puede llegar a los nanómetros, es decir, a 0'000001mm.

Figura 2.4: Robot industrial.

Controlador basado en un sistema computador.

El controlador es un computador que gobierna el funcionamiento de los órganos motrices y recoge la información de los sensores. También se encarga de realizar todo tipo de cálculos y tomas de decisión según el programa en ejecución. Gracias a la adaptación de microprocesadores (circuitos integrados que tienen unida la CPU del computador) en los circuitos electrónicos se está mejorando notablemente la potencia de los controladores.

Elemento terminal.

En la muñeca del manipulador se acopla una herramienta que será la encargada de desenvolver las tareas. Aunque tiene un peso y medida reducida, a veces debe soportar una elevada carga.

Sensores.

Los robots de última generación tienen la capacidad de relacionarse con el mundo exterior en tiempo real con el fin de obtener planos de acciones específicas en función de las circunstancias exteriores que los rodean.

Por último, vale mencionar los objetivos más destacables de un robot Industrial:

a) Aumentar la productividad.

b) Evitar la realización de trabajos pesados y repetitivos para el ser humano.

c) Amortizarse rápidamente por sustitución de la mano de obra obteniendo, así, una mayor duración de las herramientas, más precisión en los trabajos realizados, menos pérdida de material y reducido mantenimiento.

d) Realización de tareas en condiciones y ambientes peligrosos para el ser humano (hostiles, a muy altas o muy bajas temperaturas, en otros planetas, etc.).

2.2.2. Controladores clásicos

Un controlador de retroalimentación (feedback) está diseñado para generar una salida que cause algún esfuerzo correctivo para que sea aplicado a un proceso, de tal manera que se lleva la salida de dicho proceso hacia un valor deseado conocido como Punto de Referencia (Setpoint). El controlador usa un actuador que afecta al proceso y un sensor para medir el resultado. Virtualmente todos los controladores de retroalimentación determinan su salida mediante la observación del error producido entre el punto de referencia y la medición de la variable del proceso. El error ocurre cuando el operador humano cambia el punto de referencia intencionalmente o cuando una perturbación o carga sobre el proceso, cambia la variable de proceso accidentalmente. Entonces la misión del controlador es eliminar el error automáticamente. Pero para que esto suceda, la elección del controlador a usar debe de darse

siguiendo ciertas especificaciones de diseño, los cuales describen que debe hacer el sistema y como debe hacerlo.

Estas especificaciones son únicas para cada aplicación individual y con frecuencia incluyen especificaciones como estabilidad relativa, precisión en estado estable (error), respuesta transitoria, y características de respuesta en frecuencia. Esta última consideración ha sido históricamente desarrollada con una gran cantidad de herramientas gráficas tales como las trazas de Bode, la traza de Nyquist, la traza de ganancia–fase, y la carta de Nichols, pero debido al desarrollo y la disponibilidad de un software de computadora amigable, rápido y poderoso, la práctica de diseño ha cambiado rápidamente, tal es así que el diseñador puede ejecutar rápidamente, un gran número de diseños empleando especificaciones en el dominio del tiempo. Esto ha disminuido considerablemente la ventaja histórica del diseño en el dominio de la frecuencia, el cual está basado en la conveniencia de realizar el diseño gráfico en forma manual. Además es difícil, excepto para el diseñador experimentado, seleccionar un conjunto coherente de especificaciones en el dominio de la frecuencia que correspondan a requisitos de desempeño en el dominio del tiempo.

Algo que si no ha cambiado, dentro del diseño de sistemas de control, el uso permanente de los controladores Proporcional, Integral, Derivativo (PID), los cuales se utilizan debido a su robustez. En el campo de los sistemas para control de procesos, es un hecho bien conocido que los esquemas de control PID básicos y modificados han demostrado su utilidad para aportar un control satisfactorio, aunque tal vez no aportan un control óptimo en muchas situaciones específicas.

Interpretación en el Dominio del Tiempo

Para entender el significado del control PID en el dominio del tiempo, es necesario definir ciertos conceptos que constituyen el cuerpo del controlador. Como mencionamos anteriormente, entre el controlador más popular podemos mencionar al controlador PID, a continuación, desarrollaremos algunas descripciones breves acerca de sus características más relevantes respecto de las actuaciones en un proceso.

Control Proporcional

Es una técnica de control la cual multiplica la señal de error (la diferencia entre el punto de referencia y la variable de proceso) por una ganancia especificada Kp y es usada como una señal correctiva que ingresa al proceso. El resultado efectivo radica en exagerar el error y reaccionar inmediatamente para corregirlo. Kp no puede eliminar completamente los errores (a menos que el proceso tenga propiedades integrativas inherentes), ya que como el error siguiente se acerca a cero, la corrección proporcional desaparece. Esto produce cierta cantidad de error en estado estable, lo cual se puede apreciar en la Figura 2.5, donde existe un desplazamiento entre el valor deseado (punto de referencia) y el valor de la salida del sistema.

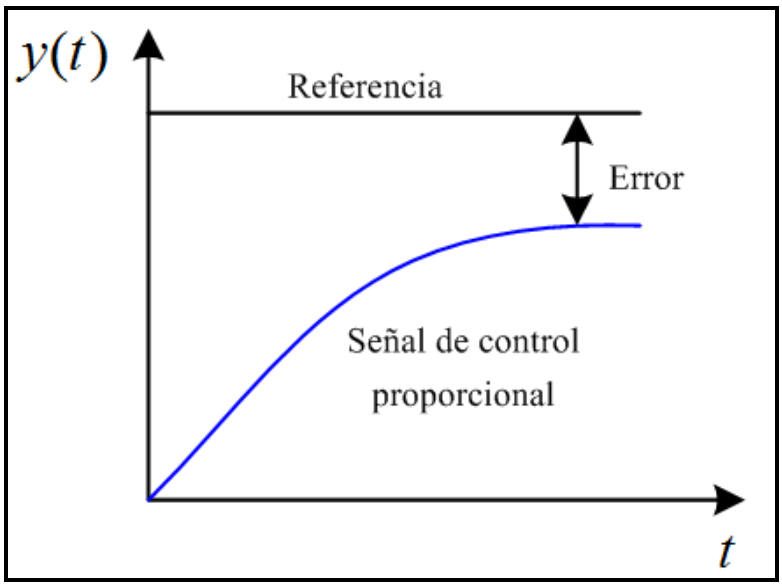


Figura 2.5: Efecto de la acción de control proporcional.

Control Integral

Es una técnica la cual acumula la señal de error (ver Figura 2.6) sobre los tiempos t1 y t2, y multiplica dicha suma por una ganancia especificada Ki y usa este resultado como una señal correctiva sobre el proceso. Debido a que esta técnica actúa sobre errores pasados, el factor de corrección no se va a cero con el error siguiente, eliminando de esta manera el error en estado estable. La ganancia integral tiene un efecto negativo importante, el cual consiste en un factor desestabilizante para el lazo de control para valores grandes de Ki, o para valores usados sin una apropiada amortiguación, los cuales pueden causar severas oscilaciones.

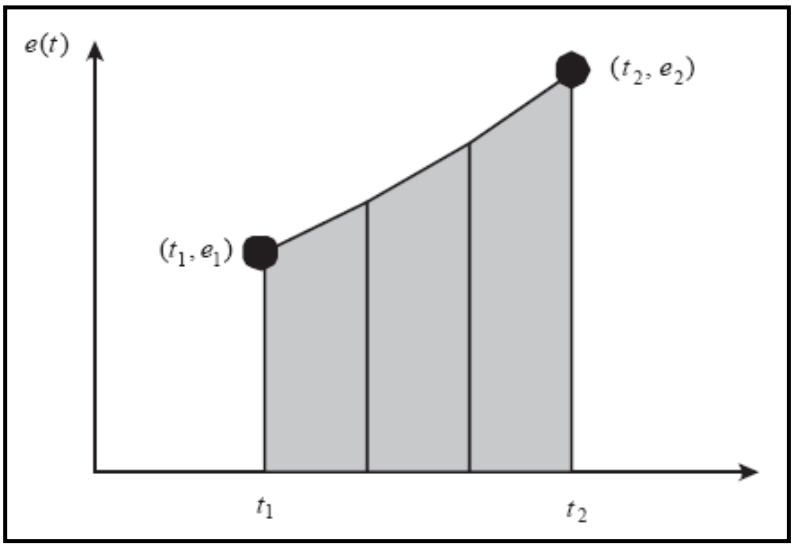


Figura 2.6: Integrando el error en el tiempo.

La señal de error acumulado genera el efecto mostrado sobre la salida del sistema, tal y como se muestra en la Figura 2.7.

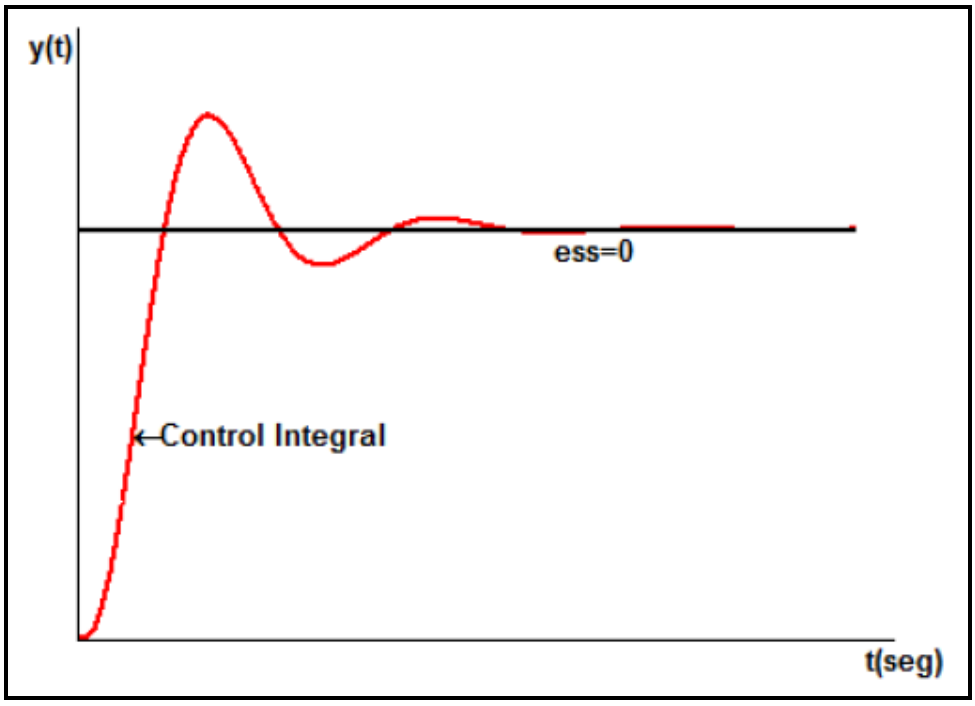


Figura 2.7: Efecto de la acción de control integral.

Control Derivativo

Se basa en la multiplicación de la razón de cambio del error siguiente por una ganancia especificada Kd y usa este resultado como una señal correctiva sobre el proceso. Otra forma de ver el control derivativo es como un control anticipativo. Esto es, al conocer la pendiente el controlador puede anticipar la dirección del error y emplearla para controlar mejor el proceso.

En forma intuitiva, el control derivativo afecta el error en estado estable de un sistema solo si el error en estado estable varía con el tiempo. Si el error en estado estable de un sistema es constante con el tiempo, la derivada con respecto al tiempo de este error es cero, y la porción derivativa del controlador no provee ninguna entrada al proceso, lo cual se muestra en la Figura 2.8 que nos indica que mientras el error existe y varíe con el tiempo, se podrá calcular la derivada de dicho error y de esta manera nos permite conocer hacia donde se dirige dicho error.

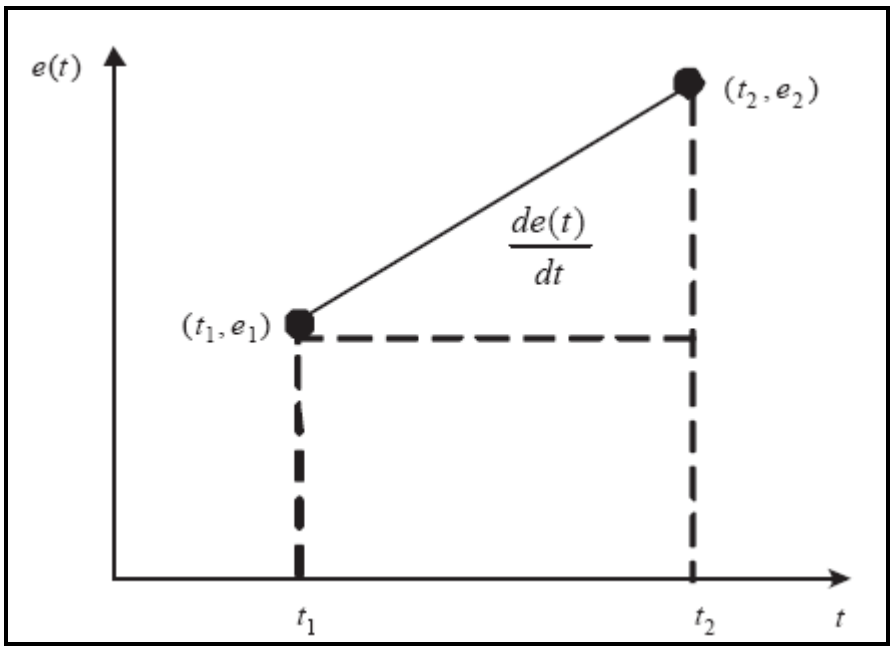


Figura 2.8: Efecto de la acción de control derivativa.

Control Proporcional Integral Derivativo

El control PID de acuerdo a Astrom, la versión del algoritmo de control PID tiene la siguiente forma:

$$u(t) = K\left[e(t) + \frac{1}{t_i}\int_0^t e(\tau)d\tau + T_d\frac{de(t)}{dt}\right]$$

$$e(t) = r(t) - y(t)$$

Donde u es la variable de control, e representa el error entre la señal de referencia r y el valor medido de la salida del proceso y. La ganancia proporcional K, la constante de tiempo

integral Ti, y la constante de tiempo derivativo Td son los parámetros del controlador. Esta versión del algoritmo PID es también llamada el "controlador PID no interactivo ideal" o "ISA" de sus siglas en inglés (Ideal noninteracting Algorithm).

Otras formas de algoritmos PID usados en la industria son los siguientes:

Controlador PID Interactivo:

$$u(t) = K'\left[e(t) + \frac{1}{t'_i}\int_0^t e(\tau)d\tau\right]\left[1 + T'_d\frac{de(t)}{dt}\right]$$

Controlador Ideal PID Paralelo:

$$u(t) = K_p e(t) + K_i\int_0^t e(\tau)\,d\tau + K_d\frac{de(t)}{dt}$$

Donde Kp es la ganancia proporcional, Ki es la ganancia Integral y Kd es la ganancia derivativa.

La simplificación de la ecuación anterior nos da la siguiente notación para los algoritmos PID:

$$u(t) = P + I + D$$

Donde P es el término proporcional, I es el término integral y D es el término derivativo.

Se puede deducir una equivalencia entre los valores del algoritmo no interactivo y el algoritmo ideal paralelo de los controladores PID como se muestra a continuación:

$$K_p = K, \qquad K_i = \frac{K}{T_i}, \qquad K_d = KT_d$$

Todas estas versiones son válidas y usadas por distintos fabricantes de controladores, por ejemplo, Foxboro y Fisher usan las versiones no interactivos, mientras que Honeywell y Texas Instruments usan el algoritmo interactivo.

Como vemos de las ecuaciones anteriores, la mayoría de los controladores operan sobre una señal de error e, la cual es generada en línea (on-line) debido a la sustracción de la salida y del punto de referencia r. Entonces el sistema de control resultante es mostrado en la Figura (2.9) donde z representa la salida del proceso, en tanto que l y n, la carga de perturbación y el ruido en la medición respectivamente, siendo estos últimos los que modifican tanto a la salida del controlador, como a la salida del proceso, generándose las señales u′ = u + l, e y = z + n respectivamente. A este sistema que se le conoce como “sistema con error retroalimentado”.

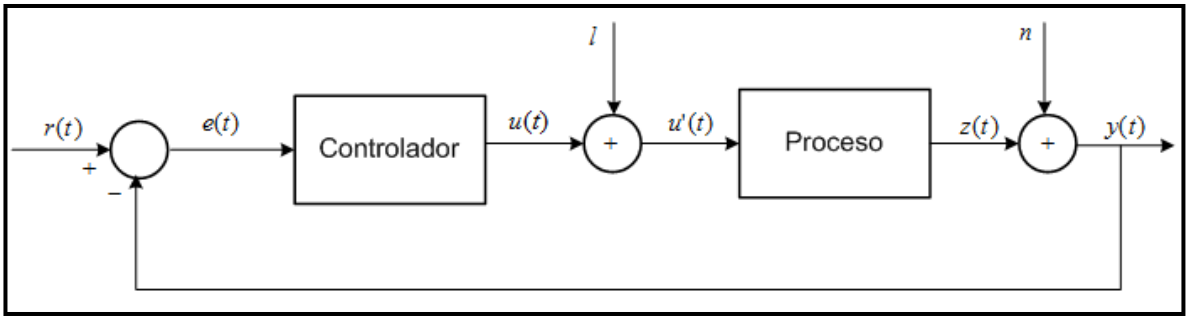


Figura 2.9: Sistema de control con error retroalimentado.

En el resto de la tesis, entenderemos por diseño a la estructura usada en los términos P, I y D y por sintonización entenderemos a la selección de los valores numéricos de los parámetros en los términos P, I y D. Además, nosotros trabajaremos con la configuración del "controlador PID no interactivo", a la cual modificaremos más adelante para dar mayor estabilidad y robustez al control. La elección de esta configuración es debido a los requerimientos de nuestro algoritmo, que como se verá más adelante, se hará una selección de los mejores controladores, pudiéndose generar valores indeseados, ya que como se sabe por ejemplo si se requiere utilizar Ti, y solo tenemos disponibles los valores de Kp, Ki y Kd, la equivalencia Ti = Kp/Ki se puede dar, pero si ambos valores Kp y Ki fueron escogidos ceros, se produce un valor indefinido.

Reglas de Sintonización de los Controladores PID

El proceso de seleccionar los parámetros del controlador que cumpla con las especificaciones de desempeño se conoce como sintonización del controlador. Hay sin embargo algunos métodos de sintonización en los cuales primero, un modelo simple del proceso es determinado por el mismo proceso a ser controlado, para luego escogerse los parámetros PID.

Debido a su uso muy difundido en la práctica, existen varios métodos para sintonización de controladores PID, muchos de los cuales datan de varias décadas atrás. Entre estos métodos "clásicos" podemos encontrar a los siguientes:

Método de Reacción de la Curva de Ziegler-Nichols.

Método de Oscilaciones Sostenidas de Ziegler-Nichols.

Método de Cohen - Coon.

Las dos primeras reglas fueron sugeridas por Ziegler y Nichols para obtener un modelo aproximado de primer orden del sistema y luego sintonizar los controladores PID basándose tanto en las respuestas escalón experimentales, como en la estabilidad marginal cuando sólo se usa la acción de control proporcional, respectivamente. En ambos métodos denominados reglas de sintonización de Z-N se pretende obtener un 25% de sobrepaso máximo en la

respuesta y(t) al escalón unitario, tal y como se observa en la Figura 2.10. Por otro lado, Cohen-Coon hizo una modificación a la segunda regla de Z-N, para que la respuesta tenga un mínimo desplazamiento y otras propiedades favorables.

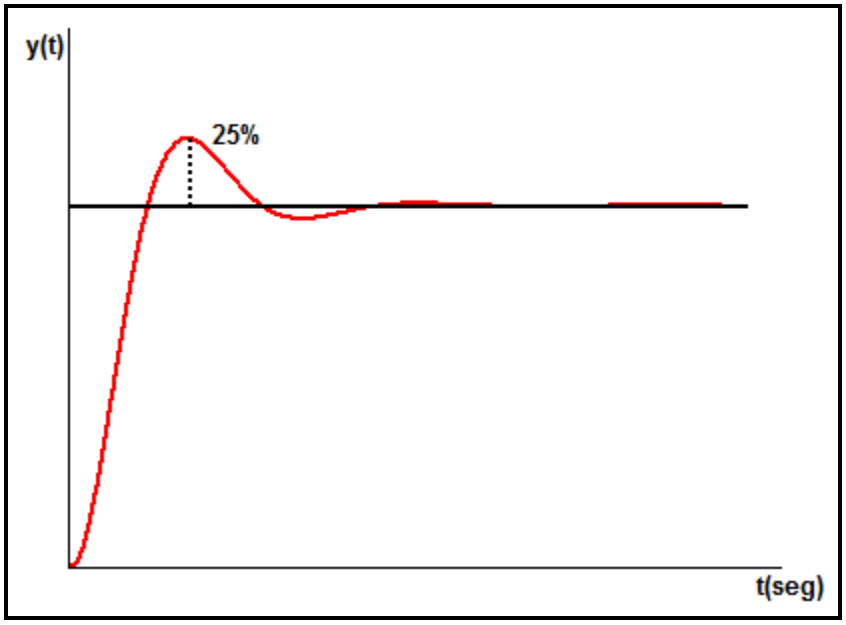


Figura 2.10: Curva de respuesta escalón unitario con 25% de sobrepaso máximo.

Las reglas de Ziegler - Nichols, que se presentan a continuación, son muy convenientes cuando no se conocen los modelos matemáticos de las plantas. (Por supuesto, estas reglas se aplican también al diseño de sistemas de control con modelos conocidos.)

Método de Reacción de la Curva de Ziegler-Nichols

En este primer método, la respuesta de la planta a una entrada escalón unitario se obtiene de manera experimental, es decir al sistema se le excita mediante una entrada escalón unitario y luego se procede a medir la señal de salida, siendo ésta de tipo sobre amortiguado como se observa en la Figura 2.11.

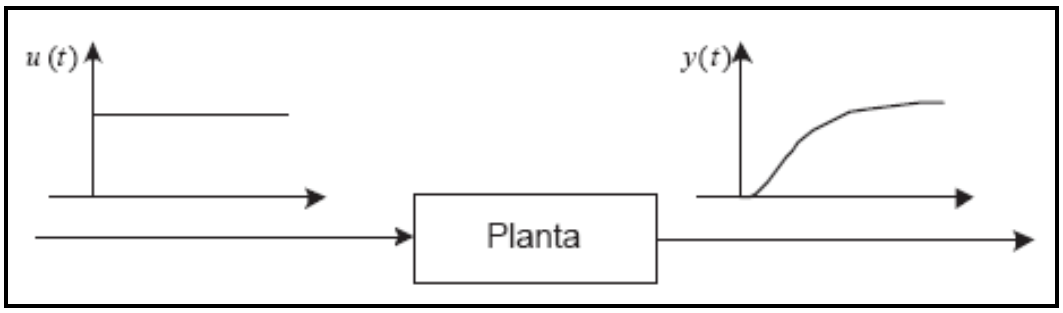


Figura 2.11. Respuesta escalón unitario de una planta.

Si la planta no contiene integradores ni polos complejos conjugados, la curva de respuesta escalón unitario puede tener la forma de S, como se observa en la Figura 2.12 (si la respuesta no exhibe una curva en forma de S, este método no es adecuado.) A esta curva también se le conoce como curva de reacción del proceso. Tales curvas de respuesta escalón se generan experimentalmente o a partir de una simulación dinámica del proceso.

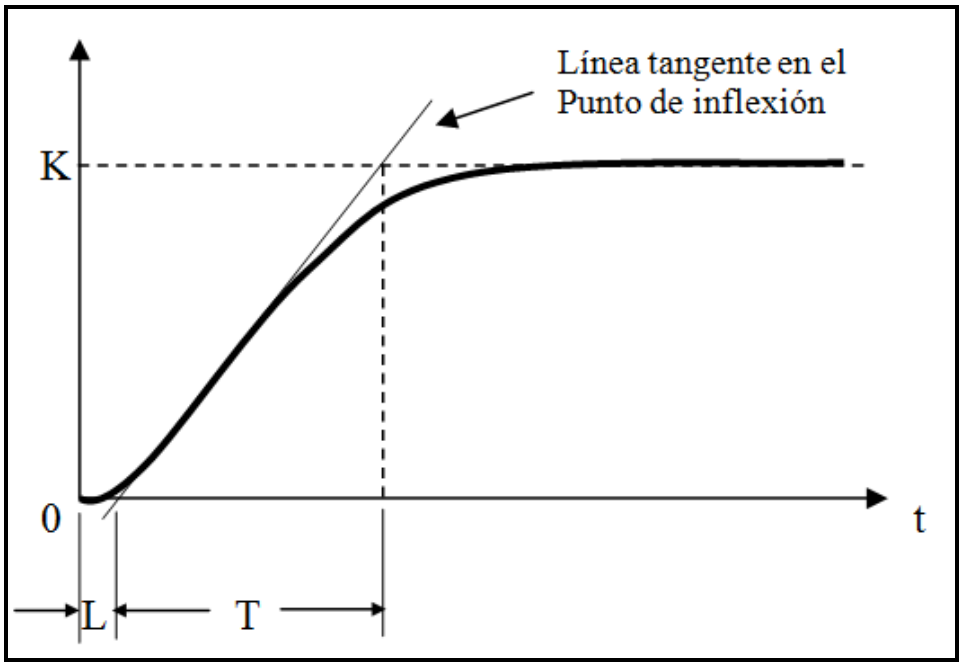


Figura 2.12: Curva de respuesta en forma de S.

En la Figura 2.12 se puede apreciar que la curva en forma de S se caracteriza por dos parámetros: el tiempo de retardo L y la constante de tiempo T. El tiempo de retardo y la constante de tiempo, se determinan dibujando una recta tangente en el punto de inflexión de la

curva con forma de S y determinando las intersecciones de esta tangente con el eje del tiempo y la línea y(t) = K. En este caso, la función de transferencia de la planta Y (s)/U(s) se aproxima mediante un sistema de primer orden con un retardo de transporte del modo siguiente:

$$\frac{Y(s)}{U(s)} = \frac{Ke^{-Ls}}{\tau + 1}$$

Ziegler y Nichols sugirieron establecer los valores de Kp, Ti y Td de acuerdo con la fórmula que aparece en la Tabla 2.1.

Tabla 2.1: Regla de sintonización de Z -N basada en la respuesta escalón de la

Planta.

Tipo de Controlador	K_p	T_i	T_d
P	$\frac{T}{L}$	∞	0
PI	$0.9\frac{T}{L}$	$\frac{L}{0.3}$	0
PID	$1.2\frac{T}{L}$	$2L$	$0.5L$

A partir de las ecuaciones anteriores se obtiene el controlador sintonizado mediante el primer método de las reglas de Ziegler - Nichols:

$$G_c(s) = 0.6T\frac{\left(s + \frac{1}{L}\right)^2}{s}$$

Por tanto, el controlador PID tiene un polo en el origen y un cero doble en $s = -1/L$.

Método de Oscilaciones sostenidas de Ziegler-Nichols

En el segundo método, primero establecemos Ti = ∞ y Td = 0. Usando sólo la acción de control proporcional (ver la Figura 2.13), se incrementa Kp de 0 a un valor crítico Kcr en donde la salida exhiba primero oscilaciones sostenidas.

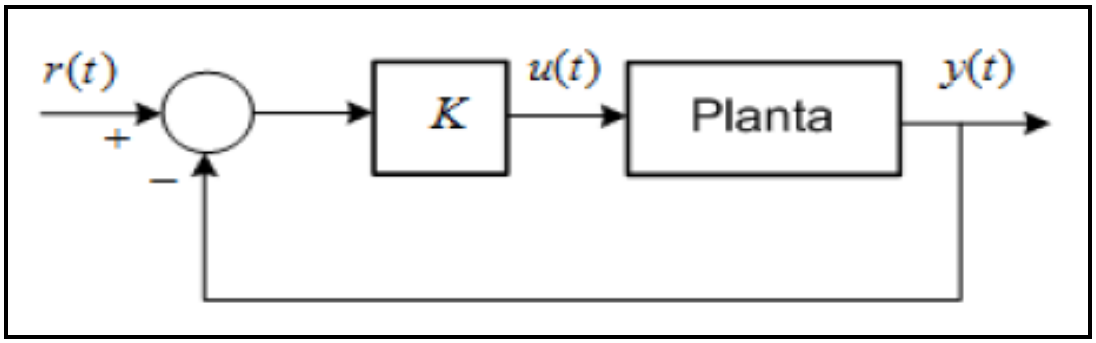


Figura 2.13: Sistema en lazo cerrado con controlador proporcional.

Si la planta no aplica oscilaciones sostenidas para cualquier valor que pueda tomar Kp no se aplica este método. Por tanto, la ganancia crítica Kcr y el periodo Pcr correspondiente se determinan experimentalmente (ver la Figura 2.14).

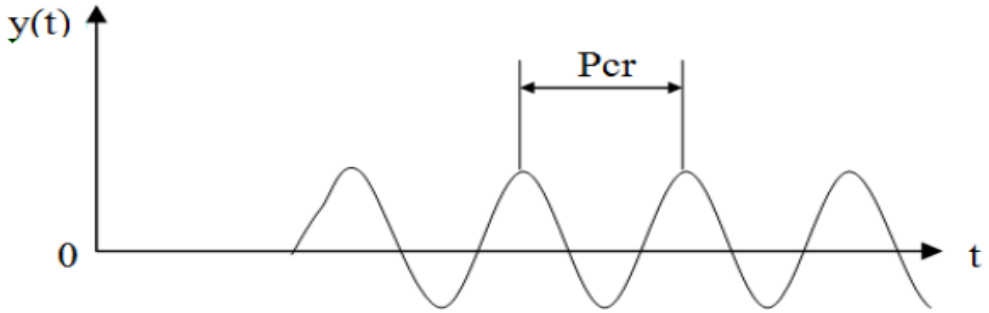


Figura 2.14: Oscilación sostenida con un periodo Pcr.

Ziegler - Nichols sugirieron que se establecieran los valores de los parámetros Kp, Ti y Td de acuerdo con la fórmula que aparece en la Tabla 2.2.

Tabla 2.2: Regla de sintonización de Z -N basada en el método de oscilaciones

sostenidas (Kcr y Pcr).

Tipo de Controlador	K_p	T_i	T_d
P	$0.5K_{cr}$	∞	0
PI	$0.45K_{cr}$	$0.82P_{cr}$	0
PID	$0.6K_{cr}$	$0.5P_{cr}$	$0.125P_{cr}$

Reemplazando los valores de la Tabla 2.2 en la ecuación presentada en el plano–s, el controlador PID sintonizado,

mediante el segundo método de las reglas de Ziegler - Nichols produce:

$$G_c(s) = 0.075K_{cr}P_{cr}\frac{\left(s+\frac{4}{P_{cr}}\right)^2}{s}$$

Por tanto, el controlador PID tiene un polo en el origen y cero doble en s = −4/Pcr.

2.2.3. La Lógica Difusa

La Lógica Difusa o Borrosa, a pesar de su corta historia, presenta un crecimiento muy rápido, ya que es capaz de resolver problemas relacionados con la incertidumbre de la información o del conocimiento, proporcionando un método formal para la expresión del conocimiento en forma entendible y compresible por los humanos.

Las bases teóricas de la Lógica Difusa, en las que están basados los controladores borrosos están mucho más cerca de la manera de razonar de los humanos y del lenguaje natural, que los sistemas lógicos tradicionales. Básicamente, proporciona un medio efectivo de captar más fácilmente la naturaleza inexacta del mundo real.

La matemática de los conjuntos difusos, como su nombre lo indica, trabaja con conjuntos que no tienen límites bien definidos, es decir, la transición entre la pertenencia y la no pertenencia de una variable a un conjunto es gradual. Se caracteriza por las funciones de pertenencia que dan flexibilidad a la modelación utilizando expresiones lingüísticas tales como mucho, poco, leve, severo, escaso, suficiente, caliente, frío, joven.

Los usuarios aceptan con relativa facilidad e interés las aplicaciones basadas en Lógica Difusa, por el paralelismo con su propio razonamiento y por la capacidad de explicación de las conclusiones.

El éxito de la aplicación de la Lógica Difusa se debe, fundamentalmente, a la capacidad de la misma de utilizar modelos de conceptos ambiguos para reducir la complejidad intuitiva de un proceso, de manera que permite realizar operaciones de control, al menos de un modo aproximado o heurístico, sobre procesos no lineales o variantes en el tiempo.

La Lógica Difusa es una herramienta moderna y una de sus aplicaciones más importantes es el Control de Procesos Industriales. Se sale del tradicional esquema de control de lazo realimentado y del rígido modo de pensar de los programas de una microcomputadora para comenzar a emplear variables lingüísticas muchas veces consideradas imprecisas. Aún cuando parece ser sinónimo de imprecisión, la Lógica Difusa está basada en una disciplina fuertemente rigurosa que permite describir exactamente un Sistema de Control sin utilizar complicadas expresiones matemáticas.

A esta nueva rama del Control Automático es a la que se le llama Inteligencia Artificial o simplemente Control Inteligente, dentro del cual se destaca la Lógica Fuzzy o Difusa el cual es un algoritmo muy utilizado en la actualidad.

Asimismo, el control borroso, originado a partir de una lógica de conceptos vagos e imprecisos, se utiliza en la mayoría de los casos para la aproximación de funciones precisas, deterministas, contradiciendo con ello el espíritu inicial de la Lógica Difusa en control de procesos.

Una de las principales metas en control inteligente de procesos industriales es la construcción de sistemas borrosos que controlen con garantía sistemas complejos de alta dimensionalidad, mediante implementaciones generales, robustas y fácilmente entendibles por el usuario.

Lo que los sistemas industriales complejos tienen en común es la presencia de una elevada incertidumbre que hace que las estrategias usuales basadas en modelos y principios de equivalencia cierta no sean técnicamente aplicables.

En muchos casos existe incertidumbre en el modelo del proceso, que puede ser por escaso conocimiento sobre el mismo, disponiéndose solo de un modelo intuitivo que describe comportamientos de orden bajo, a escalas de tiempo grandes. En otras ocasiones, la incertidumbre del modelo recae, aún conociendo bien las ecuaciones que lo describen, en los parámetros del mismo, que son conocidos de forma aproximada. El modelado de la imprecisión mediante conjuntos difusos ha permitido tratar estos problemas.

Aunque la Lógica Difusa tomó auge durante el siglo XX, sus orígenes se remontan hasta 2,500 años. Al respecto, Aristóteles consideraba que existían ciertos grados de veracidad y falsedad. Platón había considerado también grados de pertenencia.

En el siglo XVIII el filósofo y obispo anglicano irlandés David Hume, creía en la lógica del sentido común, el razonamiento basado en el conocimiento que la gente adquiere en forma ordinaria mediante vivencias en el mundo. La corriente del pragmatismo fundada a principios de ese siglo por Charles Sanders Peirce, fue la primera en considerar "vaguedades", más que falso o verdadero, como forma de acercamiento al mundo y al razonamiento humano.

El filósofo y matemático británico Bertrand Russell, a principios del siglo XX, estudió las vaguedades del lenguaje, concluyendo con precisión que la vaguedad es un grado. El filósofo austríaco Ludwing Wittgenstein estudió las formas en las que una palabra puede ser empleada para muchas cosas que tienen algo en común. La primera lógica de vaguedades fue desarrollada en 1920 por el filósofo Jan Lukasiewicz, quien visualizó los conjuntos con posibles grados de pertenencia con valores de 0 y 1; después los extendió a un número infinito de valores entre 0 y 1.

A principios de los años sesenta, Lotfi Zadeh brillante ingeniero eléctrico iraní nacionalizado en Estados Unidos, profesor de Ingeniería Eléctrica en la Universidad de California en Berkeley y en otras prestigiosas universidades norteamericanas, Doctor Honoris Causa de varias instituciones académicas, enunció las bases teóricas de la Lógica Difusa, que combina los conceptos de la lógica y de los conjuntos de Lukasiewicz mediante la definición de grados de pertenencia. La motivación original fue ayudar a manejar aspectos imprecisos del mundo real, creando "un sistema que proporciona una vía natural para tratar los problemas en los que la fuente de imprecisión es la ausencia de criterios claramente definidos". La Lógica Difusa permitió el desarrollo de aplicaciones prácticas.

En 1971, Zadeh publica el artículo, “Quantitative Fuzzy Semantics”, donde introduce los elementos formales que acabarían componiendo el cuerpo de la doctrina de la Lógica Difusa y sus aplicaciones tal como se conocen en la actualidad.

Hasta 1973, Zadeh no presenta la teoría básica de los Controladores Difusos. A partir de ésta publicación, otros investigadores comenzaron a aplicar la Lógica Difusa al control de diversos procesos, por ejemplo, el británico Ebrahim Mandani, quien en 1974 desarrolla el primer sistema de control Fuzzy práctico: la regulación de un motor de vapor.

La solución implementada por Mandani introdujo los conceptos necesarios para su aplicación en áreas industriales. Su aplicación en el área de control nace del fundamento de los operadores humanos son capaces de efectuar en muchos casos un control más efectivo que los controladores automáticos tradicionales, porque están capacitados para tomar decisiones correctas sobre la base de información lingüística imprecisa.

En 1978 comienza la publicación de la revista Fuzzy Sets and Systems, dedicada, con uno o dos números mensuales, al apoyo y desarrollo de la teoría de los conjuntos y sistemas difusos y sus aplicaciones. Esta revista es publicada por la IFSA (the International Fuzzy Systems Association).

También se puede resaltar en 1980 el desarrollo del primer sistema de control difuso comercial, al aplicar esta técnica al control de hornos rotativos en una cementera, desarrollada por los ingenieros daneses Lauritz Peter Holmbland y Jens-Jurgen Ostergaard.

Los occidentales asumieron una actitud reacia principalmente por dos razones: la primera era porque la palabra “Fuzzy” sugería algo confuso y sin forma, y la segunda porque no había forma de probar analíticamente que la teoría funcionaba correctamente, ya que el control fuzzy no estaba basado en modelos matemáticos.

Sin embargo, aparecen toda una serie de investigadores japoneses en el campo de la Lógica Difusa tales como Sugeno, Togai, Bart Kosko (el fuzzsensei), entre otros.

En 1987, se inaugura en Japón el subterráneo de Sendai, uno de los más espectaculares sistemas de control difuso creados por el hombre. Desde entonces el controlador inteligente ha mantenido los trenes rodando eficientemente.

El término "difuso" procede de la palabra inglesa "fuzz". Este término inglés significa "confuso, borroso, indefinido o desenfocado".

La principal motivación de la teoría de conjuntos borrosos [Zadeh, 1965] fue la construcción de un marco formal que permitiera el tratamiento y la manipulación de la incertidumbre presentes en numerosos ámbitos del conocimiento humano. La Lógica Difusa, es una lógica basada en la teoría de conjuntos que posibilita imitar el comportamiento de la lógica humana.

La Lógica Difusa es una rama de la inteligencia artificial que se funda en el concepto “Todo es cuestión de grado”, lo cual permite manejar información vaga o de difícil especificación si quisiéramos hacer cambiar con esta información el funcionamiento o el estado de un sistema específico. En cierto nivel, puede ser vista como un lenguaje que permite trasladar sentencias sofisticadas en lenguaje natural a un lenguaje matemático formal. Con la Lógica Difusa, es entonces posible gobernar un sistema por medio de reglas de “sentido común”, las cuales se refieren a cantidades indefinidas. Establecen una frontera gradual entre la no-pertenencia y la pertenencia, y por tanto conforman una herramienta para el modelado de la imprecisión o la incertidumbre.

La Lógica Difusa es esencialmente una lógica multivaluada que extiende a la Lógica Clásica, la cual debe su nombre a que impone a sus enunciados, únicamente, valores de *falso* o *verdadero*. Si bien la Lógica Clásica ha modelado satisfactoriamente a una gran parte del razonamiento "natural", también es cierto que el razonamiento humano utiliza valores de verdad que no necesariamente son "deterministas".

El adjetivo "difuso" aplicado a esta lógica se debe a que en ellas los valores de verdad no-deterministas utilizados tienen, por lo general, una connotación de incertidumbre. Por ejemplo, un vaso medio lleno, independientemente de que también esté medio vacío, no está lleno completamente ni está vacío completamente. Qué tan lleno puede estar, es un elemento de incertidumbre, es decir, de difusidad, entendida esta última como una propiedad de indeterminismo.

Uno de los objetivos de la Lógica Difusa es proporcionar un soporte formal al razonamiento en el lenguaje natural que se caracteriza por un razonamiento aproximado que utiliza premisas imprecisas como instrumento para formular el conocimiento. La Lógica Difusa nació, entonces, como la lógica del Razonamiento Aproximado, y en ese sentido, podía considerarse una extensión de la Lógica Multivaluada.

El concepto de Razonamiento Aproximado se puede interpretar como el proceso de obtener conclusiones imprecisas a partir de premisas también imprecisas.

Zadeh introdujo la teoría del Razonamiento Aproximado y otros muchos autores han hecho contribuciones importantes a este campo. En lenguaje natural se describen objetos o situaciones en términos imprecisos: grande, joven, tímido. El razonamiento basado en estos términos no puede ser exacto, ya que normalmente representan impresiones subjetivas, quizás probables pero no exactas.

Debido a esto, la Teoría de Conjuntos Difusos se presenta más adecuada que la Lógica Clásica para representar el conocimiento humano, ya que permite que los fenómenos y observaciones tengan más de dos estados lógicos.

La Lógica Difusa actualmente está relacionada y fundamentada en la teoría de los Conjuntos Difusos. Las reglas involucradas en un sistema borroso, pueden ser aprendidas con sistemas adaptativos que aprenden al “observar” cómo operan las personas los dispositivos reales, o estas reglas pueden también ser formuladas por un experto humano. El procedimiento de razonamiento permite inferir resultados lógicos a partir de una serie de antecedentes.

Aunque superficialmente pueda parecer que la teoría del Razonamiento Aproximado y la Lógica Clásica se diferencian enormemente, la Lógica Clásica o Lógica Booleana puede ser vista como un caso especial de la primera.

La lógica tradicional de las computadoras opera con ecuaciones muy precisas y dos respuestas: si o no, uno o cero. Ahora, para aplicaciones digitales muy mal definidas o sistemas vagos se emplea la Lógica Difusa, donde las proposiciones pueden ser representadas con grados de certeza o falsedad.

Por ejemplo, la sentencia "hoy es un día soleado", puede ser 100% verdad si no hay nubes, 80% verdad si hay pocas nubes, 50% verdad si existe neblina y 0% si llueve todo el día.

Por medio de la Lógica Difusa pueden formularse matemáticamente nociones como: un poco caliente o muy frío, para que sean procesadas por computadoras y cuantificar expresiones humanas vagas, tales como "Muy alto" o "luz brillante". De esa forma, es un intento de aplicar la forma de pensar humana a la programación digital permitiendo cuantificar aquellas descripciones imprecisas que se usan en el lenguaje y las transiciones graduales.

Los esquemas de razonamiento utilizados son "esquemas de razonamiento aproximado", que intentan reproducir los esquemas mentales del cerebro humano en el proceso de razonamiento. Estos esquemas consistirán en una generalización de los esquemas básicos de inferencia en Lógica Binaria (silogismo clásico).

Tan importante es la selección de un esquema de razonamiento como su representación material, ya que el objetivo final es poder desarrollar un procedimiento analítico concreto para el diseño de controladores "heurísticos", que nos permita inferir el control adecuado de un

determinado proceso en función de un conjunto de reglas "lingüísticas", definidas de antemano tras la observación de la salida y normas de funcionamiento de éste.

La Lógica Difusa trata de crear aproximaciones matemáticas en la resolución de ciertos tipos de problemas. Pretende producir resultados exactos a partir de datos imprecisos, por lo cual es particularmente útil en aplicaciones electrónicas o computacionales. Está definida como un sistema matemático que modela funciones no lineales, que convierte unas entradas en salidas acordes con los planteamientos lógicos que usa el razonamiento aproximado.

La Lógica Difusa ha cobrado fama por la variedad de sus aplicaciones. En general se aplica tanto a sistemas de control de complejos procesos industriales como para modelar cualquier sistema continuo de ingeniería, física, biología o economía.

Un **universo** es una colección de objetos de los que se hablará en una lógica específica. Por ejemplo, el universo de los números naturales o el universo de las edades.

Un **conjunto** en el universo es, desde un punto de vista intuitivo, una colección de objetos tal que sea posible decidir cuándo un objeto del universo está o no en esa colección. Abstrayendo la noción de conjunto, se puede considerar que un conjunto es exactamente una función del universo en el conjunto de valores 0,1 que asocia precisamente el valor 1 a los objetos que estén en el conjunto y el valor 0 a los que no.

Desde el punto de vista de la teoría clásica de conjuntos se establece que los distintos elementos de un universo pueden pertenecer a un conjunto o no, siempre y cuando satisfagan o no una determinada propiedad, por ejemplo, el conjunto de los números pares está formado por los números que son divisibles por dos.

De esta manera, si consideramos el universo de los números naturales positivos U= {1, 2, 3, 4, 5,…} podríamos decir que 3 pertenece al conjunto de los números impares, mientras que 8 no. Igualmente, 9 pertenece al conjunto de los números mayores que 5, mientras que 3 no.

Funciones de Pertenencia

La pertenencia a un conjunto de diferentes elementos suele representarse gráficamente mediante la denominada función de pertenencia o de membresía, como la que se muestra en la Figura 2.15 para los números mayores que 5. En esta función de pertenencia, toman valor 1 aquellos elementos que pertenecen al conjunto, mientras que toman valor 0 aquellos que no pertenecen.

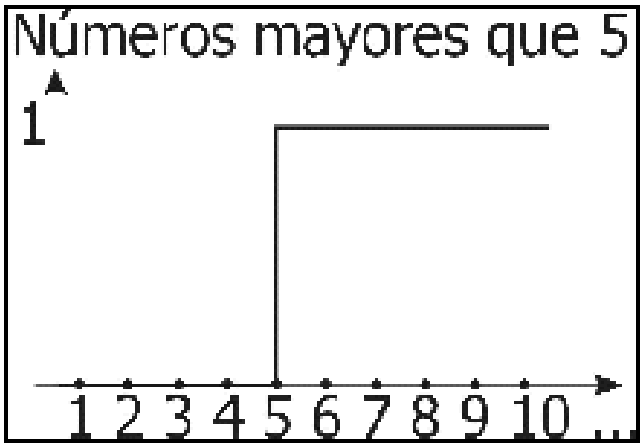


Figura 2.15 Representación gráfica de la función de pertenencia del conjunto "números mayores que 5".

Este concepto es suficiente en muchas áreas de aplicación, pero fácilmente se puede encontrar situaciones donde se necesita más flexibilidad. La mayoría de los fenómenos que encontramos cada día son imprecisos, es decir, tienen implícito un cierto grado de difusidad en la descripción de su naturaleza. Esta imprecisión puede estar asociada con su forma, posición, momento, color, textura, o incluso en la semántica que describe lo que son.

Por ejemplo, para la representación de la función de pertenencia del conjunto “caliente”, haciendo uso de la teoría clásica, como se aprecia en la Figura 2.16, quedaría:

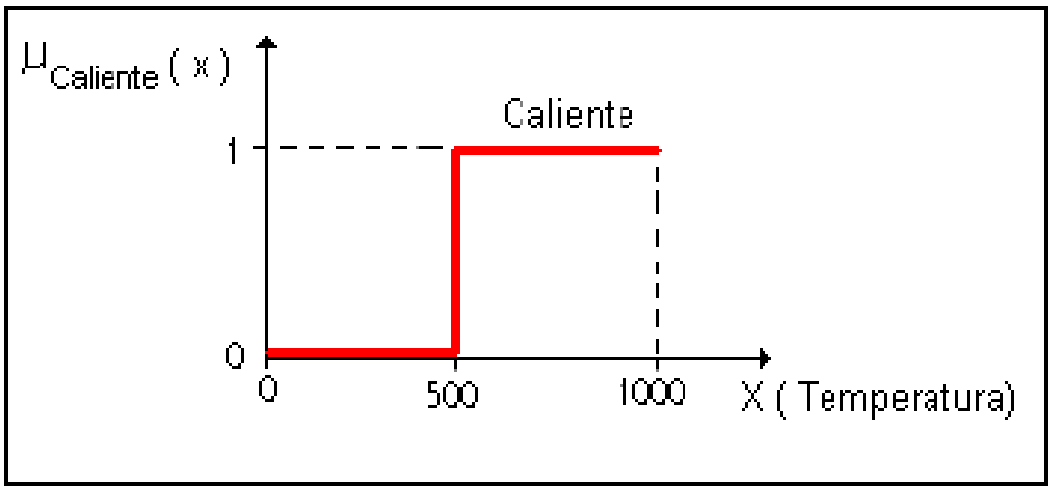


Figura 2.16 Representación gráfica de la función de pertenencia del conjunto " caliente".

Sin embargo, realmente no se puede tener una definición exacta de cuándo un valor de temperatura pasa del conjunto “frío” a “caliente”.

Aceptamos la imprecisión como una consecuencia natural de “la forma de las cosas en el mundo”. Simplemente se aproximan estos eventos a funciones numéricas y se escoge un resultado en lugar de hacer un análisis del conocimiento empírico. Consideremos las siguientes sentencias:

La temperatura está caliente

Los grandes proyectos generalmente tardan mucho

IBM es una compañía grande y agresiva

Alejandro es alto pero Ana no es bajita

Este tipo de proposiciones forman parte de nuestras relaciones cotidianas, sin embargo, son incompatibles con el modelado tradicional de sistemas de información. Si se pudieran incorporar estos conceptos se lograría que los sistemas sean potentes y se aproximen más a la realidad.

Los conjuntos clásicos se definen mediante un predicado que da lugar a una clara división del Universo de Discurso X en los valores "Verdadero" y "Falso". Sin embargo, el razonamiento humano utiliza frecuentemente predicados que no se pueden reducir a este tipo de división: son los denominados predicados difusos.

La teoría de conjuntos difusos propone la extensión del concepto de pertenencia para que admita graduación entre la no-pertenencia y la pertenencia total al conjunto. La fuzificación es independiente de cualquier capacidad para medir, ya que, un conjunto difuso, es un conjunto que no tiene límites bien definidos y es también una función que asocia a cada objeto del universo un valor en el intervalo [0,1].

Para el ejemplo del conjunto “caliente” es imposible dar al conjunto una definición clásica, ya que su correspondiente predicado no divide claramente el universo de las temperaturas en conjuntos “frío” o “caliente”. La manera más apropiada de dar solución a este problema es considerar que la pertenencia o no pertenencia de un elemento x al conjunto, no es absoluta sino gradual, definiéndose este conjunto como un Conjunto Difuso.

Por tanto, se relajaría la separación estricta entre éstos conjuntos, permitiendo la pertenencia Si o NO al conjunto pero suavizando su función de pertenencia con frases del tipo: “pertenece un poco menos a…” o “casi pertenece a…”. Es decir, ya no adoptará valores en el conjunto discreto {0,1} (lógica booleana), sino en el intervalo cerrado [0,1] como se aprecia en la Figura 2.17.

El valor 1 representa que el elemento pertenece nítidamente al conjunto, el valor 0 representa la no pertenencia absoluta al conjunto, y los demás valores indican una pertenencia parcial al conjunto.

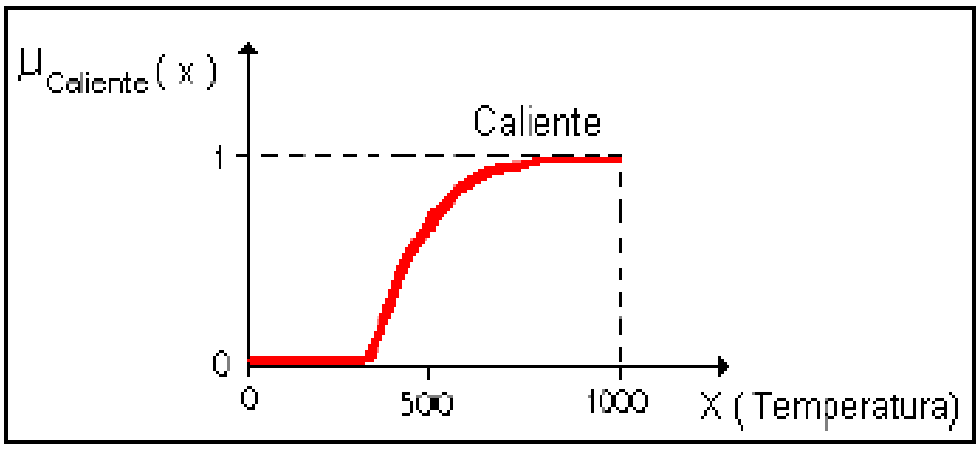


Figura 2.17 Representación gráfica de la función de pertenencia del conjunto difuso "caliente".

La función de pertenencia se establece de una manera arbitraria, lo cual es uno de los aspectos más flexibles de los Conjuntos Difusos. Por ejemplo, se puede convenir que la temperatura de 900 ºC pertenece al conjunto con grado 1, la de 500 ºC con grado 0.4 y la de 200 ºC con grado 0. Luego, cuanto mayor sea el valor de una temperatura, mayor es su grado de pertenencia al conjunto "caliente".

Estos valores entre 0 y 1 son llamados grados de pertenencia. El grado de pertenencia de un elemento a un conjunto va a venir determinado por su función de pertenencia.

Así, si se habla del conjunto "Joven" dentro del universo de las edades, se podría decir que una persona de 10 años pertenece a dicho conjunto con grado 1 (pertenece completamente), una de 35 pertenece con algún grado (por ejemplo, 0.5) y una persona de 70 años pertenecería con grado 0 (no pertenece). Esto es apreciable en su función de pertenencia (Figura 2.18).

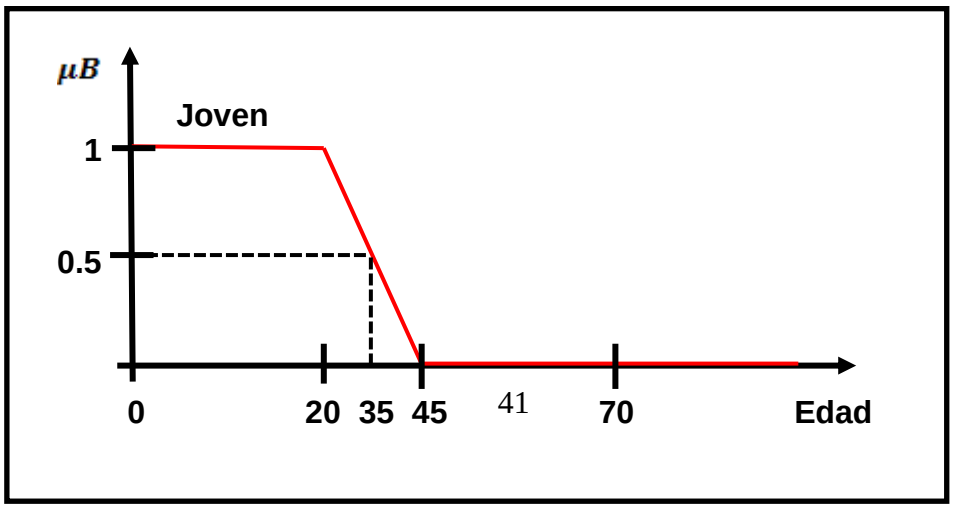


Figura 2.18 Representación gráfica de la función de pertenencia del conjunto difuso "Joven".

Como se puede apreciar, en los dos ejemplos anteriores, la función de pertenencia del conjunto difuso toma todos los valores reales comprendidos en el intervalo [0,1]. Por tanto, la función que a cada elemento asigna un grado de pertenencia a un cierto conjunto, se denomina función de pertenencia del conjunto difuso.

Tómese nuevamente el universo de la edad. Se había visto que el Conjunto Difuso "Joven" representa el grado de pertenencia respecto al parámetro juventud que tendrían los individuos de cada edad. Es decir, el conjunto expresa la posibilidad de que un individuo sea considerado joven. De igual manera, se puede definir un conjunto “Maduro” y uno “Viejo” (Figura 2.19). Los Conjuntos Difusos de la Figura 2.18 se superponen, por lo que un individuo podría tener distintos grados de pertenencia en dos conjuntos al mismo tiempo: "Joven" y "Maduro".

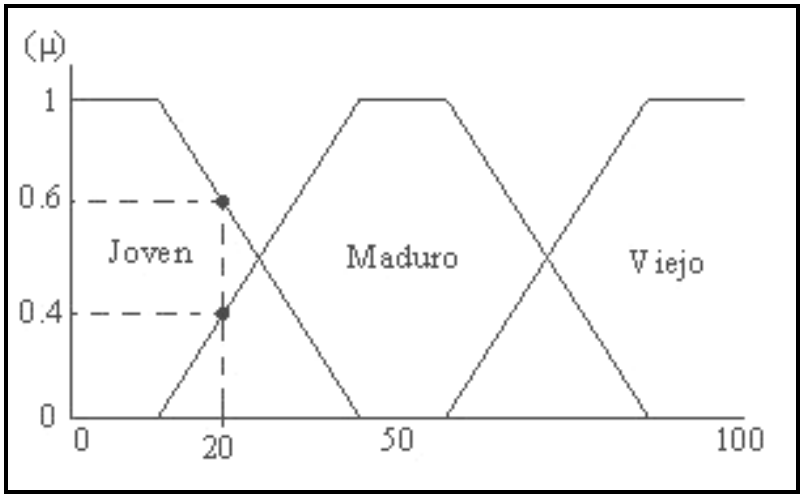


Figura 2.19 Ejemplo de Conjuntos Difusos en el universo de la edad.

Tómese un individuo x cuya edad sea de 20 años. Como se puede observar en la figura anterior, pertenece al Conjunto Difuso "Joven" y al Conjunto Difuso "Maduro". Se puede observar que posee un grado de pertenencia μ(x) de 0.6 para el Conjunto Difuso “Joven” y un grado de 0.4 para el Conjunto Difuso “Maduro”; también posee un grado de 0 para el conjunto “Viejo”.

De este ejemplo se puede deducir que un elemento puede pertenecer a varios Conjuntos Difusos a la vez aunque con distinto grado. Así, el individuo x tiene un grado de pertenencia mayor al conjunto "Joven " que al conjunto "Maduro"(0.6 > 0.4), pero no se puede decir, tratándose de Conjuntos Difusos, que x es joven o que x es maduro de manera rotunda.

Funciones de Pertenencia Típicas.

Función Singlenton:

Sea a un punto del universo, la Función Singlenton (solitaria) es aquella que toma valor 1 solo en a y 0 en cualquier otro punto.

$$\mu(x) = \begin{Bmatrix} 1 & si\ x = a \\ 0 & si\ x \neq a \end{Bmatrix}$$

Función Triangular:

Definido por sus límites (inferior a y superior b), y el valor modal m, tal que a<m<b.

$$\mu(x) = \begin{Bmatrix} 0 & si\, x \leq a \\ (x-a)/(m-a) & si\, x \in (a,m] \\ (b-x)/(b-m) & si\, x \in (m,b] \\ 0 & si\, x \geq b \end{Bmatrix}$$

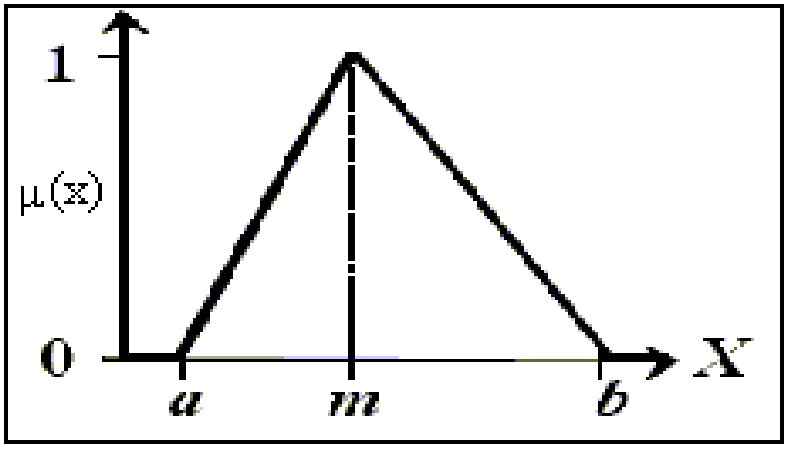


Figura 2.20 Función Triangular.

Función G (gamma):

Definida por su límite inferior a y el valor k>0.

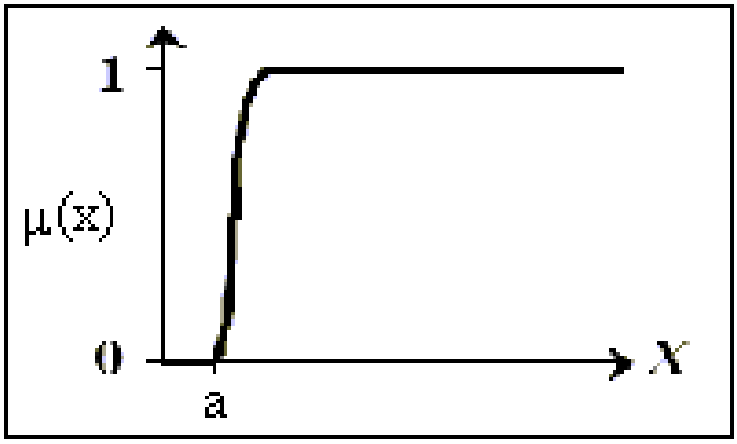


Figura 2.21 Función G.

$$\mu(x) = \begin{cases} 0 & si\ x \le a \\ 1 - e^{-k(x-a)^2} & si\ x > a \end{cases}$$

Esta función se caracteriza por un rápido crecimiento a partir de a. Cuanto mayor es el valor de k, el crecimiento es más rápido aún. Nunca toman el valor 1, aunque tienen una asíntota horizontal en 1.

Se aproximan linealmente por:

$$\mu(x) = \begin{cases} 0 & si\ x \le a \\ (x-a)/(m-a) & si\ x \in (a,m) \\ 1 & si\ x \ge m \end{cases}$$

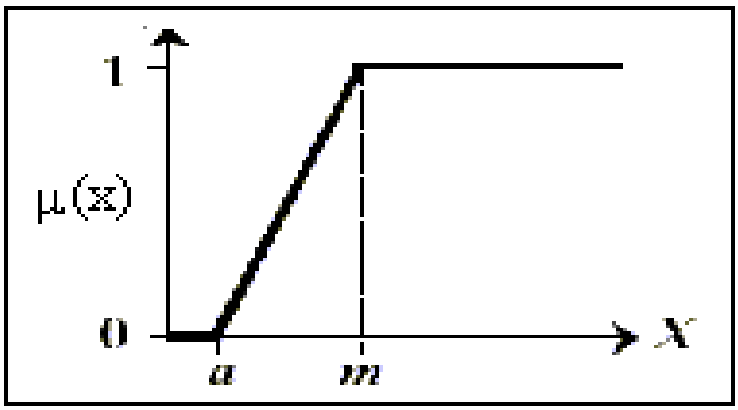


Figura 2.22 Aproximación lineal de la Función G.

Función S:

Definida por sus límites inferior a y superior b, y el valor m, o punto de inflexión tal que a<m<b. Un valor típico es: m = (a+b) / 2.

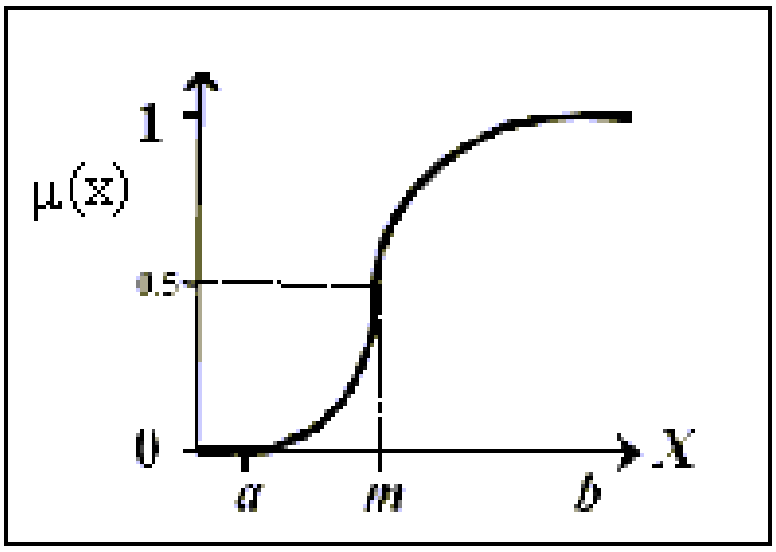


Figura 2.23 Función S.

El crecimiento es más lento cuanto mayor sea la distancia a-b.

$$\mu(x) = \begin{Bmatrix} 0 & si\, x \le a \\ 2\{(x-a)/(b-a)\}^2 & si\, x \in (a,m] \\ 1-2\{(x-b)/(b-a)\}^2 & si\, x \in (m,b] \\ 1 & si\, x \ge b \end{Bmatrix}$$

Función Gaussiana:

Definida por su valor medio m y el valor k>0. Es la típica campana de Gauss. Cuanto mayor es k, más estrecha es la campana.

$$\mu(x) = e^{-k(x-m)^2}$$

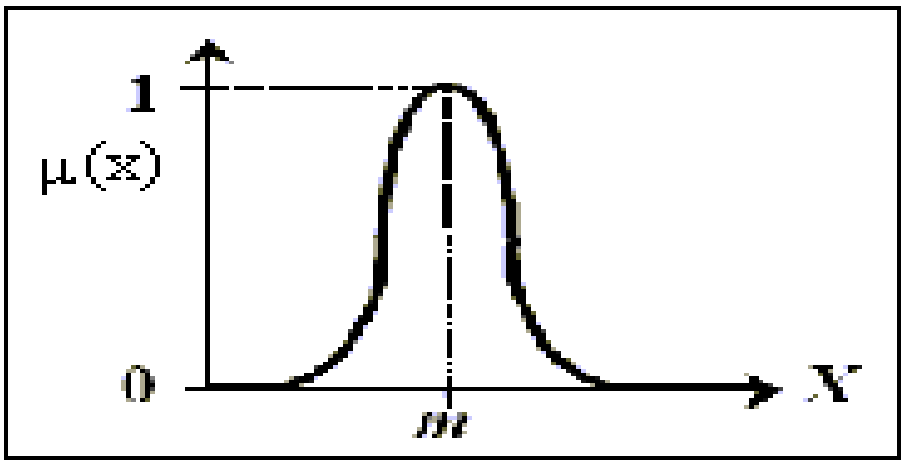


Figura 2.24 Función Gaussiana.

Función Trapezoidal:

Definida por sus límites inferior a y superior d, y los límites de su soporte, b y c, inferior y superior respectivamente.

$$\mu(x) = \begin{Bmatrix} 0 & si\, x \le a \quad o \quad x \ge d \\ (x-a)/(b-a) & si\, x \in (a,b] \\ 1 & si\, x \in (b,c] \\ (d-x)/(d-c) & si\, x \in (b,d) \end{Bmatrix}$$

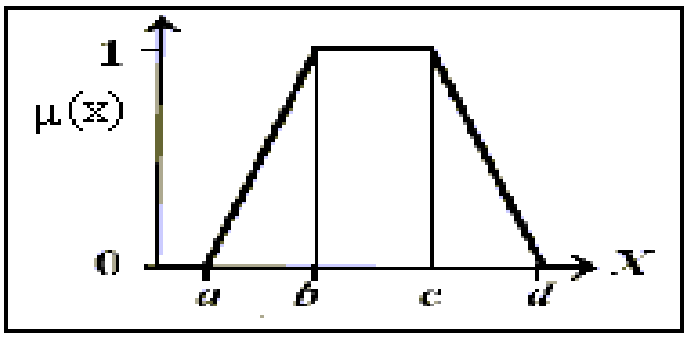


Figura 2.25 Función Trapezoidal.

La función Trapezoidal se adapta bastante bien a la definición de cualquier concepto, con la ventaja de su fácil definición, representación y simplicidad de cálculos.

Función Pseudo-Exponencial:

Definida por su valor medio m y el valor k>1. Cuanto mayor es el valor de k, el crecimiento es más rápido aún y la "campana" es más estrecha.

$$\mu(x) = \frac{1}{1 + k(x - m)^2}$$

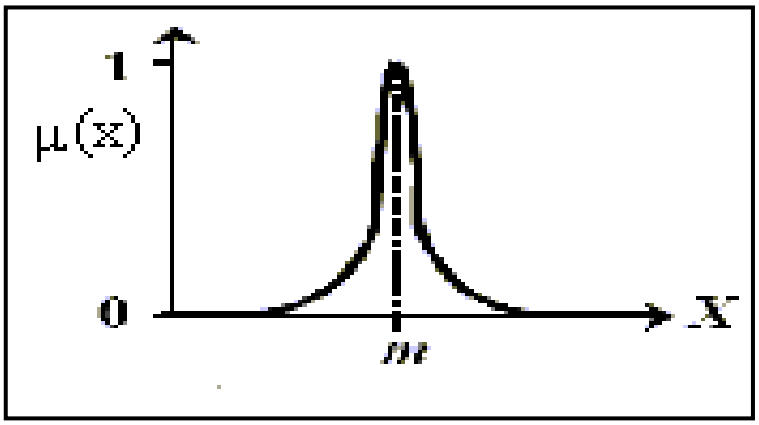


Figura 2.26 Función Pseudo-Exponencial.

Función Trapecio Extendido:

Definida por los cuatro valores de un trapecio [a, b, c, d], y una lista de puntos entre a y b, o entre c y d, con su valor de pertenencia asociado a cada uno de esos puntos.

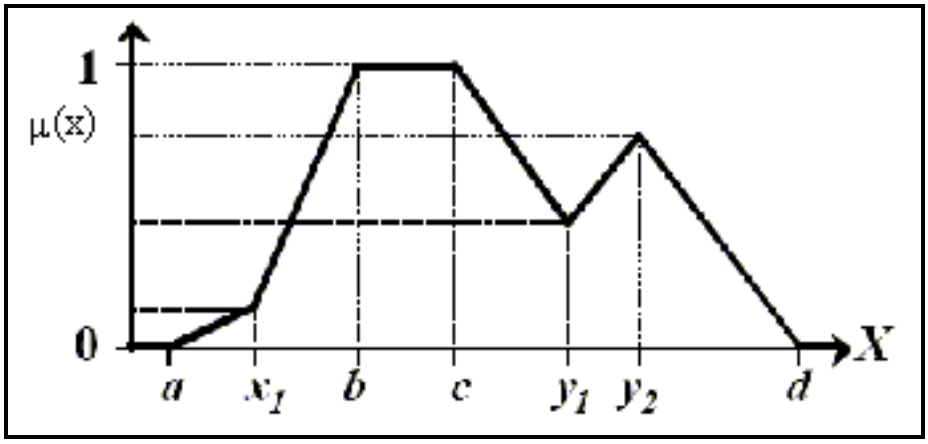


Figura 2.27 Función Trapecio Extendido.

En casos particulares, el Trapecio Extendido puede ser de gran utilidad. Éste permite gran expresividad aumentando su complejidad.

Aplicación en robótica

La Lógica Difusa ha demostrado ser una herramienta especialmente útil en el campo de la robótica, caracterizado por:

La imposibilidad de disponer de un modelo matemático fiable de un entorno real, cuando éste alcanza unos mínimos niveles de complejidad.

La incertidumbre e imprecisión de los datos proporcionados por los sensores.

La necesidad de operar en tiempo real.

La presencia de incertidumbre en el conocimiento que se tiene del entorno.

Existen distintos tipos o formas de incertidumbre [Saffiotti, 1997]. Así, si se dice que "el robot se encuentra en el almacén" se está proporcionando una información imprecisa, pues no se da una única posición del robot. Si la información que se proporciona es que "el robot se encuentra aproximadamente en el centro del almacén", esta información es vaga ya que la posición proporcionada no es exacta. Por último, la sentencia "el robot estaba ayer en la

posición (2, 3)" suministra una información no fiable, en tanto que puede que el robot ya no esté en esa posición. En los tres casos la información se puede calificar como incierta ya que no es posible conocer con exactitud la posición real actual del robot.

Cualquier intento para controlar un sistema dinámico necesita utilizar algún conocimiento o modelo del sistema a controlar. En el caso de la robótica el sistema está formado por el propio robot y el entorno en que éste opera. Aunque normalmente se puede obtener el modelo del robot, no ocurre lo mismo cuando se considera al robot situado en un entorno no estructurado. Los entornos están caracterizados por una fuerte presencia de incertidumbre debida, por ejemplo, a la existencia de personas que se desplazan, objetos que pueden cambiar de posición, nuevos obstáculos, etc.

Además, existen numerosos factores que pueden conducir a un sistema de robótica a un estado erróneo durante la ejecución de una secuencia de tareas: errores sensoriales, factores debidos al ambiente de trabajo, información imprecisa del proceso, información errónea, etc. En este sentido, la Lógica Difusa incorpora al sistema la capacidad para recuperarse de los posibles errores, presentando así a la vez robustez en la detección y recuperación de estos estados erróneos.

El tratamiento de la borrosidad permite representar de forma aproximada la geometría del problema, ordenar las distintas alternativas (subtareas) en función de la pertenencia a los estados previos, tratamiento de incertidumbre en las medidas de los sensores, etc.

Una de las aplicaciones más extendidas de las técnicas borrosas es el diseño de comportamientos. Los comportamientos son tareas como: evitar obstáculos fijos, seguir un contorno, evitar obstáculos móviles, cruzar puertas, seguir una trayectoria, empujar o cargar un objeto, etc. Estas son tareas de muy diferente complejidad. Los controladores borrosos incorporan conocimiento heurístico en forma de reglas del tipo si-entonces, y son una alternativa adecuada en el caso de que no se pueda obtener un modelo preciso del sistema a controlar.

Ventajas.

El control borroso ofrece varios beneficios a los controladores industriales:

Robustez frente a cambios en el sistema.

Tolerancia mayor a las señales ruidosas que otros métodos tradicionales de control.

Capacidad de manejar información que contiene gran incertidumbre.

No depende de ecuaciones matemáticas complejas o extensas.

Sencillez para desarrollar controladores para los distintos comportamientos (sin utilizar complejos modelos matemáticos), gracias al formato de las reglas.

Posibilidad de utilizar los mismos controladores sobre diferentes plataformas sin realizar muchos cambios, debido a su naturaleza cualitativa.

Posibilidad de evaluar mayor cantidad de variables, entre otras, variables lingüísticas, no numéricas, simulando el conocimiento humano.

Relaciona entradas y salidas, sin tener que entender todas las variables, permitiendo que el sistema pueda ser más confiable y estable que uno con un sistema de control convencional.

Capacidad de simplificar la asignación de soluciones previas a problemas sin resolver.

Posibilidad de obtener prototipos, rápidamente, ya que no requiere conocer todas las variables acerca del sistema antes de empezar a trabajar, siendo su desarrollo más económico que el de sistemas convencionales, porque son más fáciles de designar.

Simplifica también la adquisición y representación del conocimiento y unas pocas reglas abarcan gran cantidad de complejidades.

Es importante señalar, que los sistemas basados en la Lógica Difusa requieren mayor simulación y una excelente depuración y prueba antes de pasar a ser operacionales.

Desventajas

No hay actualmente un análisis matemático riguroso que garantice que el uso de un sistema experto difuso, para controlar un sistema, dé cómo resultado un sistema estable.

Es difícil llegar a una función de membresía y a una regla confiable sin la participación de un experto humano.

Dificultad de interpretación de valores difusos. Múltiples definiciones de operadores y reglas de inferencia difusas.

2.2.4 Controlador Difuso

La arquitectura típica de un controlador difuso se muestra en la siguiente figura 2.28, la cual comprende cuatro componentes principales: un fusificador, una base de reglas difusas, una máquina de inferencia y un defusificador. Si la salida del defusificador no es una acción de control para una planta, entonces el sistema es un sistema de decisión lógica. El fusificador tiene el efecto de transformar un dato "duro" en valores lingüísticos adecuados. La base de reglas difusas almacena el conocimiento empírico de la operación del proceso del dominio del experto. La máquina de inferencia es el núcleo de un controlador difuso, y tiene la capacidad de simular la toma de decisiones de un experto humano realizando razonamiento aproximado para alcanzar la estrategia de control deseada. El defusificador es utilizado para proporcionar una decisión no difusa o acción de control a partir de la acción de control difusa obtenida por la máquina de inferencia.

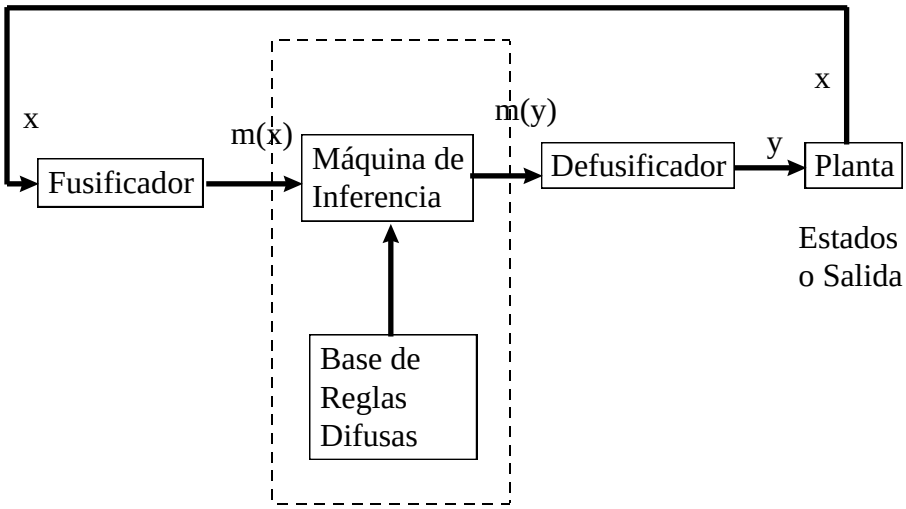


Figura 2.28: Estructura de un controlador difuso.

El propósito de los controladores difusos es calcular valores de variables de control (o acción) a partir de la observación o medidas de las variables de estado del proceso controlado tal que se alcance una cierta funcionalidad deseada del sistema. Así, es esencial escoger apropiadamente las variables de estado y de control del proceso para la caracterización de la operación de un Sistema de Control de Lógica Difusa (sistema de control de lógica difusa) y tiene efectos substanciales sobre la efectividad de un controlador difuso. La experiencia experta y el conocimiento de ingeniería juegan una regla importante durante este proceso de selección de variables de estado y de control. Típicamente las variables de entrada en un controlador difuso son el estado, el error del estado, la derivada del error del estado, la integral del error del estado, etc. Sigue la definición de las variables lingüísticas, el vector de entrada x el cual incluye las variables lingüísticas de estado x_i y el vector de estado de salida y el cual incluye las variables lingüísticas de estado de la salida (o control) y_i los cuales pueden ser definidos como:

$$x = \left\{ (x_i, U_i, \left\{ T_{x_i}^1, T_{x_i}^2, \ldots, T_{x_i}^{k_i} \right\}, \left\{ \mu_{x_i}^1, \mu_{x_i}^2, \ldots, \mu_{x_i}^{k_i} \right\}) \Big|_{i=1,\ldots,n} \right\}$$
$$y = \left\{ (y_i, V_i, \left\{ T_{y_i}^1, T_{y_i}^2, \ldots, T_{y_i}^{l_i} \right\}, \left\{ \mu_{y_i}^1, \mu_{y_i}^2, \ldots, \mu_{y_i}^{l_i} \right\}) \Big|_{i=1,\ldots,m} \right\}$$

donde las variables lingüísticas de entrada x_i forman un espacio de entrada difuso $U=U_1\times U_2\times,\ldots,\times U_n$ y las variables lingüísticas de salida y_i forman un espacio de salida difuso $V=V_1\times V_2\times,\ldots,\times V_m$, En las ecuaciones anteriores $T(x_i)$ es el conjunto de términos de x_i , esto es el conjunto de nombres de valores lingüísticos de x_i con cada valor $T_{x_i}^{k_i}$ siendo el número difuso con función de membresía $\mu_{x_i}^{k_i}$ definidas sobre U_i . De modo que $\mu(x_i)$ es una regla semántica para asociar cada valor con su significado.

Fusificador

Un fusificador realiza la función de fusificación lo cual es una evaluación subjetiva para transformar datos medidos a la evaluación de valores subjetivos. Dado que, estos pueden ser definidos como un mapeo desde un espacio de entrada observado a etiquetas de conjuntos difusos en un universo de discurso de entrada especificado. Debido a que la manipulación de datos en un controlador de lógica difusa está basado sobre la teoría de conjuntos difusos, la fusificación es necesaria y deseable en un estado inicial. En las aplicaciones de control difuso, los datos observados son usualmente duros (aunque ellos pueden tener ruido añadido). Un enfoque natural y simple de fusificación es convertir un valor duro x_0 en un único A dentro del universo de discurso especificado. Esto es, la función de membresía de A, μ_A (x), es igual a 1 en el punto x_0 y cero en otros lugares. En este caso, para un valor específico x_i (t) en el tiempo t, es mapeado al conjunto difuso $T_{x_i}^1$ con grado $\mu_{x_i}^1(x_i(t))$ y al conjunto difuso $T_{x_i}^2$ con grado $\mu_{x_i}^2(x_i(t))$, etc. Este enfoque es ampliamente usado en aplicaciones de control con lógica difusa debido a que simplifica grandemente el proceso de razonamiento difuso. En un caso más complejo donde los datos observados están corrompidos por ruido aleatorio, un fusificador debe convertir los datos probabilísticos en números difusos, estos es, datos difusos (posibilidad). Por esto, Dubois y Prade definieron una transformación biyectiva la cual transforma una medida de probabilidad en una medida de posibilidad usando el concepto de

grados de necesidad. En sistemas de gran escala, algunas observaciones se relacionan directamente al funcionamiento de tales sistemas, otras son medibles únicamente en un sentido estadístico, y algunos, referidos como "híbridos" requieren de caracterizaciones tanto probabilísticas como de posibilidades.

Base de Reglas Difusas

Las reglas de control difusas son caracterizadas por una colección de reglas SI - ENTONCES difusas en las cuales los antecedentes y los consecuentes involucran variables lingüísticas. Esta colección de reglas de control difusas (o estatutos de control difusos) caracterizan la simple relación de entrada - salida del sistema. La forma general de las reglas de control difusas en el caso de sistemas de múltiples entradas - una sola salida es:

R^i : SI x es A_i ,…,AND y es B_i, ENTONCES z = C_i, i=1,2,…n,

donde x,…,y, y z son variables lingüísticas que representan las variables de estado del proceso y las variables de control, respectivamente, y A_i ,…, B_i y C_i son los valores lingüísticos de las variables lingüísticas x,…,y, y z en el universo de discurso U,…,V y W, respectivamente. Una variante de este tipo es que el consecuente está representado como una función de las variables de estado del proceso x,…,y,, esto es:

R^i : SI x es A_i ,…,AND y es B_i, ENTONCES z = f(x,…,y), donde f(x,…,y) es una función de las variables de estado x,…,y. Las reglas de control difuso en las ecuaciones anteriores evalúan el estado del proceso en el tiempo t y calculan y deciden las acciones de control como una función de las variables de estado (x,…,y).

Máquina de inferencia

Este es el núcleo del controlador difuso en la modelación de toma de decisiones humanas dentro del marco conceptual de la lógica difusa y el razonamiento aproximado.

Desdifusión

La desdifusión es un mapeo desde el espacio de las acciones de control difusas definidas sobre un universo de discurso de salida en un espacio de acciones no difusas (duras). Este proceso es necesario debido a que en muchas aplicaciones prácticas se requieren acciones de control duras para que actúe el control.

Se requiere una estrategia de defusificación para producir una acción de control que mejor represente la distribución de posibilidad de una acción de control difusa inferida. Desafortunadamente no hay un procedimiento sistemático para escoger una estrategia de defusificación. Dos métodos de defusificación comúnmente usados comúnmente usados son el método del centro del área (COA) y el método de la media del máximo (MOM).

La estrategia ampliamente usada del COA genera el centro de gravedad de la distribución de posibilidades de una acción de control. En el caso de un universo discreto, este método es:

$$z_{COA}^{*} = \frac{\sum_{j=1}^{n} \mu_C\left(z_j\right) z_j}{\sum_{j=1}^{n} \mu_C\left(z_j\right)}$$

donde n es el número de niveles de cuantificación de la salida, z_j es la cantidad de salida de control al nivel de cuantificación j, y $\mu_C(z_j)$ representa su valor de membresía en el conjunto difuso C. Si el universo de discurso es continuo, luego la estrategia del COA genera una acción de control de salida de:

$$z_{COA}^{*} = \frac{\int_z \mu_C(z) z dz}{\int_z \mu_C(z) dz}$$

La estrategia del MOM genera una acción de control que representa el valor medio de todas las acciones de control local cuyas funciones de membresía alcanzan el máximo, En el caso de un universo discreto, la acción de control puede expresarse como:

$$z^*_{MOM} = \sum_{j=1}^{m} \frac{z_j}{m}$$

donde z_j es el valor de soporte en el cual la función de membresía alcanza el valor máximo $\mu_C(z_j)$ y m es el número de tales valores de soporte.

De estas dos estrategias comúnmente usadas, la estrategia del COA ha mostrado proporcionar resultados superiores. No obstante el la estrategia del MOM proporciona una ejecución más rápida, y la estrategia del COA proporciona una mejor aproximación (error cuadrático medio más bajo).

Construcción de sistemas de control difusos.

A partir de las discusiones previas, podemos ver que los elementos de diseño principales de un controlador difuso incluyen:

Los dos primeros principios de diseño indican que, en el diseño de un controlador difuso, uno debe identificar las variables de estado del proceso principales y las variables de control y determinar un conjunto de términos que tengan el nivel correcto de granularidad para describir los valores de cada variable lingüística. Esto tiene un efecto esencial sobre que tan fino puede obtenerse el control. Además pueden usarse diferentes tipos de funciones de membresía.

Para los principios de diseño cuarto y quinto no existen metodologías sistemáticas para realizar el diseño de la máquina de inferencia o para seleccionar un operador de defusificación. Muchos utilizan estudios y resultados empíricos para realizar la selección.

El tercer principio de diseño en la determinación de las reglas de control difuso depende fuertemente sobre la naturaleza de la planta a ser controlada. En general, hay cuatro métodos para la derivación de las reglas de control difusas, y estos métodos no son mutuamente exclusivas. Puede ser necesaria una combinación de ellos para construir un método efectivo para la derivación de las reglas de control difuso.

Experiencia de un Experto y Conocimiento de Ingeniería del Control. Las reglas de control difusas se diseñan refiriéndose al conocimiento de ingenieros y/o operadores humanos.

Modelación de las Acciones de Control de un Operador De esta manera es más fácil obtener el conjunto de reglas preguntándole al operador qué tipo de información utiliza en sus acciones de control u observando estas acciones.

Basándose sobre un modelo difuso o análisis del funcionamiento de un proceso controlado. Si se tiene un modelo previo del proceso o si conocemos algunas propiedades útiles del mismo se puede optimizar diseñando un conjunto de reglas de control difusas.

Basado sobre el aprendizaje (o Auto organizativos) Son sistemas actualmente en investigación que permitirán el auto aprendizaje de modo que generarán sus propias reglas de acuerdo a la situación actual.

CAPÍTULO III : DISEÑO DEL SISTEMA

3.1. MODELAMIENTO DINÁMICO DEL ROBOT PUMA

Para realizar el control se ha escogido un robot PUMA de cuatro GDL rotacionales. Para todas las medidas y cálculos se usó el Sistema Internacional de Unidades.

El brazo robótico ha sido diseñado en base a la gran versatilidad del brazo humano, en un principio los diseños fueron rudimentarios pero con el tiempo se logró igualar, incluso, superar la complejidad de los movimientos. En la Figura 3.1 se compara el brazo robótico respecto al brazo humano.

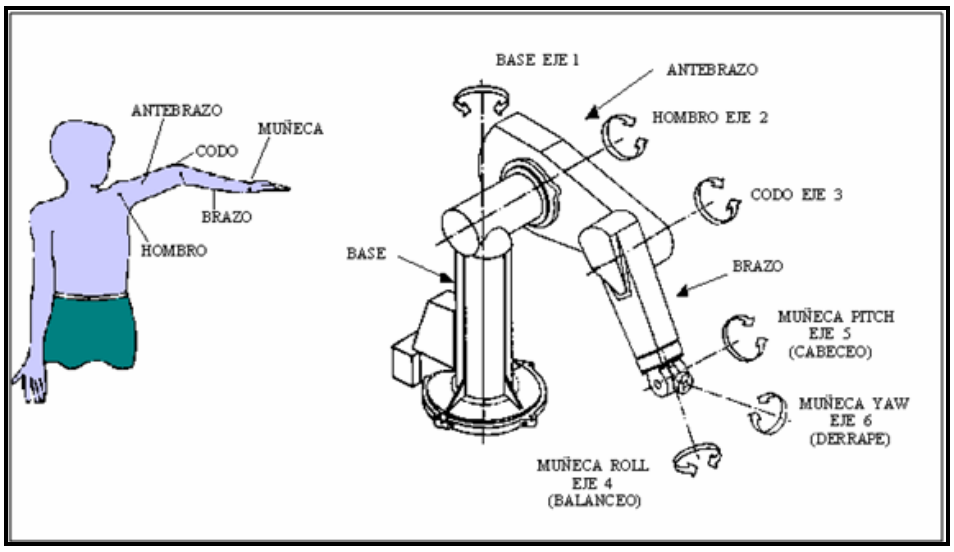


Figura 3.1: Comparación entre un Brazo Robótico y un Brazo Humano.

El brazo robótico es de gran importancia en la industria no solo por su eficacia al realizar trabajos, sino porque sustituye al hombre en labores de producción que no corresponden a un humano, es decir mejora la calidad de vida.

Para el diseño del robot se realizó un modelo utilizando Solidworks el cual puede observarse en la figura 3.2.

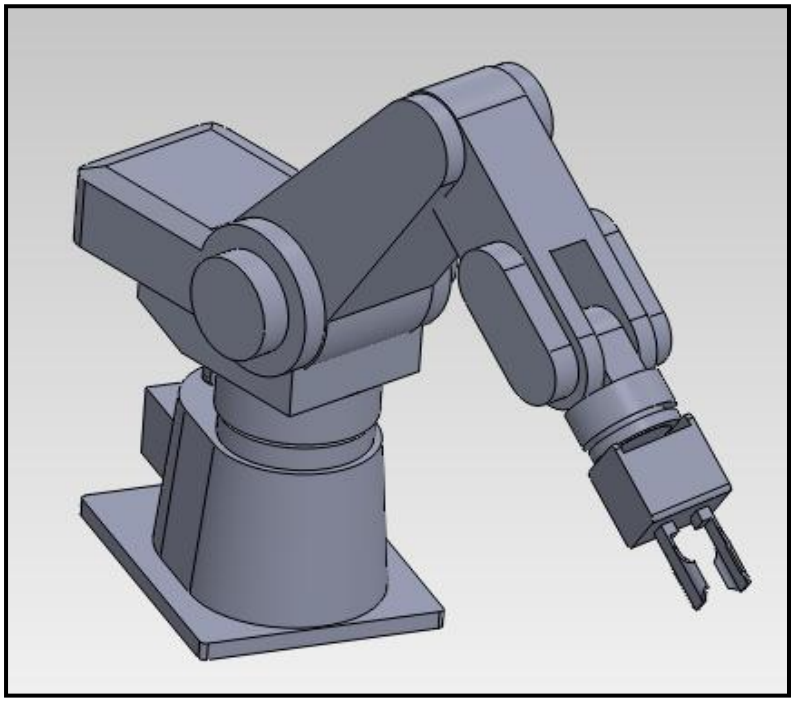


Figura 3.2: Diseño del Robot Manipulador PUMA en Solidworks.

El cálculo de la dinámica se realizó usando el método de Lagrange-Euler. La dinámica del robot que puede ser expresada de la siguiente forma:

$$\tau = H\ddot{q} + C + G$$

Donde H es la matriz de inercia, C la matriz de Coriolis y G la matriz de Gravedad.

Los parámetros físicos del robot como distancias centros de masa e inercias fueron escogidos según los datos obtenidos en el programa SolidWorks y replanteados para un manejo más sencillo.

También es posible utilizar distintos toolbox de Matlab como el Robotics Toolbox o métodos iterativos para su cálculo. Se realizó un modelo en Simulink para la simulación de la dinámica de nuestro robot.

Para aplicar el método de Lagrange-Euler se tomó en consideración los siguientes datos del robot:

t L2 L3

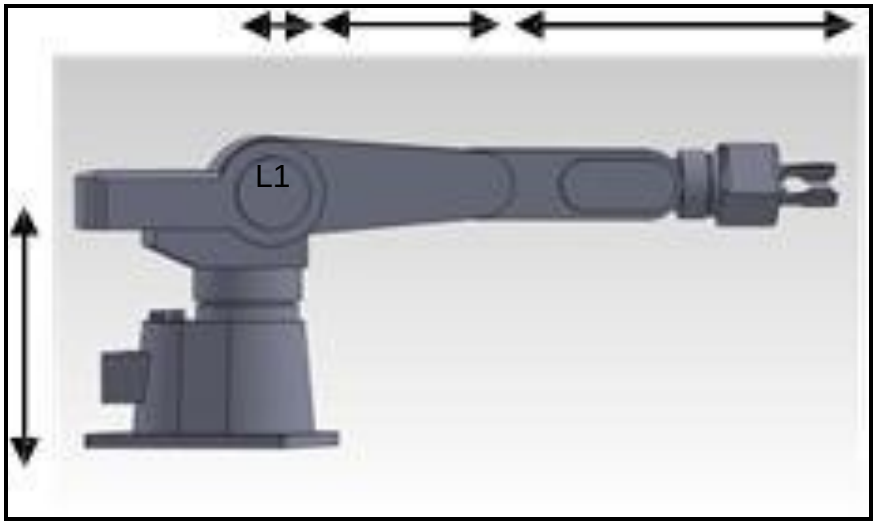


Figura 3.3: Vista de perfil del Robot Manipulador PUMA.

Dimensiones:

L1 = 0.300 m

L2 = 0.250 m

L3 = 0.380 m

t = 0.02 m

Masas:

m1 = 26.72 kg

m2 = 22.44 kg

m3 = 11.48 kg

m4 = 1.44 kg

El algoritmo de resolución del problema cinemático directo del robot PUMA es el siguiente:

Mover el manipulador a su posición cero.

Asignar un sistema de coordenadas a cada elemento del robot.

c) Describir las posiciones y orientaciones entre los elementos del robot. La representación de Denavit-Hartenberg de un elemento rígido depende de cuatro parámetros geométricos asociados con cada elemento:

θ : Ángulo entre el eje X_{i-1} y la normal común H_iQ_i, medido alrededor de Z_{i-1} en sentido positivo.

d_i : Distancia del origen O_{i-1} al punto H_i.

a_i : Longitud de la normal común H_iO_i.

α: Ángulo entre los ejes Z_{i-1} y Z_i, medido alrededor de X_i en sentido positivo.

A continuación se presentan los parámetros Denavit-Hartenberg del robot PUMA:

Articulación	θ	d	a	α
1	q1	L1	t	pi/2
2	q2	0	L2	0
3	q3 + pi/2	0	0	pi/2
4	q4	L3	0	0

Tabla 3.1: Tabla de Parámetros de Denavit – Hartenberg (DH).

3.2. ALGORITMO DE CÁLCULO DEL TENSOR DE INERCIA DEL ROBOT H(q):

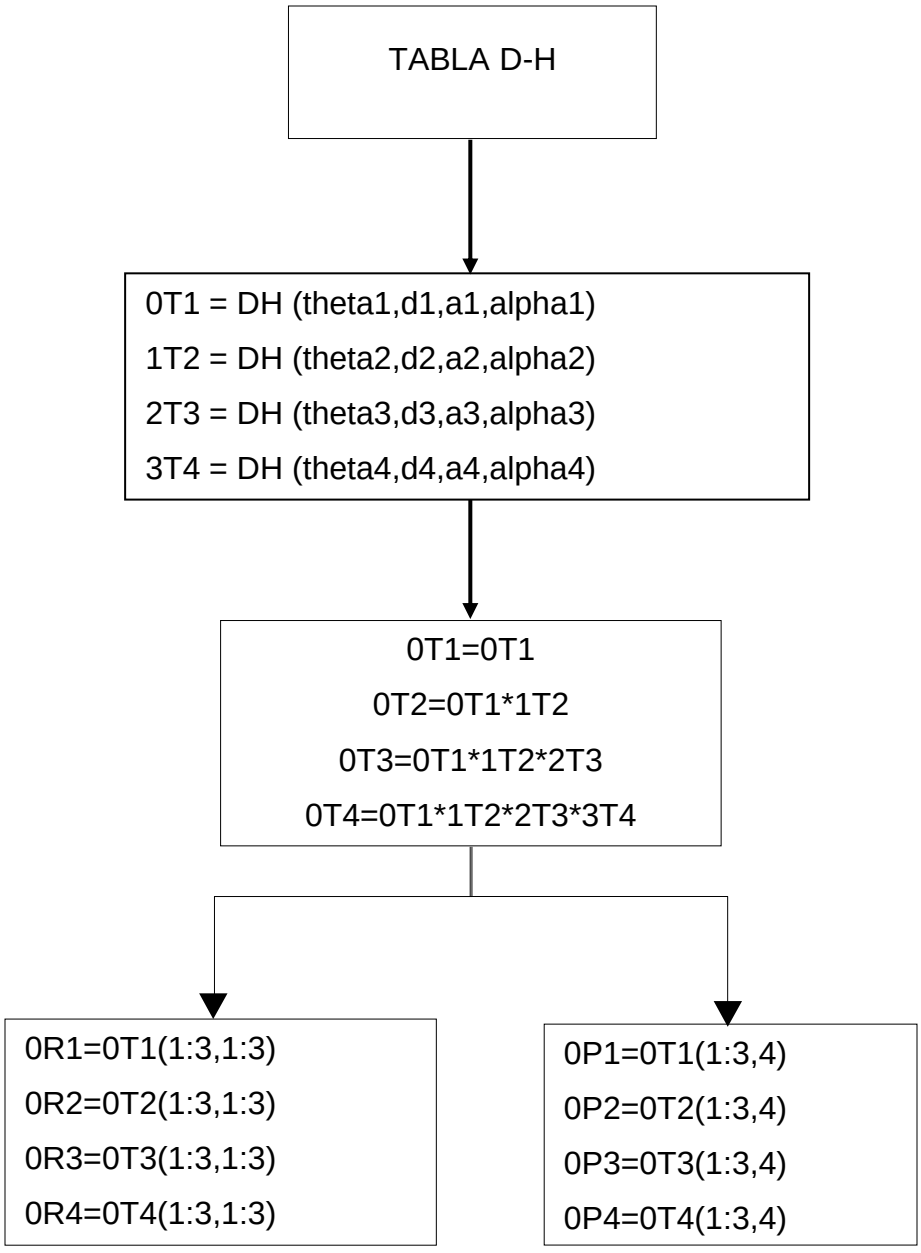
Paso 1: Matriz de Transformación.

$$[{}_0T^k_{(q)}] = \begin{bmatrix} [{}_0R^k_{(q)}] & [{}_0P^k_{(q)}] \\ 0 & 1 \end{bmatrix}_{4x4}$$

Paso 2: Posición de los centros de masa de cada eslabón con referencia $Xk_kY_kZ_k$.

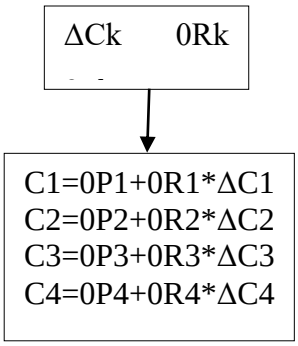
$$\Delta\overline{C}_1 = [-0.0074 \quad -0.042 \quad 0]^t m$$
$$\Delta\overline{C}_2 = [-0.0174 \quad 0.002 \quad 0]^t m$$
$$\Delta\overline{C}_3 = [0 \quad 0 \quad 0.099]^t m$$
$$\Delta\overline{C}_4 = [0 \quad 0 \quad -0.098]^t m$$

Se presenta el algoritmo para la obtención de la tabla de Denavit-Hartenberg (D-H):



Paso 3: Posición de los centros de masa de cada eslabón respecto al referencial $X_oY_oZ_o$.

$$\overline{C_{k\,(q)}} = [{}_0P^k_{(q)}] + [{}_0R^k_{(q)}] * \Delta\overline{C}_k$$



Paso 4: Matrices de Inercia de los centros de masas de cada eslabón con respecto al referencial $X_kY_kZ_k$.

$$\overline{I_1} = \begin{bmatrix} 0.13887 & -0.04079 & 0 \\ -0.04079 & 0.20758 & 0 \\ 0 & 0 & 0.21945 \end{bmatrix} kg.m^2$$

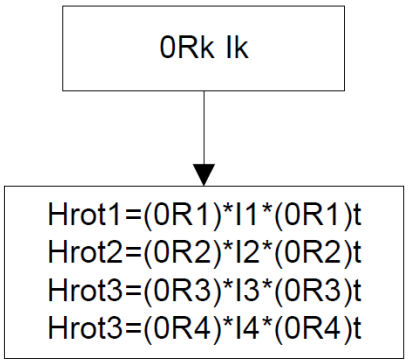
$$\overline{I_2} = \begin{bmatrix} 0.05357 & 0.00060 & 0 \\ 0.00060 & 0.19866 & 0 \\ 0 & 0 & 0.19361 \end{bmatrix} kg.m^2$$

$$\overline{I_3} = \begin{bmatrix} 0.08250 & 0 & 0 \\ 0 & 0.08182 & 0 \\ 0 & 0 & 0.01214 \end{bmatrix} kg.m^2$$

$$\overline{I_4} = \begin{bmatrix} 0.00205 & 0 & 0 \\ 0 & 0.00197 & 0 \\ 0 & 0 & 0.00146 \end{bmatrix} kg.m^2$$

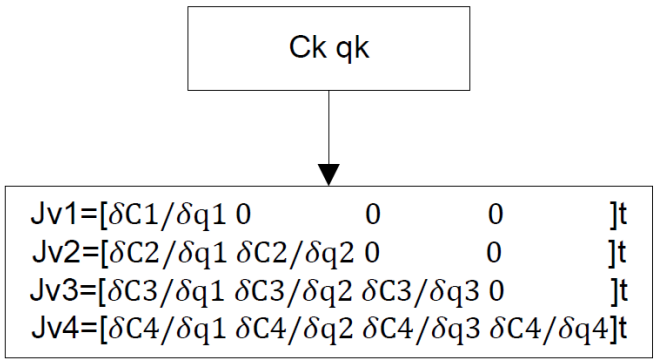
Paso 5: Tensor Inercial Rotacional de cada eslabón.

$$\overline{H_{k(q)}} = [{}_0R^k_{(q)}] * \overline{I}_k * [{}_0R^k_{(q)}]^t$$



Paso 6: Jacobiano de velocidades lineales del centro de masa de cada eslabón.

$$J_{v\,(q)}^{k} = \left[\frac{\partial \overline{C_{k}}_{(q)}}{\partial q_1} \quad \frac{\partial \overline{C_{k}}_{(q)}}{\partial q_2} \quad \frac{\partial \overline{C_{k}}_{(q)}}{\partial q_3} \quad \frac{\partial \overline{C_{k}}_{(q)}}{\partial q_4}\right]$$



Paso 7: Jacobiano de velocidades angulares de cada eslabón.

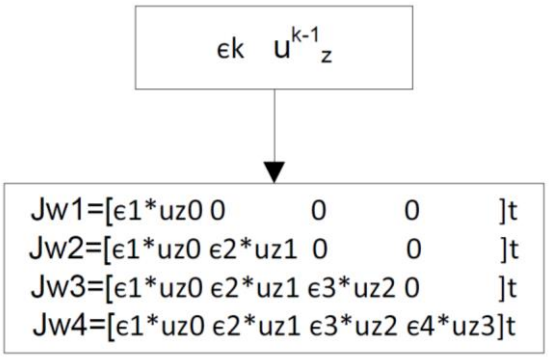
$$J_{w\,(q)}^{k} = [\varepsilon_1 * u_z^0 \quad \ldots \quad \varepsilon_k * u_z^{k-1} \quad 0 \quad \ldots \quad 0]$$

Donde:

ϵ_k = 1: Articulación rotacional

0: Articulación traslacional

$u^{k-1}{}_z$: Vector unitario del eje z del referencial X_{k-1} $Yk-1$ Z_{k-1} expresado en coordenadas $X_oY_oZ_o$ (3ra columna de $[oR^{k-1}{}_{(q)}]$).

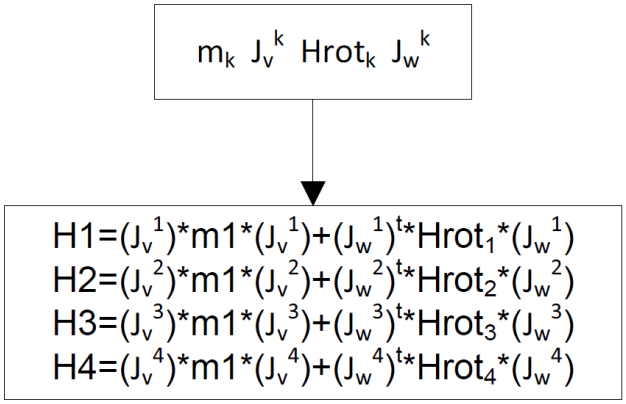


Observación: Jacobiano total de cada eslabón:

$$J_{(q)}^{k} = \begin{bmatrix} J_{v\,(q)}^{k} \\ J_{w\,(q)}^{k} \end{bmatrix}$$

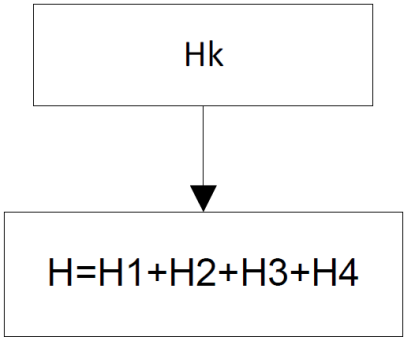
Paso 8: Tensor de Inercial Total de cada eslabón.

$$H_{k\,(q)} = [J_v^k]^t * m_k * [J_v^k] + [J_w^k]^t * \overline{H}_k * [J_w^k]$$



Paso 9: Tensor Inercial Total del Robot.

$$H_{(q)} = \sum_{k=1}^{4} H_{k\,(q)}$$



3.3. ALGORITMO DE CÁLCULO DEL VECTOR DE CORIOLIS C(q,q'):

Paso 1: Derivamos la matriz H(q).

$$\frac{\partial H}{\partial q_k}$$

$$= \begin{pmatrix} \partial H_{11}/\partial q_k & \partial H_{12}/\partial q_k & \cdots & \partial H_{1n}/\partial q_k \\ \partial H_{21}/\partial q_k & \partial H_{22}/\partial q_k & \cdots & \partial H_{2n}/\partial q_k \\ \vdots & \vdots & \ddots & \vdots \\ \partial H_{n1}/\partial q_k & \partial H_{n2}/\partial q_k & \cdots & \partial H_{nn}/\partial q_k \end{pmatrix}$$

Paso 2: Vector de velocidad articular.

$$\dot{q} = \begin{pmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \\ \dot{q}_4 \end{pmatrix}$$

Paso 3: Vector de Coriolis.

$$C(q,q') = \frac{\partial H}{\partial q_1} \times \dot{q} \times \dot{q_1} + \frac{\partial H}{\partial q_2} \times \dot{q} \times \dot{q_2} +$$

$$\cdots + \frac{\partial H}{\partial q_4} \times \dot{q} \times \dot{q_4} - \frac{1}{2} \begin{pmatrix} \dot{q}^T \frac{\partial H}{\partial q_1} \dot{q} \\ \dot{q}^T \frac{\partial H}{\partial q_2} \dot{q} \\ \dot{q}^T \frac{\partial H}{\partial q_3} \dot{q} \\ \dot{q}^T \frac{\partial H}{\partial q_4} \dot{q} \end{pmatrix}$$

3.4. ALGORITMO DE CÁLCULO DEL VECTOR DE FUERZA GENERADO POR LA GRAVEDAD h(q):

Paso 1: Energía Potencial.

Donde:

$$E_p = -\sum_{k=1}^{n} m_k g^T \overline{C_k}\,(q)$$

m_k = masa del eslabón k

g^T= vector de gravedad

$C_k(q)$ = vector de centro de masa de cada elemento

Paso 2: Derivamos la energía potencial.

$$h(q) = \frac{\partial E_p}{\partial q}$$

h(q) = vector de dimensión n x 1

3.5. MODELO DINÁMICO DEL ROBOT EN SIMULINK

En la Figura 3.4 se presenta el desarrollo del Simmechanics de cada uno de los elementos del robot PUMA de cuatro grados de libertad. En este desarrollo se consideró el hecho que el robot presenta varias no linealidades como fricción entre las articulaciones.

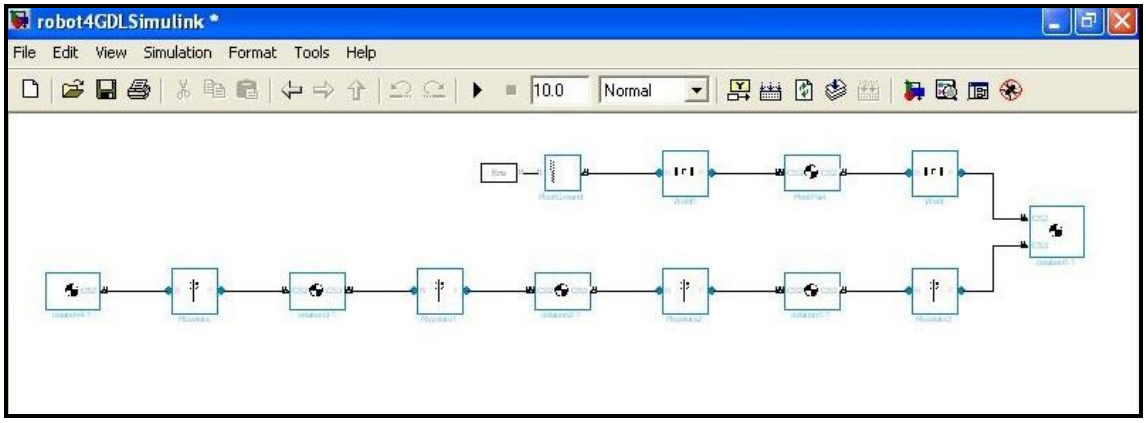


Figura 3.4: Desarrollo del Simmechanics del robot PUMA.

En la figura 3.5 se observa el brazo robótico diseñado en Solidworks. Este diseño interactúa con el Simulink a través de la interface Simmechanics, con la finalidad de simular el movimiento del robot PUMA.

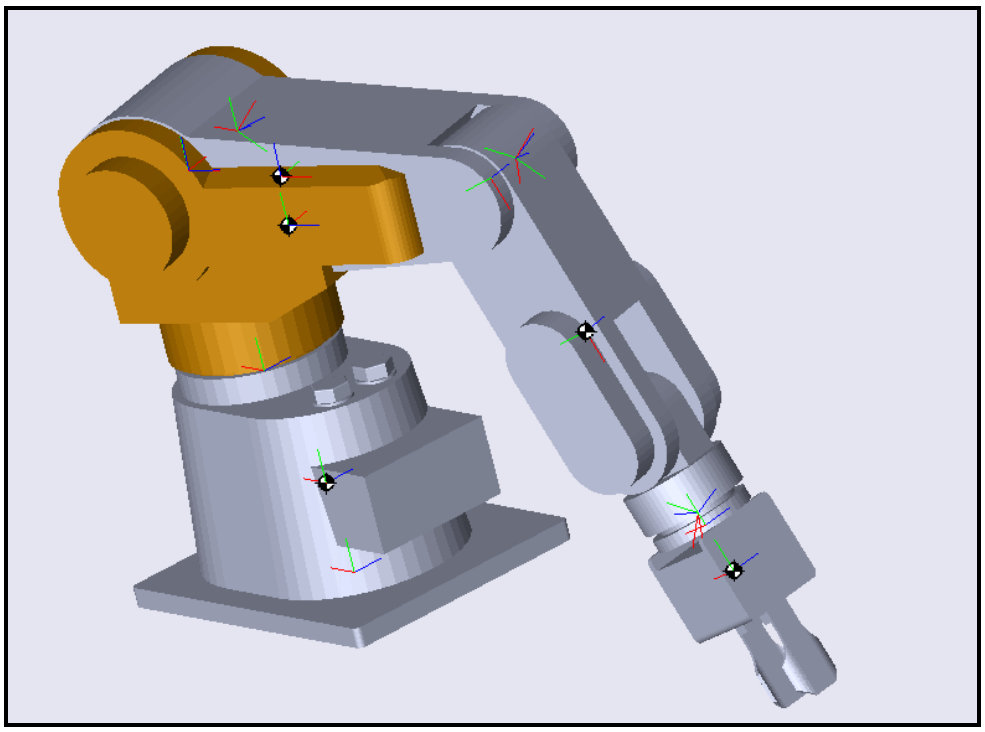


Figura 3.5: Robot Manipulador Simmechanics.

Entre la figura 3.6 hasta la figura 3.18 se presenta la configuración en Simulink de cada uno de los elementos que conforman el robot PUMA de cuatro grados de libertad.

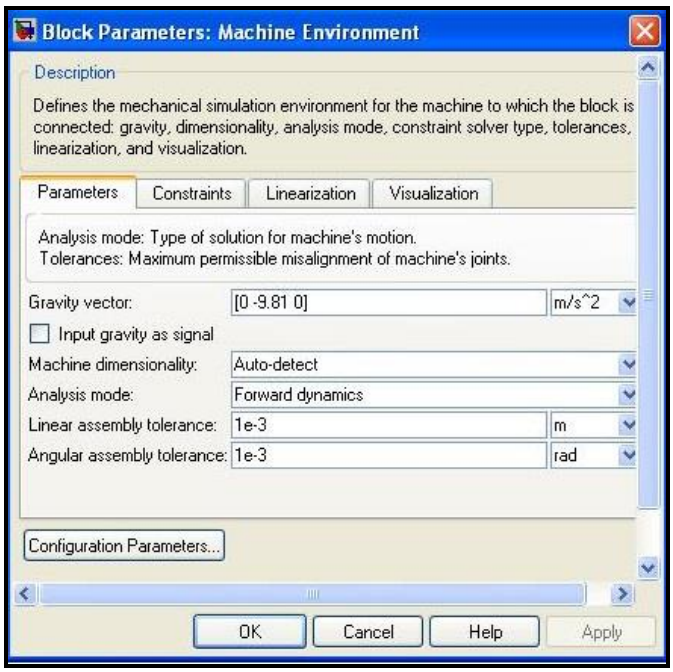


Figura 3.6: Configuración de los parámetros de la simulación mecánica del robot.

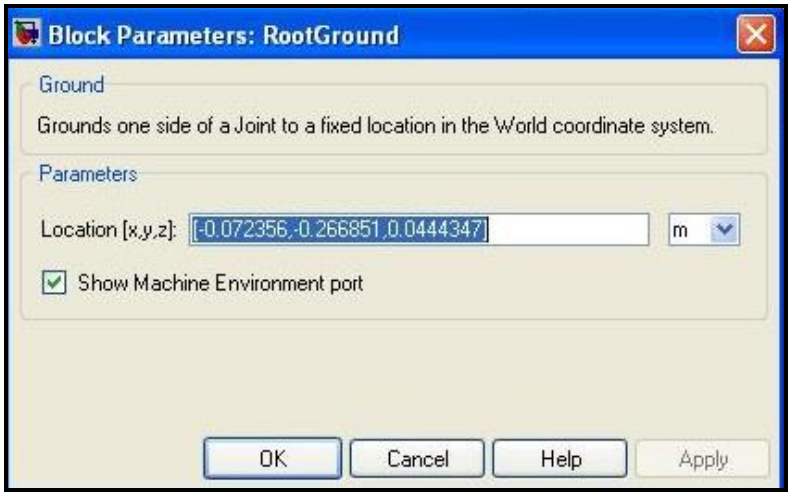


Figura 3.7: Configuración del sistema de coordenadas ubicado en la base del robot.

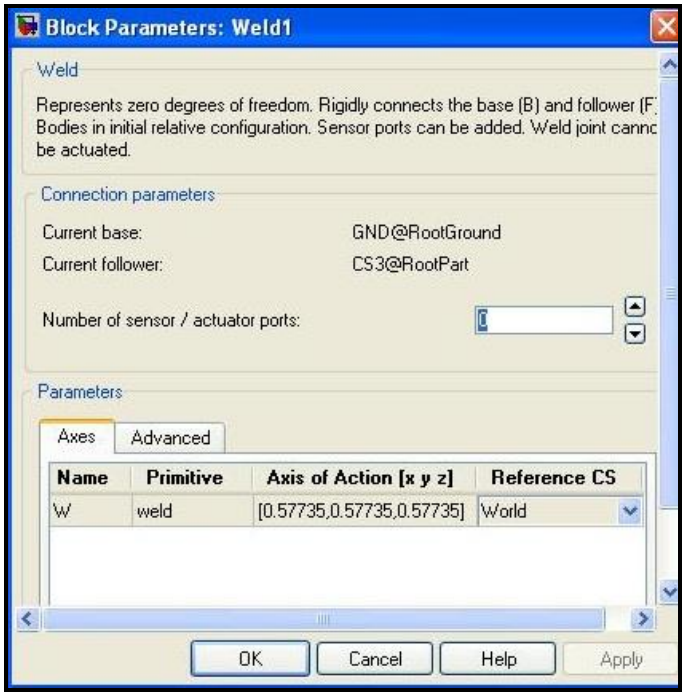


Figura 3.8: Configuración de los parámetros de la base del robot.

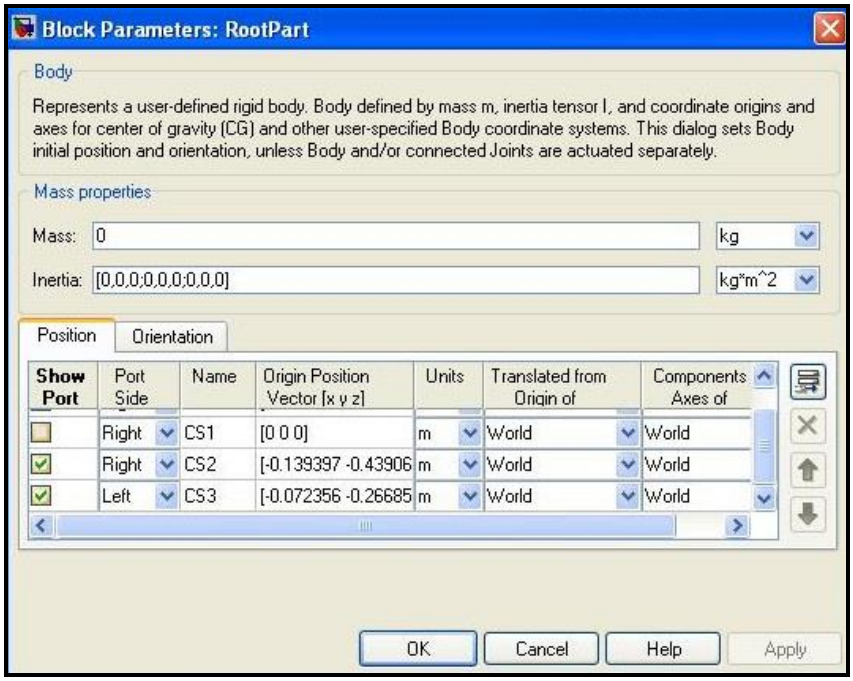


Figura 3.9: Definición de los parámetros de la dinámica del robot.

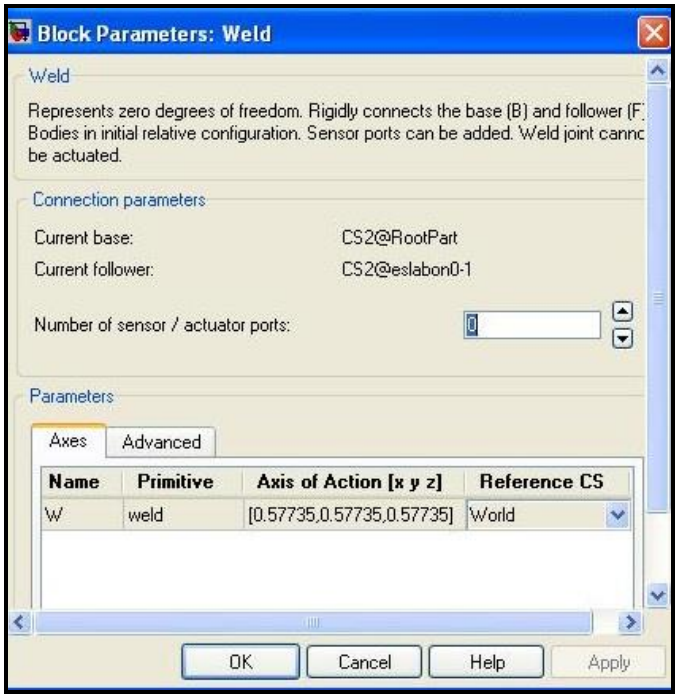


Figura 3.10: Configuración de los parámetros de la articulación fija del eslabón 0 del robot.

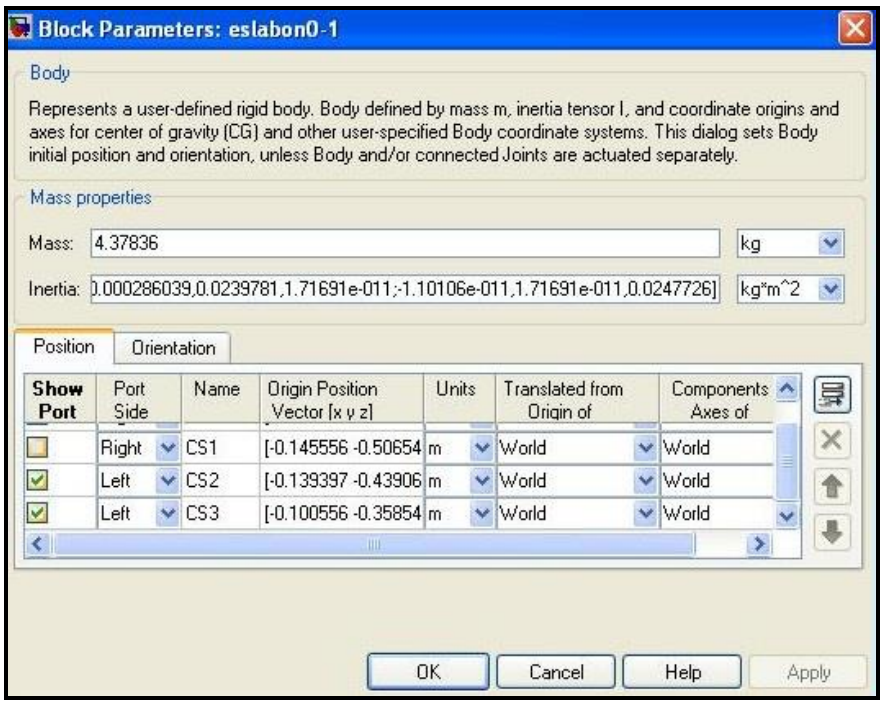


Figura 3.11: Configuración de los parámetros del eslabón 0 del robot.

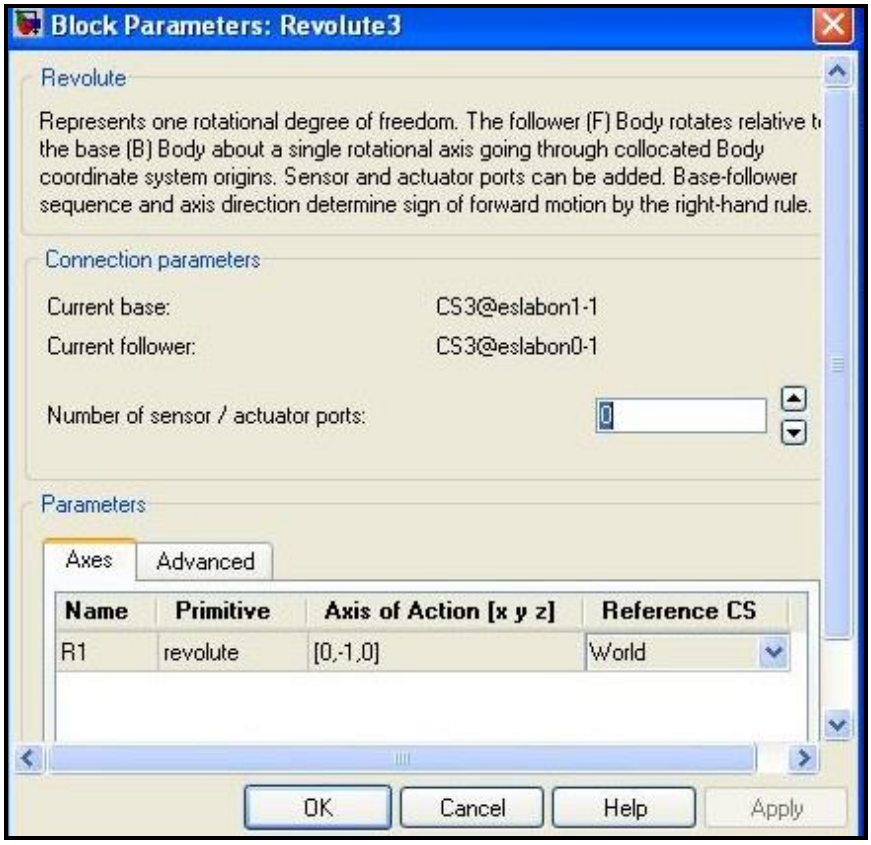


Figura 3.12: Configuración de los parámetros de la articulación que une los eslabones 0 y 1.

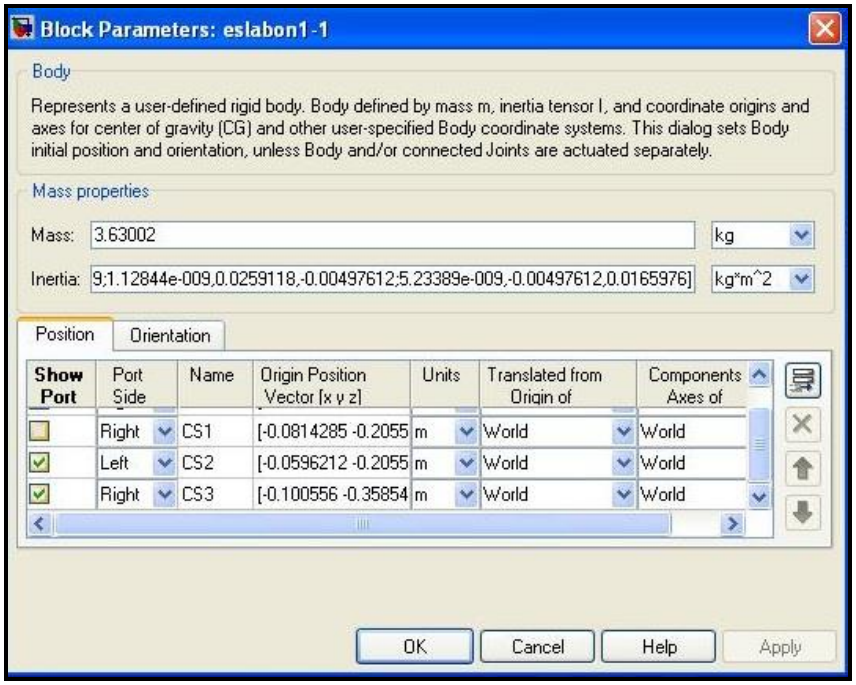


Figura 3.13: Configuración de los parámetros del eslabón 1 del robot.

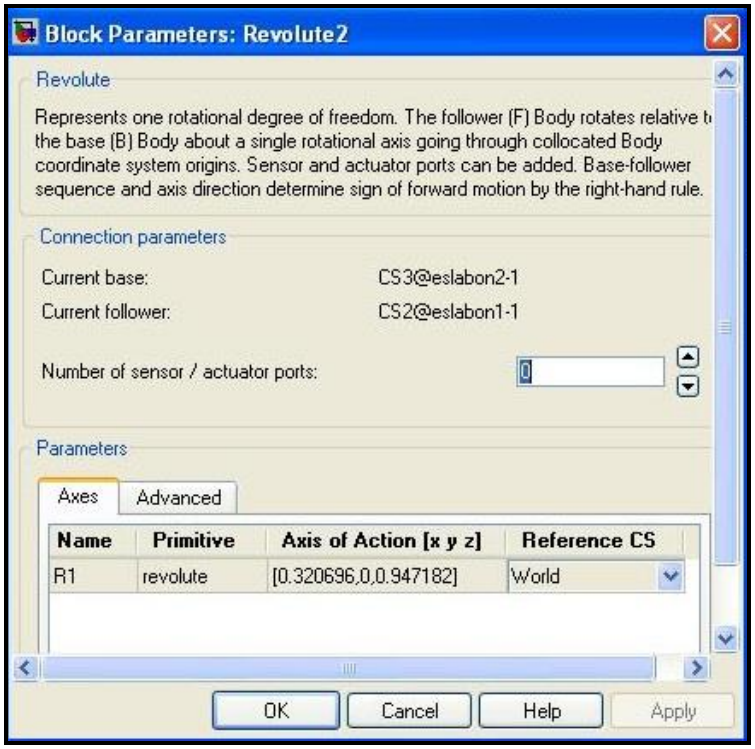


Figura 3.14: Configuración de los parámetros de la articulación que une los eslabones 1 y 2.

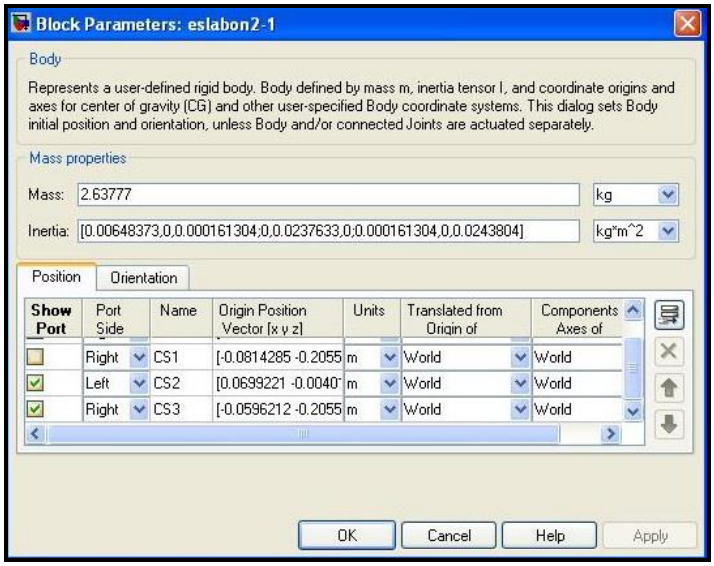


Figura 3.15: Configuración de los parámetros del eslabón 2 del robot.

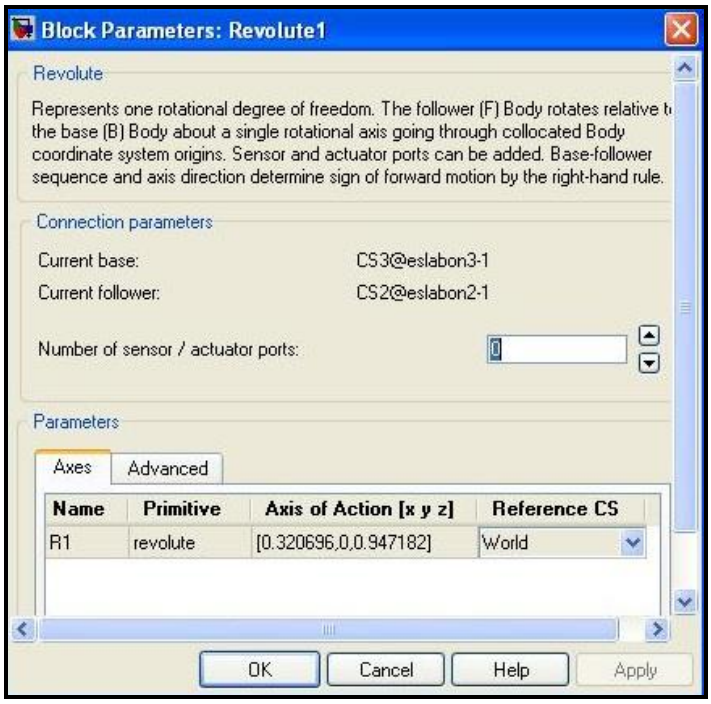


Figura 3.16: Configuración de los parámetros de la articulación que une los eslabones 2 y 3.

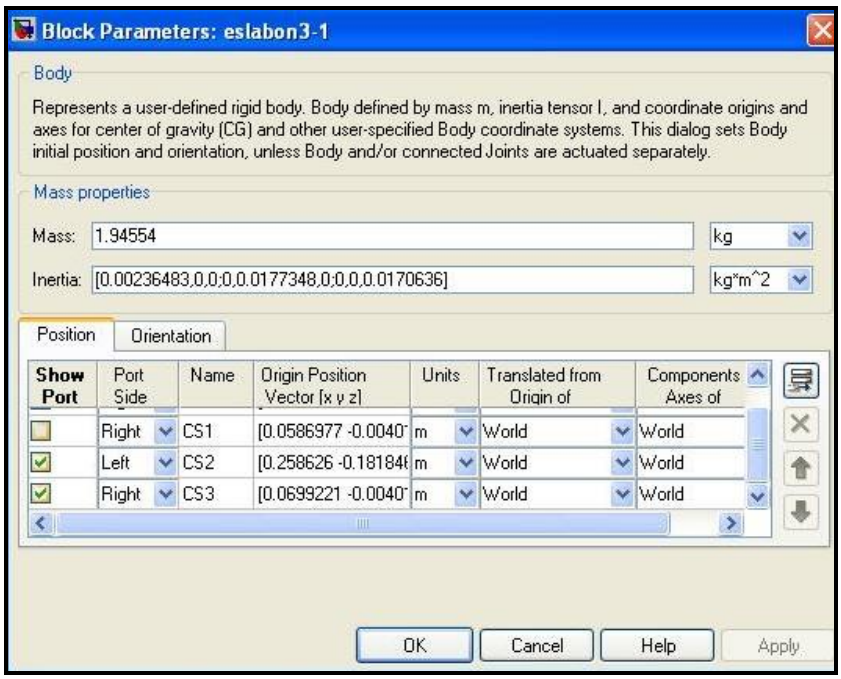


Figura 3.17: Configuración de los parámetros del eslabón 3 del robot.

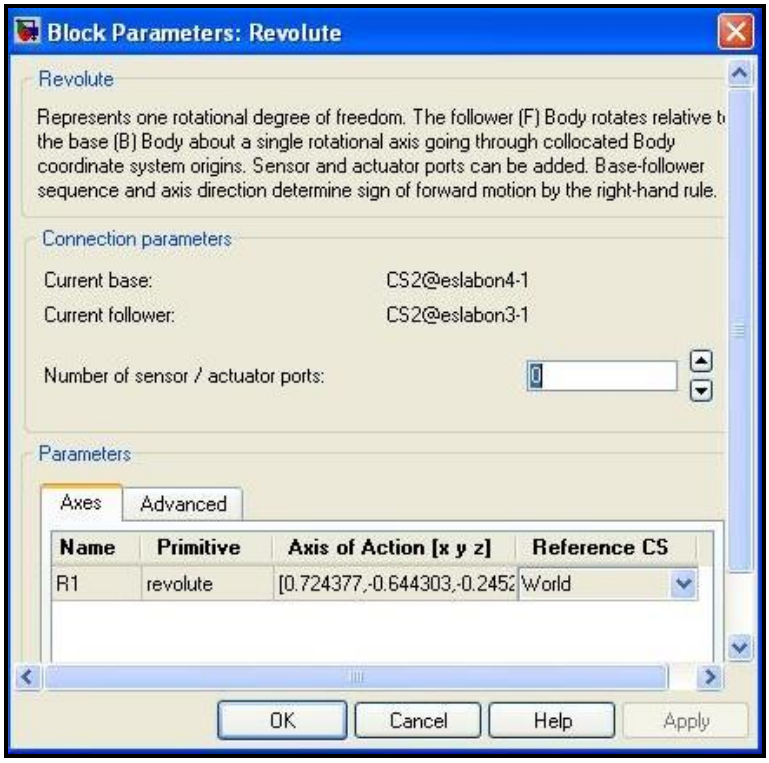


Figura 3.18: Configuración de los parámetros de la articulación que une los eslabones 3 y 4.

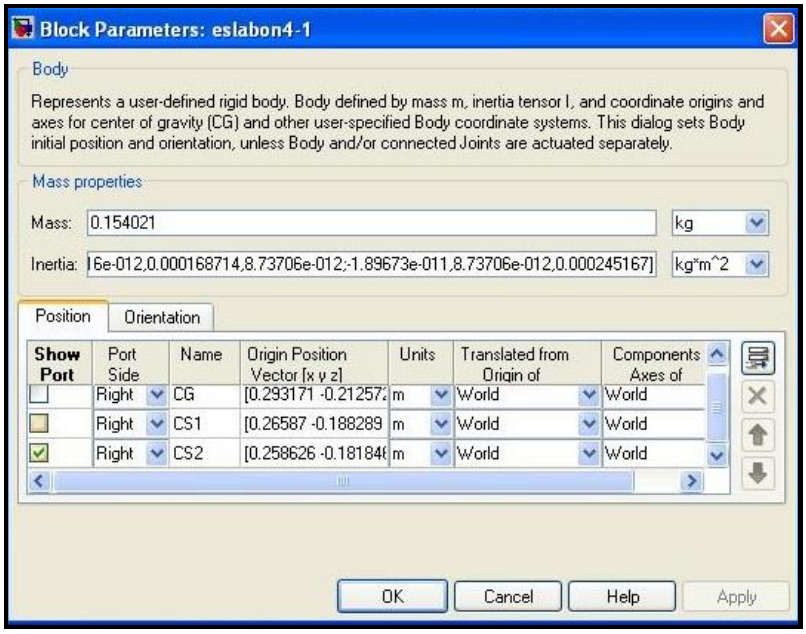


Figura 3.19: Configuración de los parámetros del eslabón 4 del robot.

A partir de la ecuación dinámica del robot:

$$\boldsymbol{\tau = H(q)\ddot{q} + C(q, q') + h(q)}$$

Dónde:

H(q) = Tensor de Inercia

C(q,q') = Vector de Fuerza generado por Coriolis

h(q) = Vector de Fuerza generado por la Gravedad

En la Figura 3.20 se diseña el diagrama en Simulink para la dinámica de Lagrange del robot:

$$\boldsymbol{\ddot{q} = H(q)^{-1}[\tau - C(q, q') - h(q)]}$$

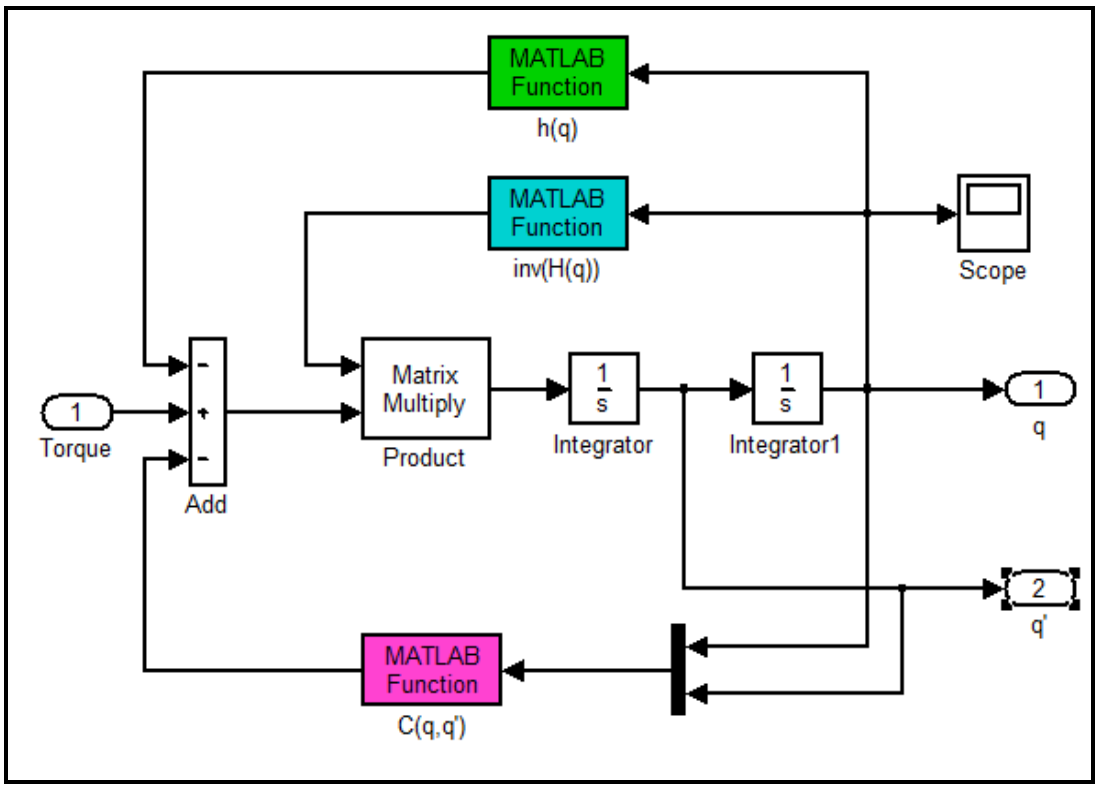


Figura 3.20: Dinámica de Lagrange del robot en SIMULINK.

3.6. CONTROL FUZZY PARA SINTONIZACIÓN PID

La lógica difusa es una herramienta de la inteligencia artificial que en los años recientes viene siendo utilizada en diversos sistemas de control, pero veamos brevemente en que se basa este tipo de control.

La lógica difusa inicia con el concepto de conjunto Fuzzy. A diferencia de un conjunto clásico, en un conjunto Fuzzy no todos sus elementos están totalmente definidos, es decir hay un cierta probabilidad manejada por una función de membresía que indicará en qué medida una variable pertenecerá a un conjunto o no.

Pero si este concepto parece demasiado superficial y difícil de entender, veámoslo con un ejemplo muy conocido, si queremos listar el conjunto dado por los días que corresponden a fines de semana, muchos diremos que sábado y domingo pertenecerán a este conjunto, pero

quizá algunos consideraremos que viernes también pertenece a este conjunto, es aquí cuando se define un conjunto Fuzzy y es la clave del concepto anterior.

De forma general un sistema de control con lógica difusa que presenta las siguientes partes:

Interface de fuzzyficación: convierte los valores numéricos de entrada (números reales) en conjuntos difusos.

Aplicación de las Operaciones difusas: en esta fase se analizarán y ejecutarán las reglas de producción.

Aplicando Método Implicación: Con la operación ya obtenida en el paso anterior se truncará la función de membresía de salida, generalmente triangular.

Acoplando Salidas: Debido a que las decisiones son tomadas por una serie de reglas, las salidas de cada regla deben ser sumadas para luego poder obtener una decisión final.

Interface de Defuzzificación: En esta fase final se transformará las variables lingüísticas de la salida en valores numéricos. El método más conocido es el método del cálculo del centroide de la curva final obtenida del penúltimo paso.

La lógica difusa es una lógica alternativa a la lógica clásica que pretende introducir un grado de vaguedad en las cosas que evalúa ya que en el mundo en que nos desarrollamos existe mucho conocimiento ambiguo e impreciso, por lo que la lógica difusa fue diseñada para imitar el comportamiento del ser humano.

La lógica difusa en comparación con la lógica convencional permite trabajar con información que no es exacta, en oposición de la lógica convencional que permite trabajar con información exacta y precisa.

La lógica difusa se puede aplicar en procesos demasiados complejos, cuando no existe un modelo matemático preciso. También se puede aplicar cuando ciertas partes del sistema a controlar son desconocidas y no pueden medirse en forma confiable y cuando el ajuste de una variable puede producir el desajuste de otras.

Para diseñar el controlador difuso se utilizó el Fuzzy Logic Toolbox, que es una herramienta de Matlab que permite simular el comportamiento de un controlador difuso mediante un Fuzzy Inference Sistem (FIS) o sistema difuso de inferencia. La interfaz gráfica de un FIS hace que la programación de un controlador difuso sea muy sencilla y amigable.

Para utilizar esta herramienta se teclea la palabra fuzzy en la ventana de comandos de Matlab, la cual abrirá la ventana FIS Editor, la cual se muestra en la figura 3.21.

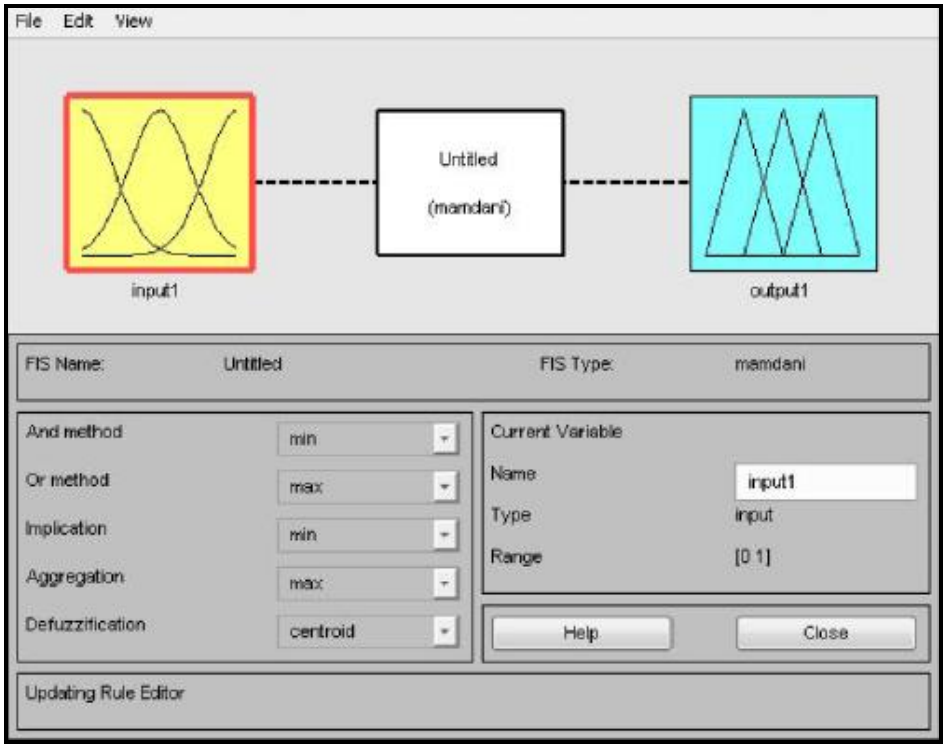


Figura 3.21: Ventana FIS Editor.

El método de inferencia que se aplicó es de Mandani y después definimos las variables de entrada y salida que se utilizaron en el controlador, es decir configuramos el FIS. A continuación se siguieron los siguientes pasos:

Paso N°01: Configuración y programación del controlador difuso

La programación y configuración del controlador se realizó utilizando Fuzzy Logic Toolbox, herramienta que permitió elegir el método de inferencia, el método de defusificación, editar

las variables y los conjuntos difusos, así como las reglas que describirán el comportamiento de los controladores.

Paso N°02: Definición de las variables

Las variables utilizadas son el error generado por el ángulo deseado de la articulación y el ángulo actual de la articulación y el cambio en el error que es la velocidad a la que se mueve la articulación. A la salida del controlador se debe de generar el torque que moverá a las articulaciones.

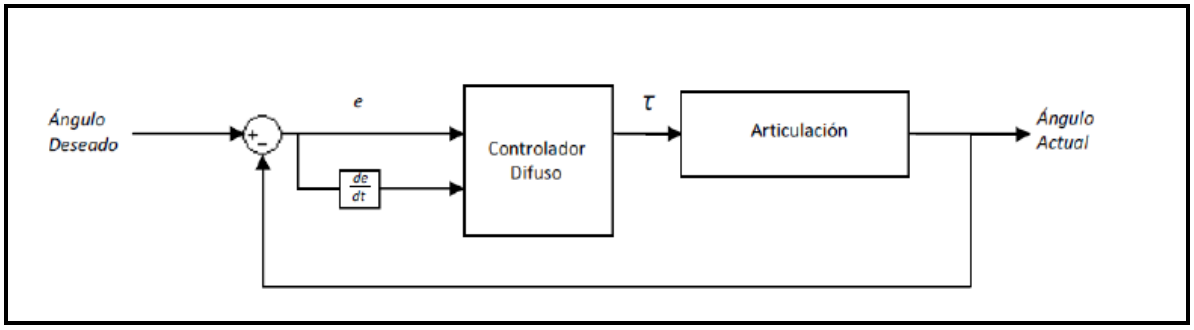


Figura 3.22: Diagrama de bloques del control.

Paso N°03: Articulación 01 (de la base).- Las variables de entrada y de salida para controlar el movimiento de la primera articulación, se conforman por cinco conjuntos difusos tal como se muestra en la figura 3.23.

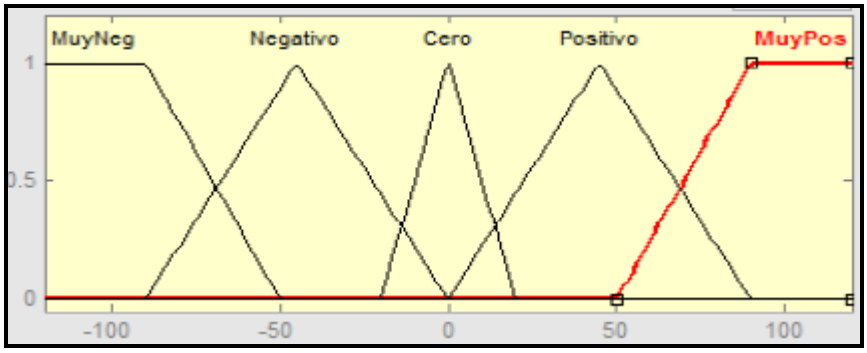


Figura 3.23: Conjuntos difusos de la variable error.

El rango de valores para esta articulación se ha considerado de 120º.

Los conjuntos difusos para la variable velocidad, así como las etiquetas lingüísticas se muestran en la Figura 3.24.

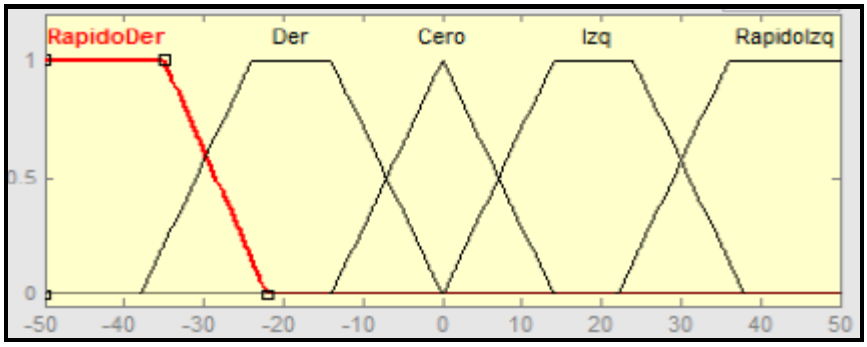


Figura 3.24: Conjuntos difusos de la variable velocidad de error.

Consideramos que la articulación 1 se mueve a una velocidad máxima de 50 grados por segundo para un torque de 2N.m, las etiquetas lingüísticas se muestran en la figura 3.25.

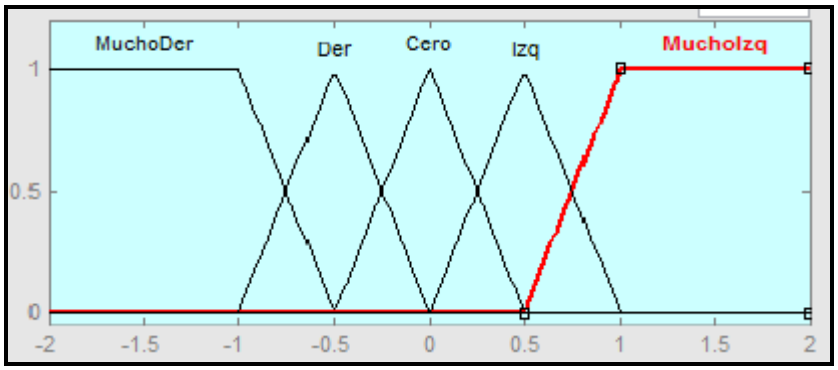


Figura 3.25: Conjunto difusos de la variable Torque de salida.

Luego de determinar las variables de entrada con sus correspondientes conjuntos difusos, determinaremos las reglas para conformar la base de conocimiento que regirán el comportamiento del controlador. La tabla 3.2 muestra la tabla de inferencia generada para la articulación 1.

Torque		Velocidad				
		Rapido Der	**Der**	**Cero**	**Izq**	**Rápido Izq**
ERROR	**Muy Neg**	Cero	Der	Mucho Der	Mucho Der	Mucho Der
	Negativo	Der	Cero	Der	Mucho Der	Mucho Der
	Cero	Mucho Izq	Izq	Cero	Der	Mucho Der
	Positivo	Mucho Izq	Mucho Izq	Izq	Cero	Izq
	Muy Positivo	Mucho Izq	Mucho Izq	Mucho Izq	Izq	Cero

Tabla 3.2: Tabla de inferencia para la articulación 01.

Paso N°04: Configuración del controlador difuso en Fuzzy Logic Toolbox

Definidas las variables de entrada y salida, las reglas de la base de conocimiento así como el método de inferencia y defusificación que se utilizaron, finalmente se agregó esta información en el sistema de inferencia difuso (FIS). La configuración de las demás articulaciones se muestra en el anexo 04.

Primero se definió el método de inferencia que utilizó el FIS y se creó las variables que se definieron anteriormente como se muestra en la figura 3.26.

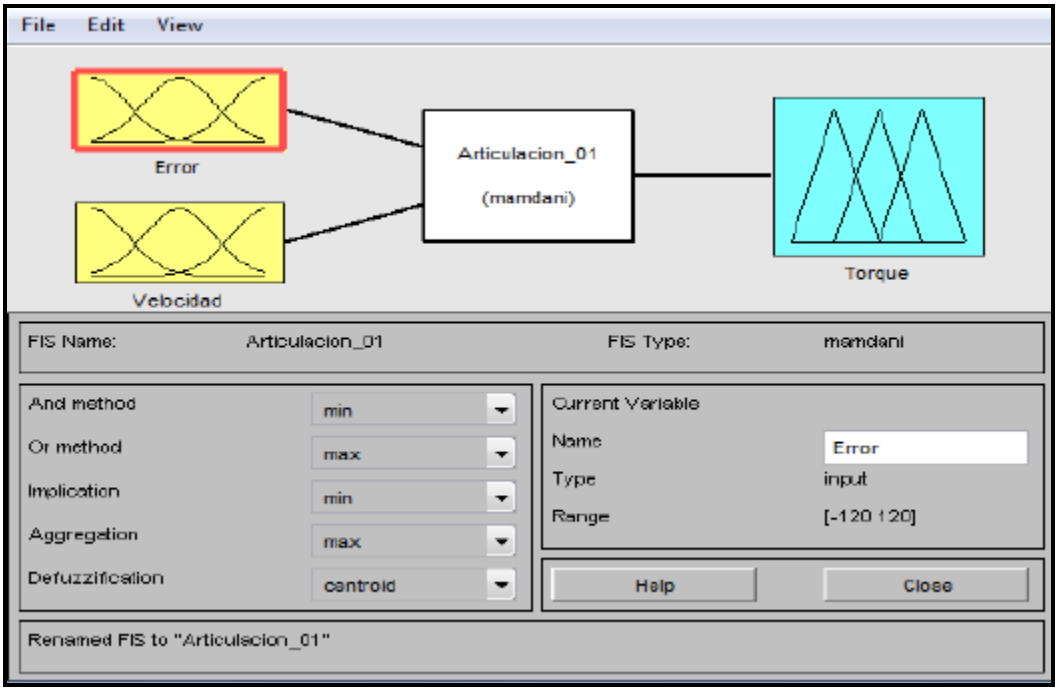


Figura 3.26: Definición de las variables en la ventana FIS Editor.

La ventana de edición del FIS muestra las variables de entrada y salida así como el método de inferencia que es el Mandani. En la misma ventana se seleccionó el método de defusificación que es el método del Centroide.

Luego se definieron los conjuntos difusos de las variables de entrada y salida tal como se muestra en la figura 3.27. En esta ventana se define la forma de los conjuntos difusos, así como los parámetros que marcaron los límites de los conjuntos y las etiquetas lingüísticas.

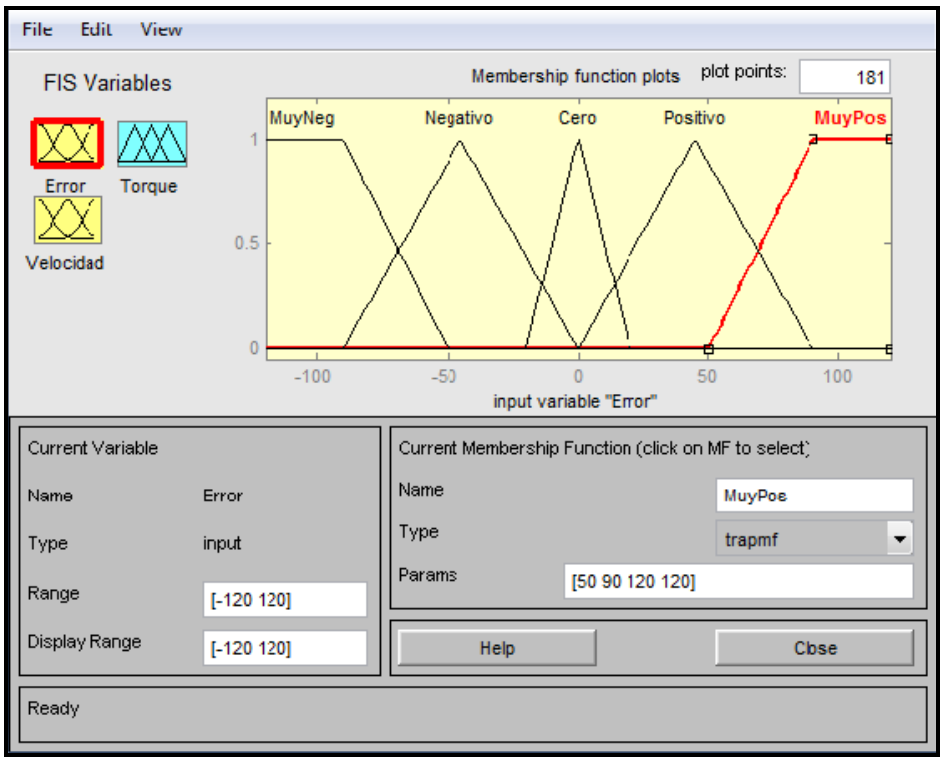


Figura 3.27: Definición de los conjuntos difusos de la variable error.

CAPÍTULO IV : PRUEBAS Y RESULTADOS

4.1. RESULTADOS DE APLICACIÓN DE LOS CONTROLADORES CLÁSICOS

En este subcapítulo de la tesis se presentan los resultados de aplicación de los controladores clásicos con respecto al robot PUMA de cuatro grados de libertad. Considerando los siguientes valores de ganancia Kp, Ki, Kd y el valor referencial q para cada una de las cuatro articulaciones del robot PUMA, se procede a la simulación respectiva en simulink.

Kp=diag([100,100,60,6])

Ki=diag([30,50,90,3])

Kd=diag([56,64,24,0.8])

Con referencia: q = [1 0 -0.8 0]

4.1.1. CONTROLADOR PROPORCIONAL (P)

A continuación se presenta el diagrama simulink del controlador tipo P aplicado al robot PUMA de cuatro grados de libertad.

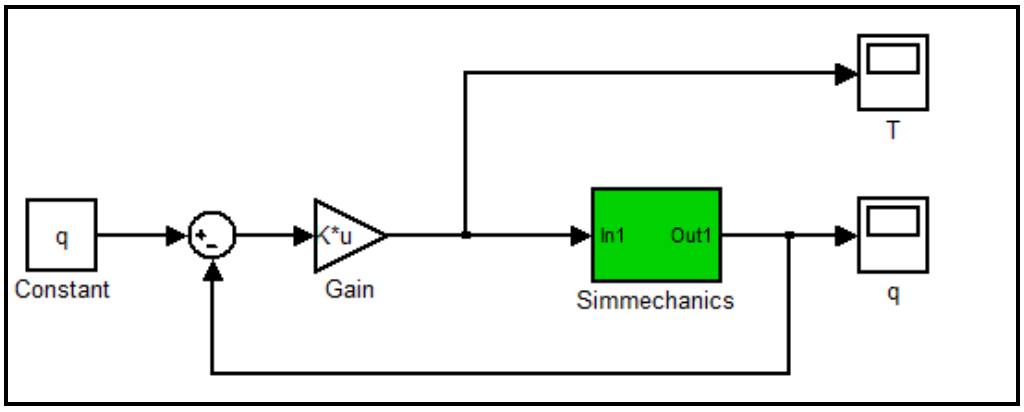


Figura 4.1: Controlador Proporcional.

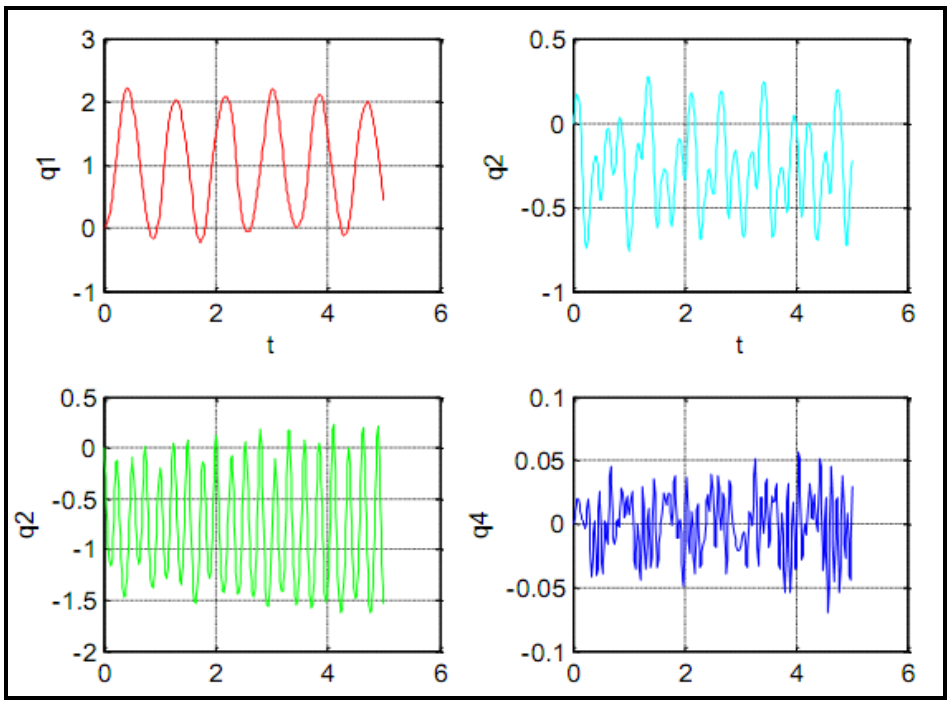


Figura 4.2: Salida de Articulaciones P.

Al observar la Figura 4.2 se concluye que el controlador tipo P es insuficiente ya que necesita amortiguar las oscilaciones.

4.1.2 CONTROLADOR PROPORCIONAL DERIVATIVO (PD)

A continuación se presenta el diagrama simulink del controlador tipo PD aplicado al robot PUMA de cuatro grados de libertad.

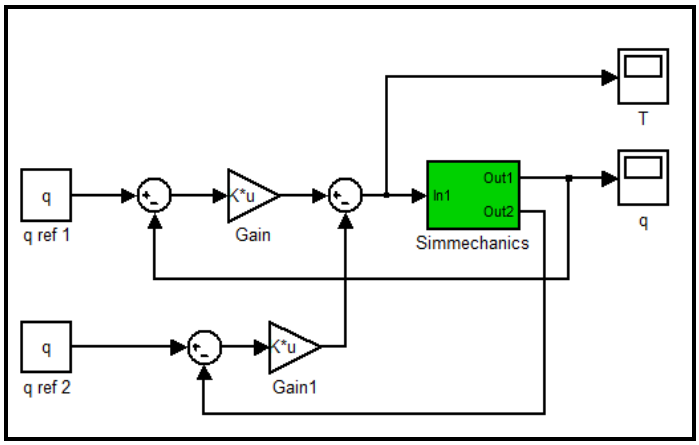


Figura 4.3: Controlador PD.

Al observar la Figura 4.3 el controlador tipo PD amortigua las oscilaciones. Nótese que una articulación complicada es q_2 ya que en el giro coordinado que se está realizando la tercera articulación está afectada en gran medida por la gravedad.

4.1.3. CONTROLADOR PD + COMPENSACIÓN G

A continuación se presenta el diagrama simulink del controlador tipo PD + compensación G aplicado al robot PUMA de cuatro grados de libertad.

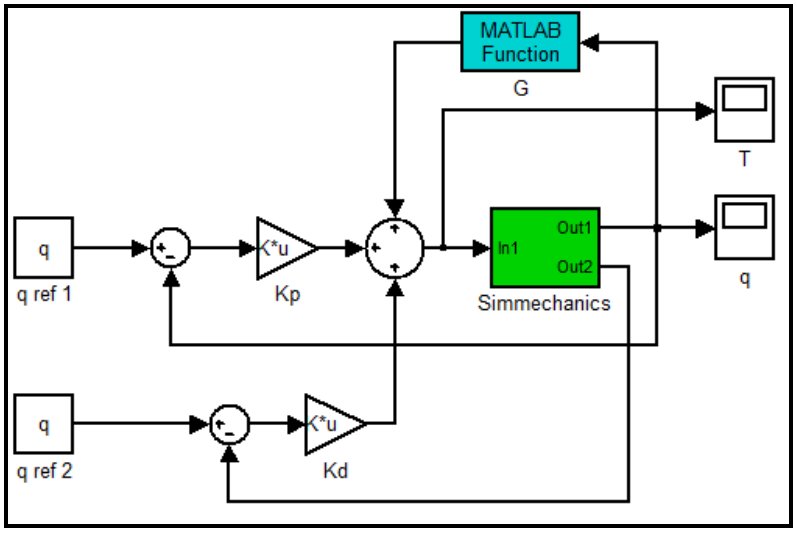


Figura 4.4: Controlador PD más Compensador G.

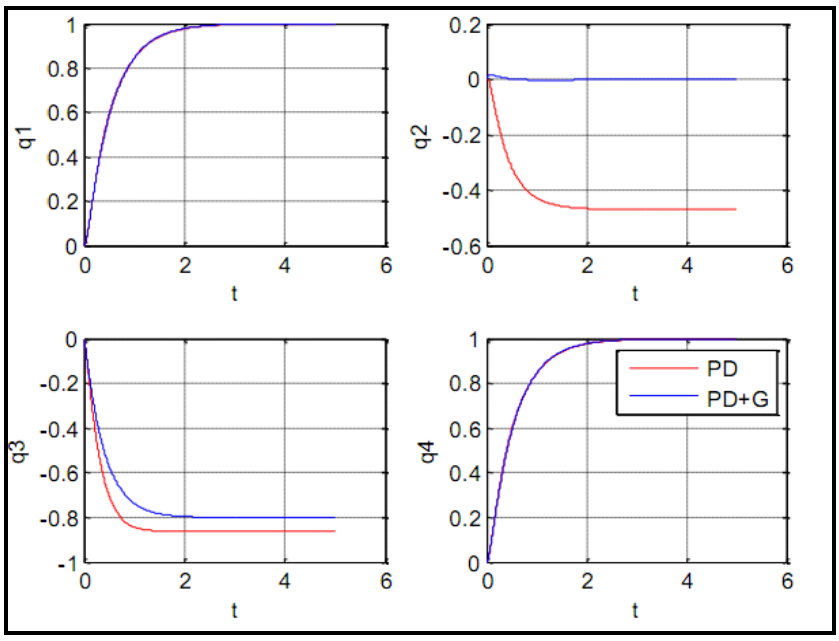


Figura 4.5: Salidas de Articulaciones (Rad).

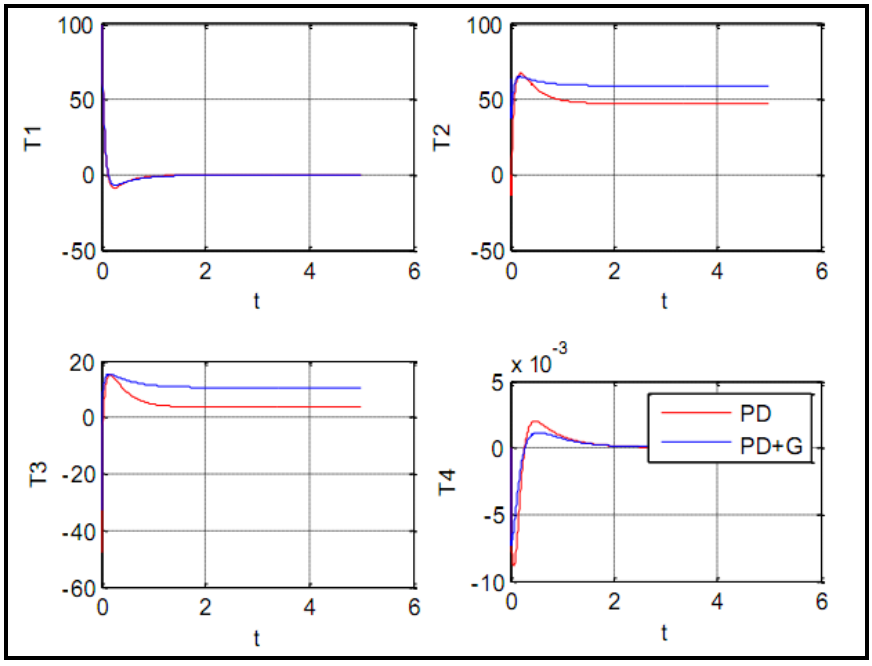


Figura 4.6: Torques utilizados (N.m).

Al observar la Figura 4.5 y 4.6 y comprar los controladores PD y PD + G se concluye que ambos amortiguan las oscilaciones. La articulación complicada es q_2 ya que en el giro coordinado que se está realizando la tercera articulación está afectada en gran medida por la gravedad y esto se corrige al compensar G.

4.1.4 CONTROLADOR PID

A continuación se presenta el diagrama en simulink del controlador PID

aplicado al robot PUMA de cuatro grados de libertad:

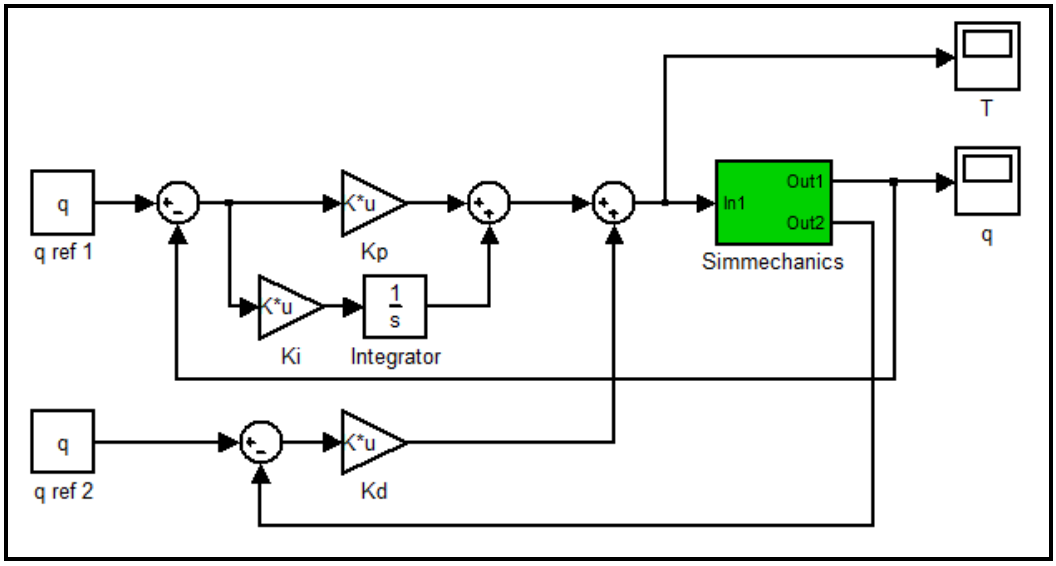
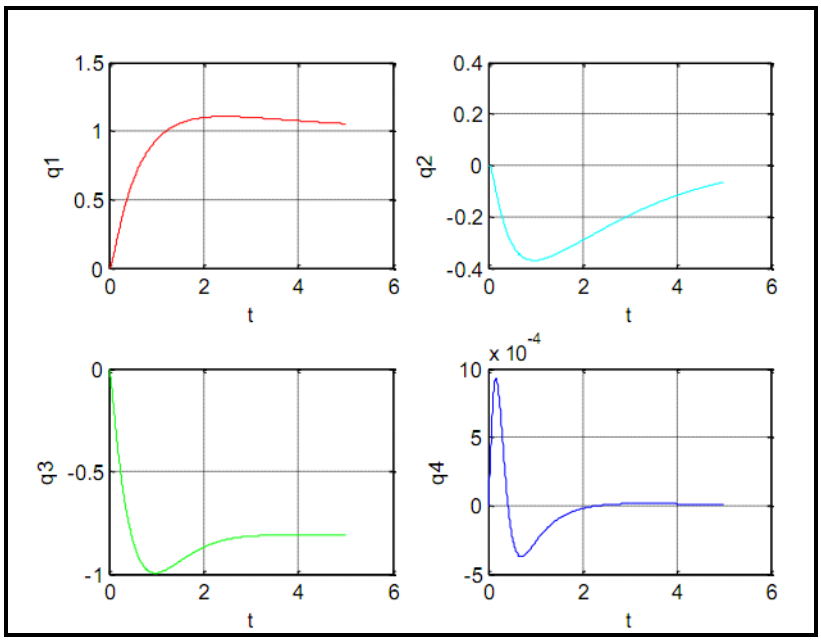


Figura 4.7: Controlador PID.



` Figura 4.8: Salidas de Articulaciones PID (Rad).

El integrador sirve para disminuir (tender a cero) los errores en estado estacionario por lo que q_2 tiende a ser cero sin necesidad de compensar la gravedad como se observa en la figura 4.8 y los torques necesarios para este controlador PID son similares a los de la figura 4.6.

Tenemos valores de ganancia:

Kp=diag([100, 100, 60, 6])

Ki=diag([30, 50, 90, 3])

Kd=diag([56, 64, 24, 0.8])

A continuación se tiene el robot modelado con Lagrange, en SIMULINK:

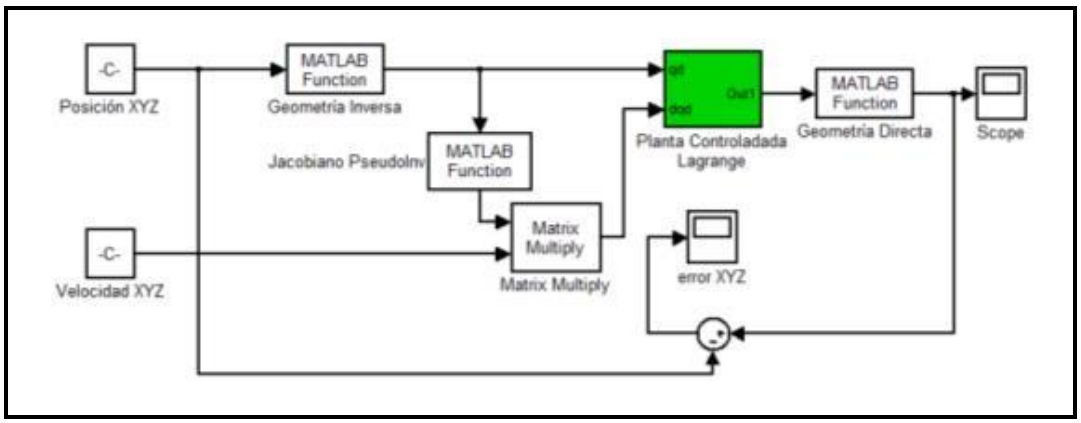


Figura 4.9: Control PID ideal de posición.

Se tiene el robot modelado con Simmechanics:

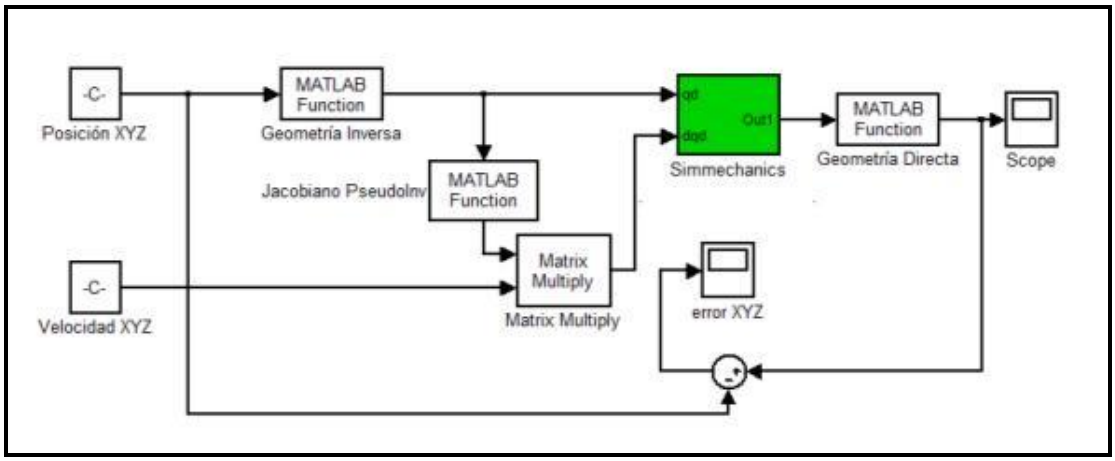


Figura 4.10: Control PID real de posición.

Para la referencia considerada [0.5 0.1 0.3 pi/2]':

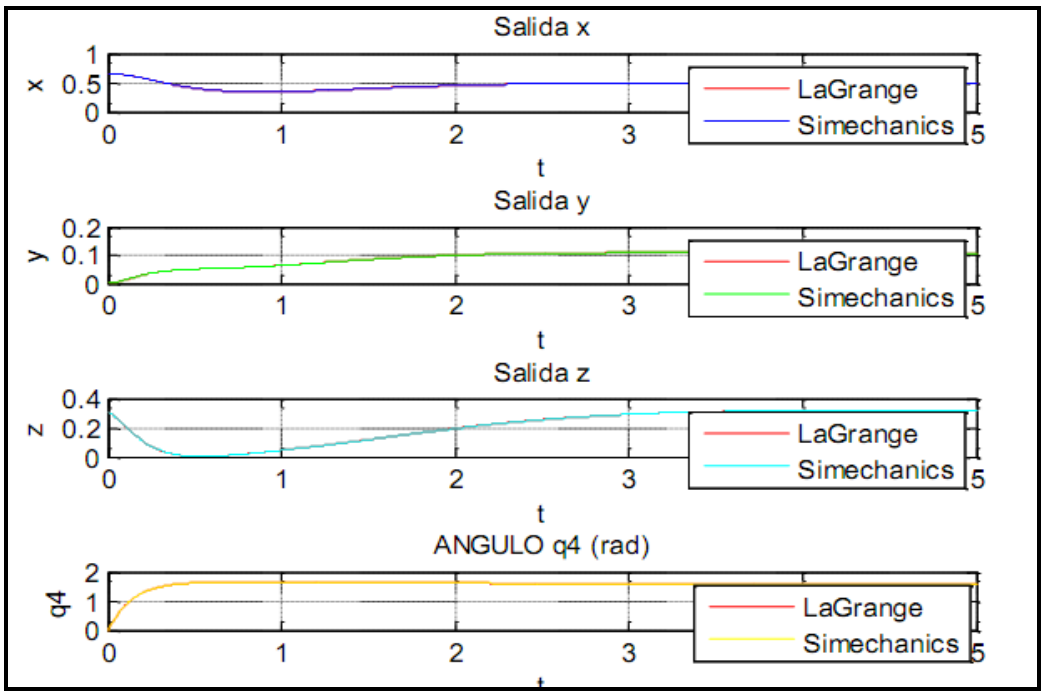


Figura 4.11: Salida Real XYZ.

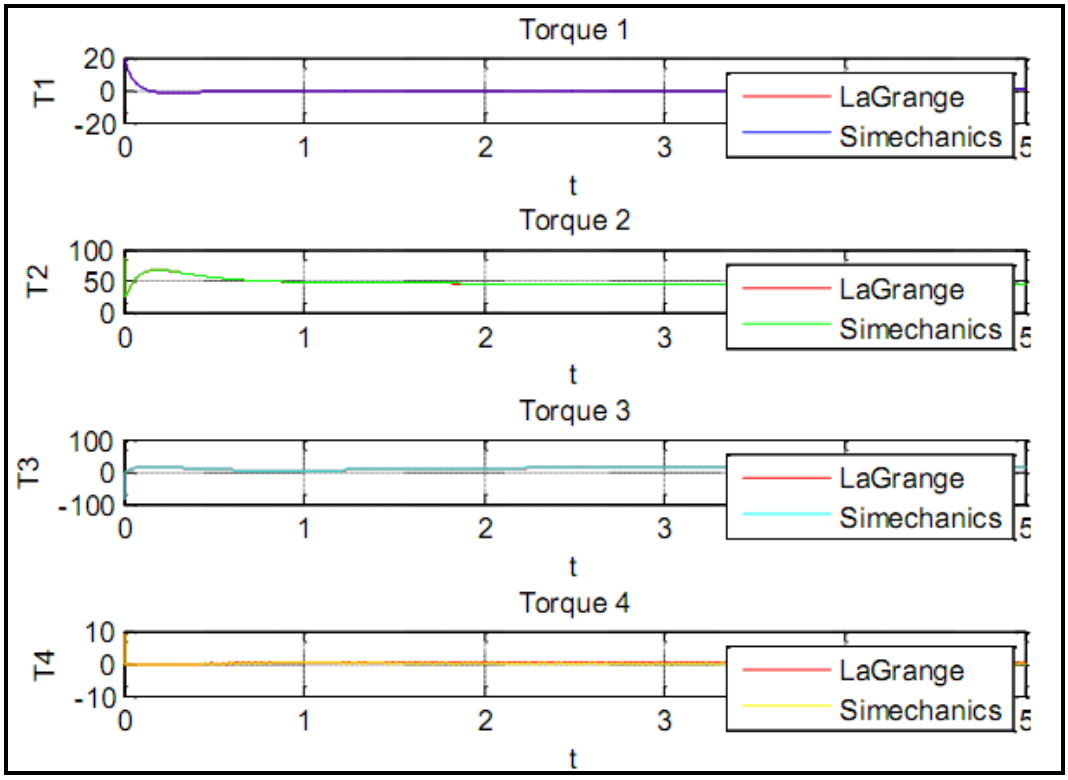


Figura 4.12: Torque hacia la posición referencial.

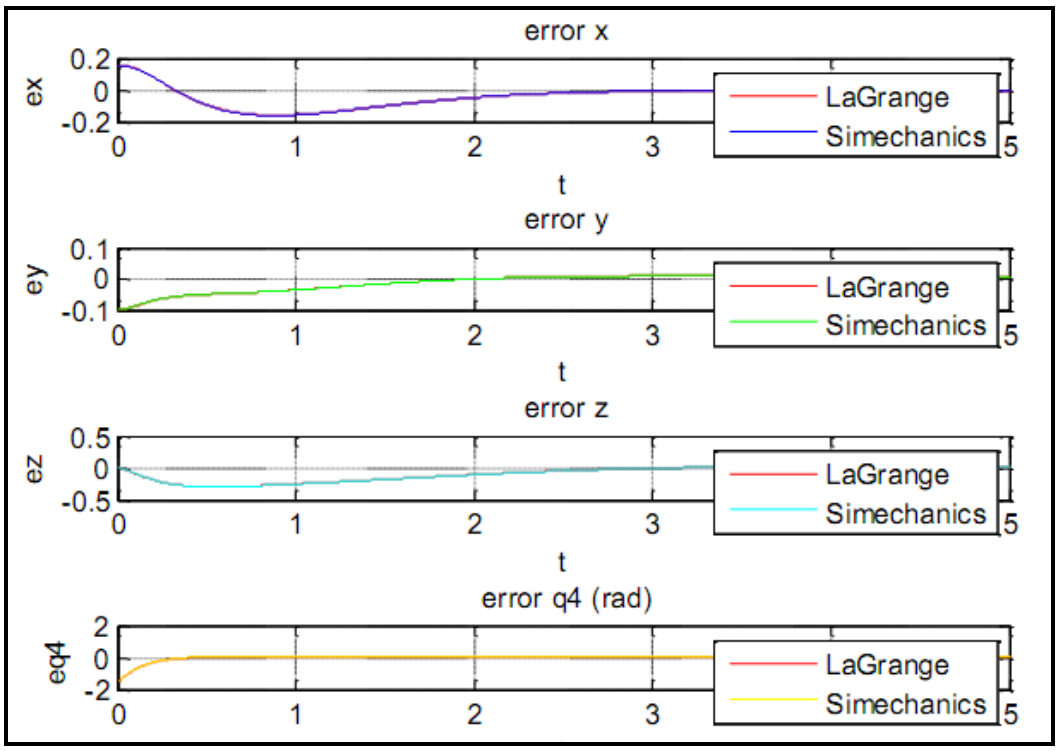


Figura 4.13: Error obtenido XYZ.

Finalmente se observa que con los valores de las ganancias elegidas, la planta se controla adecuadamente y el tiempo de establecimiento aumenta. Esto es prudente para disminuir los torques de actuación, los cuales según la figura 4.12 son siempre menores a 100N.m. Los resultados obtenidos en ambos casos sin compensación de gravedad son parecidos como era de esperarse.

4.2. RESULTADOS DE APLICACIÓN DEL TORQUE COMPUTADO

En este subcapítulo de la tesis se presentan los resultados de aplicación del controlador por torque computado del robot PUMA de cuatro grados de libertad. Considerando los siguientes valores de ganancia K_p, K_i, K_d que van a ser sintonizados se procede a la simulación respectiva en simulink.

K_p=diag([100,100,60,6])

K_i=diag([30,50,90,3])

K_d=diag([56,64,24,0.8])

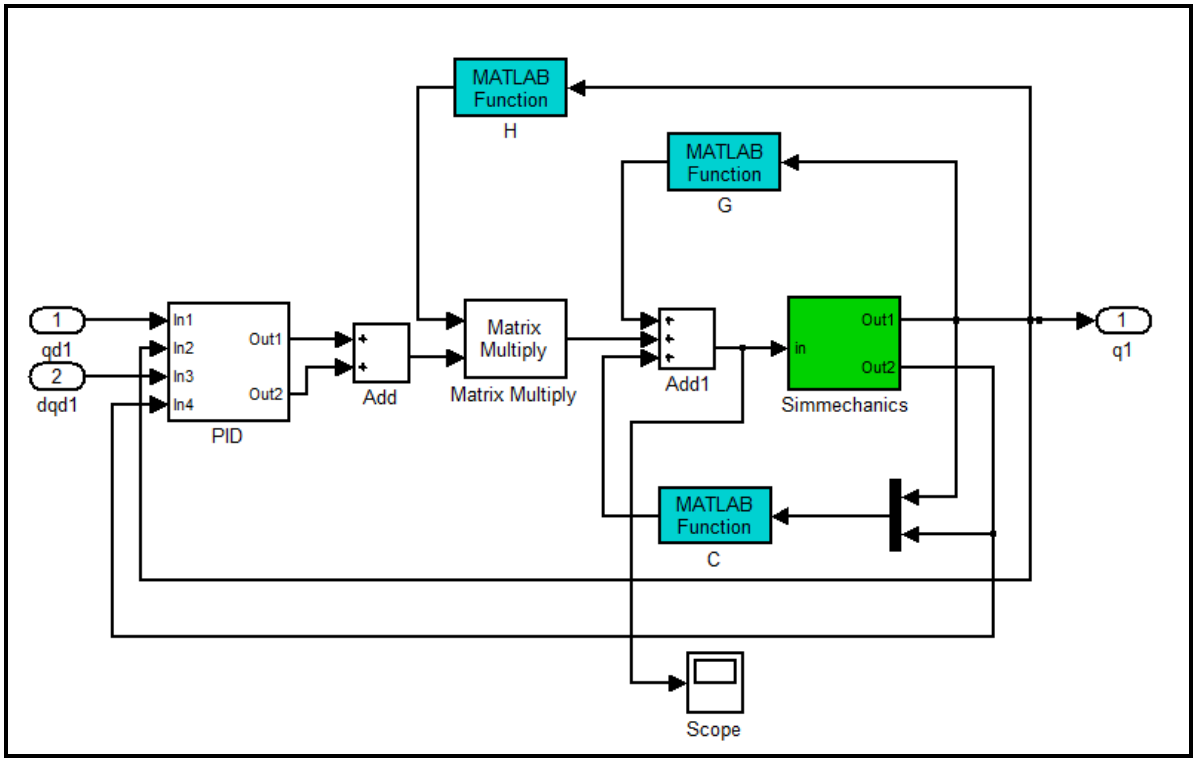


Figura 4.14: Controlador Torque Computado.

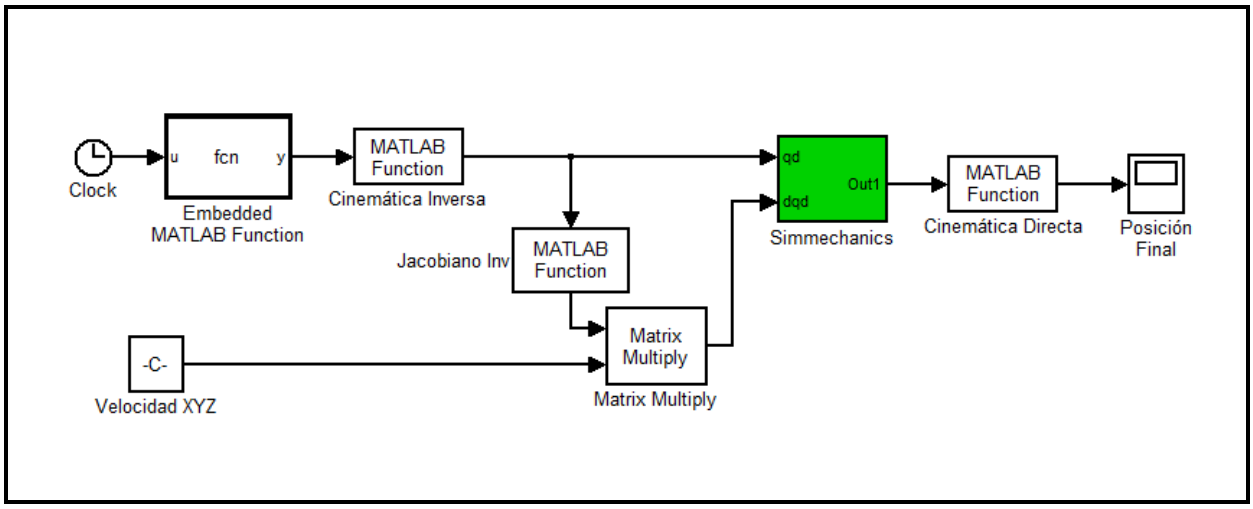


Figura 4.15: Controlador de Trayectoria.

Con la finalidad de estudiar el comportamiento del efector final del robot PUMA controlado por torque computado, con respecto a una trayectoria deseada, se considera la siguiente función:

```
function y = fcn(u)

y=zeros(4,1);

y(1)=0.63*cos(u);%0.653

y(2)=0.63*sin(u);

y(3)=0.002*cos(5*u)+0.305;%0.305

y(4)=0.1*u;
```

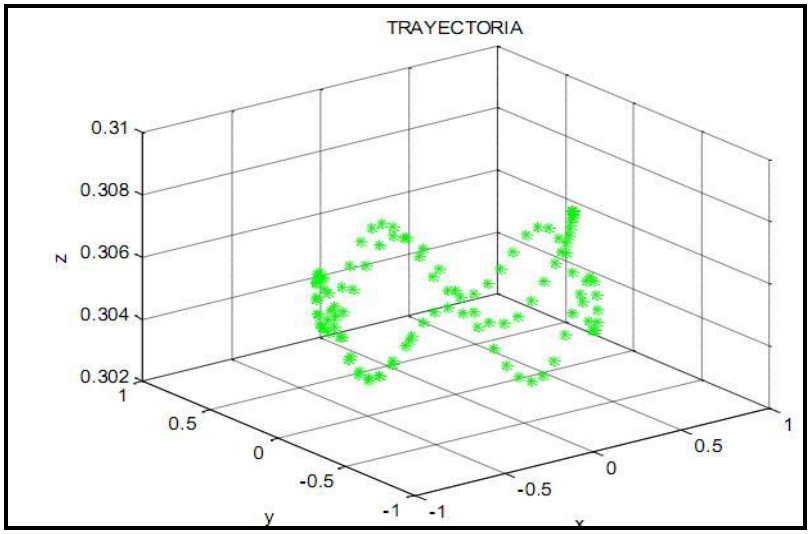


Figura 4.16: Trayectoria del Efector Final.

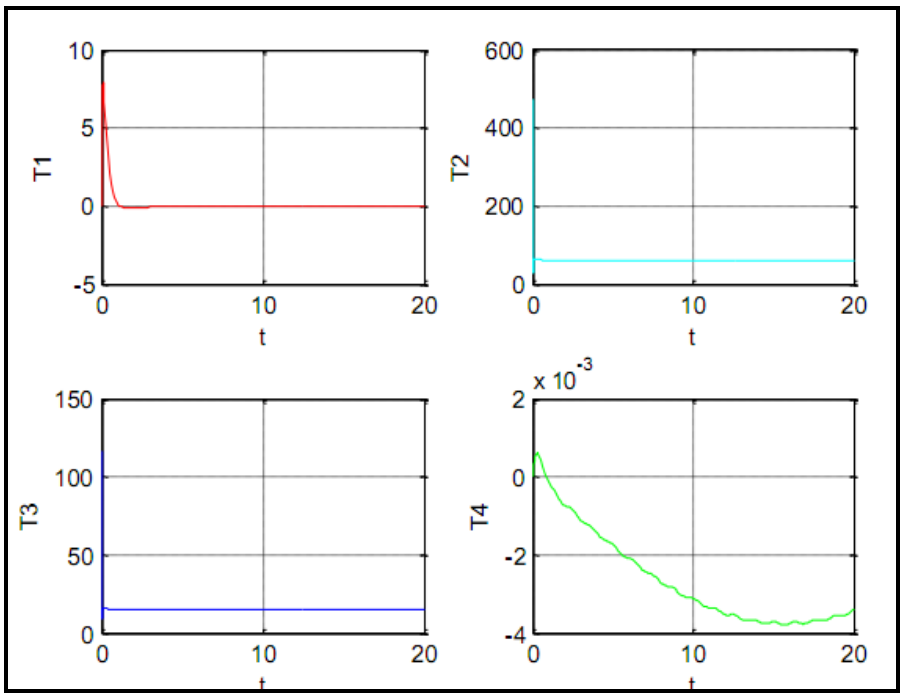


Figura 4.17: Torque para la Trayectoria Deseada.

4.3. RESULTADOS APLICANDO LUH – WALKER

En este subcapítulo de la tesis se presentan los resultados de aplicación del controlador Luh-Walker del robot PUMA de cuatro grados de libertad.

A continuación se presenta el diagrama simulink del controlador Luh-Walker:

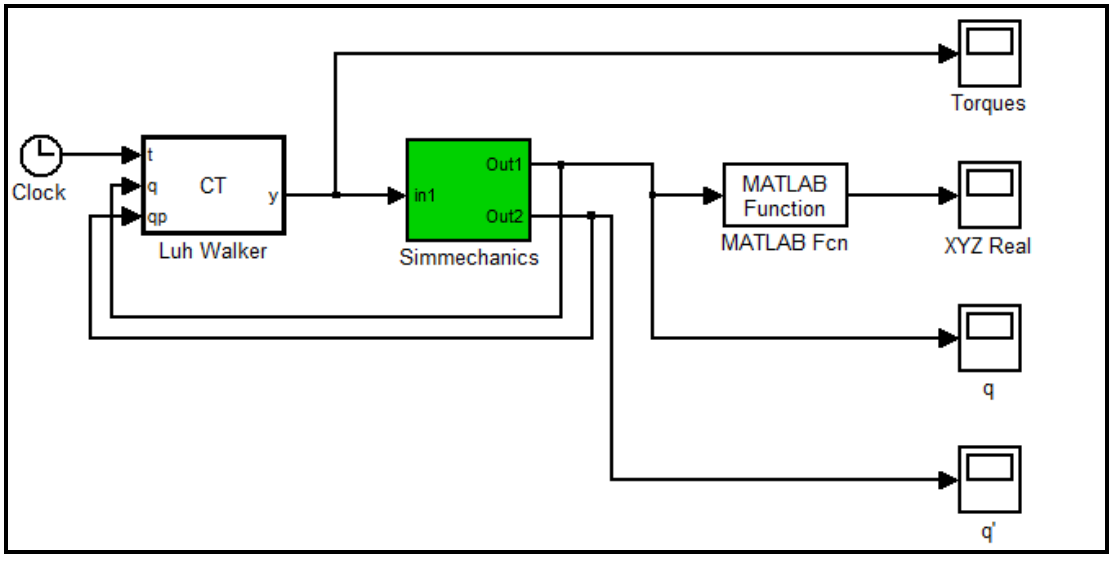


Figura 4.18: Trayectoria del Efector Final.

Considerando la siguiente trayectoria:

```
function r=Trayectoria(t)

r=zeros(3,3);

R=0.62;%0.653

x=R*cos(t);

y=R*sin(t);

z=-0.01*t+0.305;%0.305
```

Y también el controlador PD

```
u=diag([1000,990,800,800])*(eq)+diag([30,30,40,30])*(eqp)
```

En la Figura 4.20, se obtiene la siguiente trayectoria real del efector final del robot PUMA:

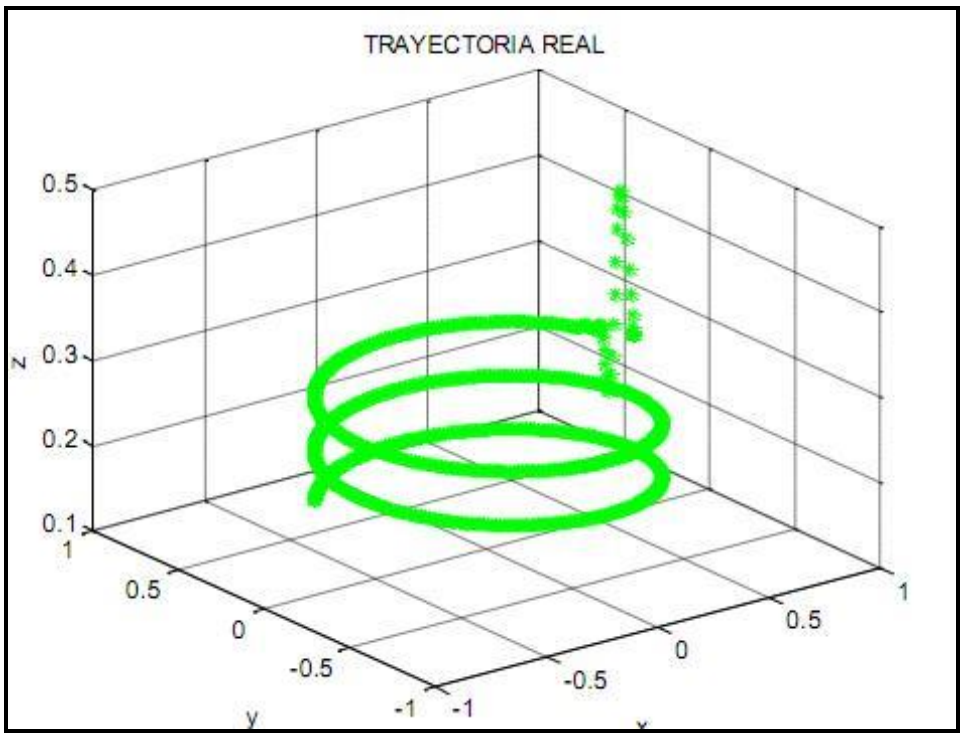


Figura 4.19: Trayectoria del Efector Final.

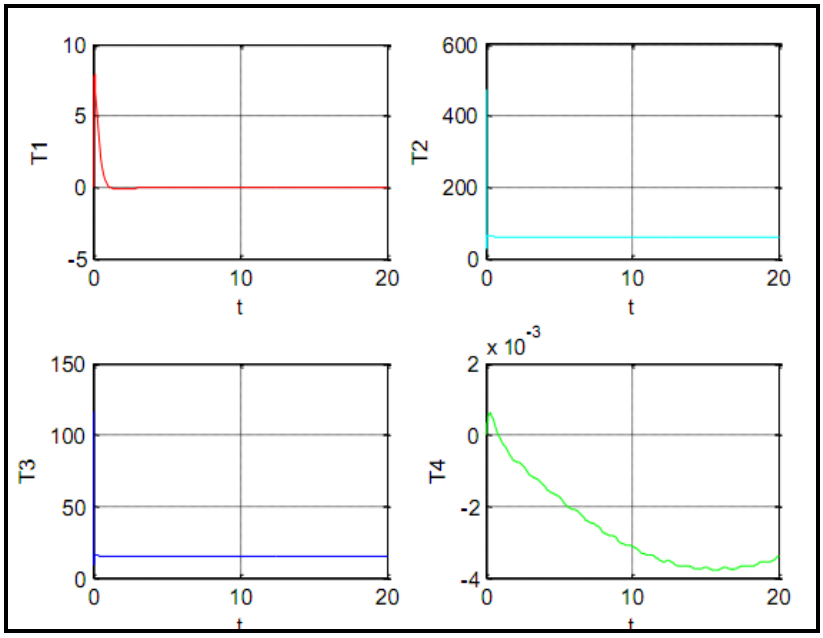


Figura 4.20: Torque en el Luh-Walker.

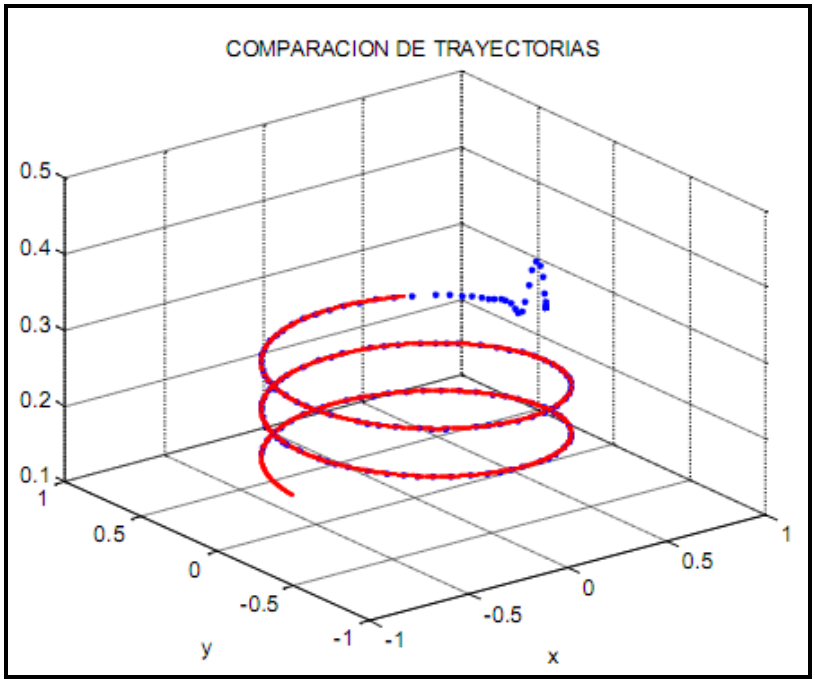


Figura 4.21: Error de la Trayectoria.

4.4. RESULTADOS APLICANDO CONTROL DIFUSO PARA LA SINTONIZACIÒN PID.

En este subcapítulo de la tesis se presentan los resultados de aplicación del controlador difuso para la sintonización PID del robot PUMA de cuatro grados de libertad.

Los resultados obtenidos de la simulación se muestran a continuación:

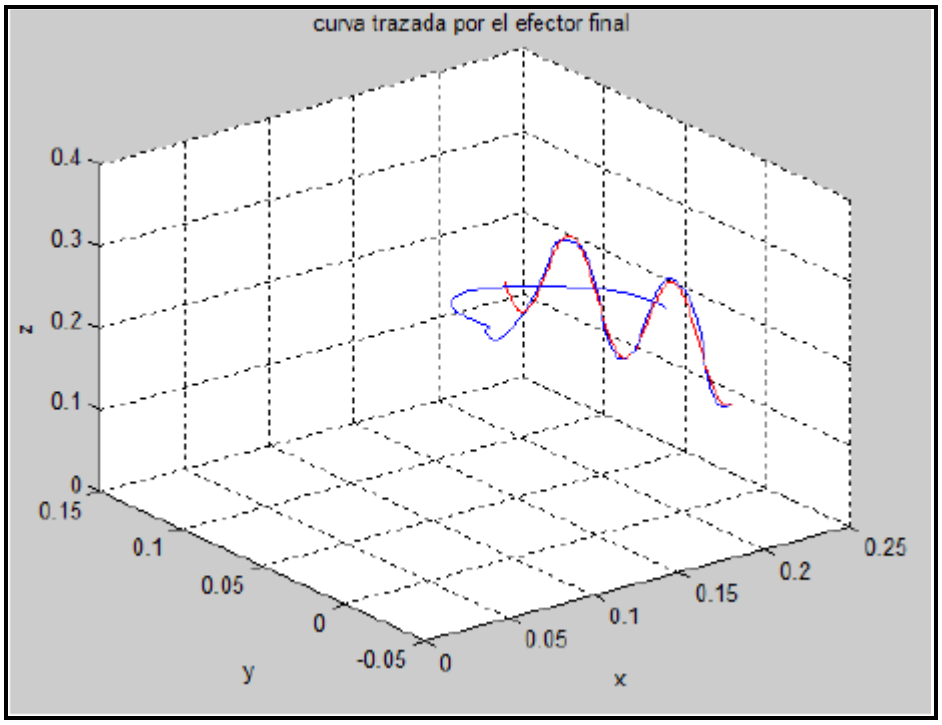


Figura 4.22: Curva trazada por el efector final.

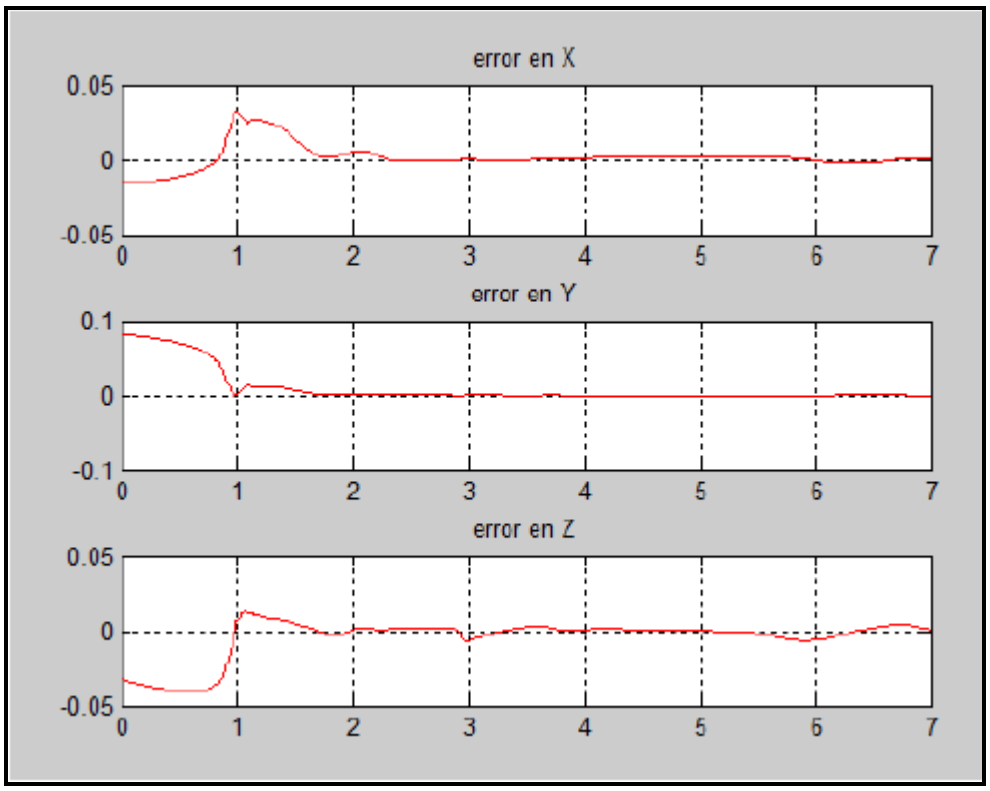


Figura 4.23: Errores obtenidos en el efector final.

En la figura 4.23 la curva de color roja muestra la trayectoria de referencia, mientras que la curva azul muestra la trayectoria real trazada por el efector final del robot. Se puede observar que debido a las condiciones iniciales, el robot tarda un poco en alcanzar la referencia, pero una vez alcanzada no tiene problemas en seguir la trayectoria deseada.

CAPÍTULO V: DISCUSIÓN DE RESULTADOS

5.1. Contrastación de hipótesis con los resultados

De las simulaciones presentadas en la capítulo 4, se observa que el controlador difuso PID supera a los métodos de diseño convencional, como son: el proporcional P, el proporcional derivativo PD, el proporcional derivativo compensando PD + G y el proporcional integral derivativo PID de Ziegler Nichols.

En tal sentido, se ha cumplido con la hipótesis planteada en el trabajo, el cual paso a anotar:

“Es posible optimizar los parámetros del controlador PID aplicado al robot PUMA de cuatro grados de libertad mediante la lógica difusa.”

5.2 Contrastación de resultados con otros estudios similares

Al comparar los resultados obtenidos en el presente trabajo con respecto a los trabajos de Tibaduiza, D., Amaya, I., Rodríguez, S., Mejía, N., Flores, M. (2011), Implementación de un control fuzzy para el control cinemático directo en un robot manipulador, de la Universidad Politécnica de Cataluña, Barcelona, España y la Facultad de Ingeniería Mecatrónica de la Universidad Autónoma de Bucaramanga, Colombia y de Torres, S., Méndez, J. (2009), Seguimiento de trayectorias en robots manipuladores revisión de soluciones y nuevas propuestas, de la Universidad de La Laguna – Tenerife, España. Se observa que mediante la lógica difusa, la trayectoria real del efector final del robot PUMA de cuatro grados de libertad es óptima cuando el sistema del control difuso autosintoniza los parámetros del controlador PID.

CONCLUSIONES

El buen diseño y la implementación de un algoritmo de control, implica conocer ciertas características del modelo del proceso, el mismo que será utilizado en el diseño del control. También se pudo observar que el uso de las ganancias PID involucra tener presente el aporte que genera cada una de esas ganancias, así como el efecto de la variación de las mismas. Además fue necesario conocer dentro de que rangos se encontraban dichas ganancias, lo cual no hubiese sido posible sin valernos de un método complementario que nos diera el punto de arranque, en este caso nos referimos a los resultados obtenidos usando la sintonía de controladores PID utilizando métodos convencionales.

Como se vio en las simulaciones, nos permitieron conocer la trayectoria real del efector final del robot PUMA de cuatro grados de libertad para cada uno de los controladores convencionales y para el controlador difuso.

Por otra parte la lógica difusa brindo resultados aceptables durante la simulación de trayectorias deseadas como se mostró en las simulaciones las cuales se usan frecuentemente en el control de posición de servomotores DC lo que nos lleva a concluir que el algoritmo es muy eficiente para el seguimiento óptimo de trayectorias.

Aún para el robot analizado de tan sólo cuatro GDL, el cálculo y sintonización de las constantes K_p, K_d, K_i para cada articulación resultó demasiado tedioso, debido a esto es que se busca optimizar el ajuste de dichos parámetros utilizando métodos alternativos.

El control difuso resultó muy efectivo dado que, a través de la estimación de los torques a partir de las propiedades del robot, así como su configuración evitamos errores no deseados (se estiman estas propiedades para que «lo real alcance la referencia»).

La simulación resultó lenta debido a la incorporación del toolbox Fuzzy de Matlab, y esto debido al alto requerimiento de procesamiento para el análisis de reglas y fuzzificación y defuzzificación de las variables.

RECOMENDACIONES

A partir de los resultados obtenidos con el control propuesto se recomienda implementar el sistema de control, con la garantía que el sistema pueda ser gobernado experimentalmente sin problemas.

Se puede sugerir que para trabajos futuros de sistemas donde los parámetros cambian constantemente se utilicen lógica difusa para la optimización de los controladores PID aplicados a estos robots manipuladores.

Si se desea sólo usar un controlador PID, se recomienda utilizar un algoritmo optimizador para el cálculo de las constantes K_p, K_i y K_d; para así evitar la tarea tediosa e ineficiente de la obtención de estos parámetros.

Se debe siempre tener en cuenta la relación de masa-longitud para cada elemento-eslabón y así no exigir demasiado torque a los eslabones de la base.

Se puede y debe optimizar el controlador Fuzzy, combinándolos con otros algoritmos, formando así controladores Neuro-Fuzzy por ejemplo.

BIBLIOGRAFÍA Y REFERENCIAS:

[1] AGUADO, A. (2000). Temas de Identificación y Control Adaptable. Instituto de Cibernética, Matemática y Física, La Habana, Cuba.

[2] AGUINAGA, A. (2005). Automatización en la Industria, Robótica. Quito: EPN – FIM.

[3] ANGULO, J. (2005). Introducción a la Robótica. Madrid.

[4] ARENA, P., FORTUNA, L. (2002). Analog Cellular Locomotion Control of hexapod robots. IEEE Controls System, December.

[5] BEZDEC, J. (1981). Pattern Recognition with Fuzzy Objective Function Algorithms, Plenum Press, New York.

[6] BORSHEVICH, V., MUSTYATSA, A., OLEINIK, W. (1993). Fuzzy Spectral Analysis of Heart´s Rhythms. Fifth IFSA World Congress, páginas:561-563.

[7] CARREÑO, M., CARDONA, O., CAMPOS, A. (2003). Sistema Experto para la toma de decisiones de habitabilidad y reparabilidad en edificios después de un sismo. Asociación Colombiana de Ingenieria Sísmica, Colombia.

[8] CRAIG, J. (2006). Robótica. Ed. Pearson Educación, 3a edición, México.

[9] DUBOIS, D. and PRADE, H. (1980). Fuzzy Sets and Systems: Theory and Applications, Academic Press. New York.

[10] HOLLAND, J. (1995). Adaptation in Natural and Artificial Systems.

[11] IÑIGO M., R. (2002). Robots Industriales Manipuladores. Barcelona: Ediciones UPC.

[12] JANG, J. and SUN, C. (1995). Neuro-fuzzy modeling and control, Proceedings of the IEEE. March.

[13] KELLY, R., SANTIBAÑEZ, V. (2003). Control de movimiento de robots manipuladores. Ed. Pearson Educación, Madrid.

[14] MATHWORKS. Fuzzy Logic Toolbox de Matlab.

[15] MILLÁN, E. (1999). Ampliación de Ingeniería de Conocimiento. Razonamiento Aproximado. Universidad de Bogotá, Colombia.

[16] MINCHALA, I. (2009). Fundamentos de la Robótica Industrial (Seminario). Quito: EPN – FEPON.

[17] OLLERO, A. (2001). Robótica, Manipuladores y Robots Móviles. Madrid: Marcombo.

[18] OGATA, K. (2003). Ingeniería de Control Moderna. Pearson, 4ta edición.

[19] PARDO, A., PULIDO, D. Técnicas del control en tiempo real para el monitoreo y control de los procesos en el edificio inteligente Simón Bolívar. Facultad de Ciencias Naturales y Tecnológicas, Norte de Santander, Colombia.

[20] SALA, A., PICÓ, J., BONDIA, J. (2000). Tratamiento de la incertidumbre en modelado y control borrosos. Universidad Politécnica de Valencia.

[21] SHABANA, A. Computational Dynamics. Editorial John Wiley.

[22] SPONG, M., HUTCHINSON, S., and VIDYASAGAR M. Robot Modeling and Control. 1era Ed.-pags. 101-107.

[23] SPONG, M., VIDYASAGAR, M. (2006). Robot modeling and control. Ed. Wiley.

[24] SUGENO, M. (1995). Industrial applications of fuzzy control. Elsevier Science Pub. Co.

[25] SUGENO, J., GRIFFIN, M.F. (1993). Fuzzy hierarchical control of an unmanned helicopter. Fifth IFSA World Congress, páginas: 179-182; 1993b.

[26] SUGENO, J., PARK, G. (1993). An Approach to Linguistic Instruction Based Learning and Its Application to Helicopter Flight Control. Fifth IFSA World Congress, páginas: 1082-1085.

[27] TIBADUIZA, D., AMAYA, I., RODRÍGUEZ, S., MEJÍA, N., FLORES, M. (2011). Implementación de un control fuzzy para el control cinemático directo en un robot manipulador. Departamento de Matemática Aplicada III de la Universidad Politécnica de Cataluña, Barcelona, España y la Facultad de Ingeniería Mecatrónica de la Universidad Autónoma de Bucaramanga, Colombia.

[28] TORRES, S., MÉNDEZ, J. (2009). Seguimiento de trayectorias en robots manipuladores revisión de soluciones y nuevas propuestas. Departamento de Ingeniería de Sistemas y Automática y Arquitectura y Tecnología de Computadores de la Universidad de La Laguna – Tenerife, España

[29] TRILLAS, E., ALSINA, C., TERRICABRAS, J. (1995). Introducción a la Lógica Borrosa. Barcelona.

ANEXOS

ANEXO 1: PROGRAMA EN MATLAB PARA OBTENER LOS PARÁMETROS DENAVIT-HANTENBERG DEL ROBOT TIPO PUMA.

```
clear all; close all; clc;

 L{1} = link([ pi/2  0       0    0      0]);

 L{2} = link([ 0     .4318   0    0      0]);

 L{3} = link([-pi/2  .0203   0   .15005  0]);

 L{4} = link([pi/2   0       0   .4318   0]);

 L{5} = link([-pi/2  0       0    0      0]);

 L{6} = link([0      0       0    0      0]);

 L{1}.m = 0;

 L{2}.m = 17.4;

 L{3}.m = 4.8;

 L{4}.m = 0.82;

 L{5}.m = 0.34;

 L{6}.m = .09;

 L{1}.r = [ 0        0      0 ];

 L{2}.r = [ -.3638  .006    .2275];
```

```
L{3}.r = [ -.0203 -.0141  .070];

L{4}.r = [ 0       .019    0];

L{5}.r = [ 0        0      0];

L{6}.r = [ 0        0     .032];

L{1}.I = [ 0       .35      0      0   0   0];

L{2}.I = [.13      .524    .539    0   0   0];

L{3}.I = [.066     .086    .0125   0   0   0];

L{4}.I = [1.8e-3  1.3e-3  1.8e-3   0   0   0];

L{5}.I = [ .3e-3   .4e-3   .3e-3   0   0   0];

L{6}.I = [ .15e-3  .15e-3  .04e-3  0   0   0];

L{1}.Jm =  200e-6;

L{2}.Jm =  200e-6;

L{3}.Jm =  200e-6;

L{4}.Jm = 33e-6;

L{5}.Jm =  33e-6;

L{6}.Jm =  33e-6;

L{1}.G =  -62.6111;

L{2}.G =  107.815;

L{3}.G =  -53.7063;

L{4}.G =  76.0364;
```

```
L{5}.G =  71.923;

L{6}.G =  76.686;

% Friccion viscosa

L{1}.B =   1.48e-3;

L{2}.B =   .817e-3;

L{3}.B =    1.38e-3;

L{4}.B =   71.2e-6;

L{5}.B =   82.6e-6;

L{6}.B =   36.7e-6;

% Friccion de Coulomb

L{1}.Tc = [ .395    -.435];

L{2}.Tc = [ .126    -.071];

L{3}.Tc = [ .132    -.105];

L{4}.Tc = [ 11.2e-3 -16.9e-3];

L{5}.Tc = [ 9.26e-3 -14.5e-3];

L{6}.Tc = [ 3.96e-3 -10.5e-3];

% Posiciones

qz = [0 0 0 0 0 0];

qr = [0 pi/2 -pi/2 0 0 0];

qstretch = [0 0 -pi/2 0 0 0];
```

```
p560 = robot(L, 'Puma 560', 'Unimation', 'params of 8/95');

p560.name = 'Puma 560';

p560.manuf = 'Unimation';

%  p560 =

%

% Puma 560 (6 axis, RRRRRR) [Unimation] <params of 8/95>

%                grav = [0.00 0.00 9.81]                standard D&H parameters

%

%   alpha               A               theta           D               R/P

% 1.570796    0.000000        0.000000        0.000000        R       (std)

% 0.000000    0.431800        0.000000        0.000000        R       (std)

% -1.570796   0.020300        0.000000        0.150050        R       (std)

% 1.570796    0.000000        0.000000        0.431800        R       (std)

% -1.570796   0.000000        0.000000        0.000000        R       (std)

% 0.000000    0.000000        0.000000        0.000000        R       (std)

%
```

ANEXO 2: PROGRAMA EN MATLAB DE LA CINEMÁTICA INVERSA DEL ROBOT TIPO PUMA.

```
clear all; close all; clc;

q = [0 -pi/4 -pi/4 0 pi/8 0];

q = [0 0 -pi/2 0 0 0];

% q = 0 -0.7854 -0.7854 0 0.3927 0

T = fkine(p560, q);

%  T =[ 0.3827    0.0000    0.9239   0.7371

%     -0.0000    1.0000   -0.0000   -0.1501

%     -0.9239   -0.0000    0.3827   -0.3256

%          0         0         0    1.0000]

TA=eye(4);

figure

frame(T,'g',0.5), hold

frame(TA,'r',0.5)

view(29,19)

grid

qi = ikine(p560, T);

qi';

% ans =
```

```
%   -0.0000
%   -0.7854
%   -0.7854
%   -0.0000
%    0.3927
%    0.0000
q; % q = 0  -0.7854  -0.7854  0  0.3927  0
T = fkine(p560, qr);       % qr = [0 pi/2 -pi/2 0 0 0];
qi = ikine(p560, T);
qi'; % ans =
%    -0.0000
%     1.5238
%    -1.4768
%     0.0000
%    -0.0470
%     0.0000
qr; % qr = 0  1.5708 -1.5708  0  0  0
fkine(p560, qi) - fkine(p560, qr);
% ans =
%   1.0e-016 *
```

```
%         0   -0.0000   -0.0000   -0.3123
%    0.0000         0    0.0000         0
%   -0.0000   -0.0000         0         0
%         0         0         0         0
% También pueden computarse las cinemáticas inversas para una trayectoria.
% Si nosotros tomamos un camino en línea recta cartesiana.
t = [0:.056:2];                 % creamos el vector de tiempo
T1 = transl(0.6, -0.5, 0.0);    % define el punto de partida
% T1 =
%     1.0000         0         0    0.6000
%          0    1.0000         0   -0.5000
%          0         0    1.0000         0
%          0         0         0    1.0000
T2 = transl(0.4, 0.5, 0.2);     % destino
% T2 =
%     1.0000         0         0    0.4000
%          0    1.0000         0    0.5000
%          0         0    1.0000    0.2000
%          0         0         0    1.0000
T = ctraj(T1, T2, t/2);         % computa el camino cartesiano
```

```
figure

frame(T1,'g',1), hold

frame(T2,'r',1)

frame(T,'k',1)

view(-30,15)

grid

q = ikine(p560, T);

toc

% tiempo transcurrido =

%    0.9000 seconds

figure

subplot(331), plot(t,q(:,1),'k'), ylabel('Art. 1 (rad)')

subplot(332), plot(t,q(:,2),'k'), ylabel('Art. 2 (rad)')

subplot(333), plot(t,q(:,3),'k'), ylabel('Art. 3 (rad)')

subplot(334), plot(t,q(:,4),'k'), ylabel('Art. 4 (rad)')

subplot(335), plot(t,q(:,5),'k'), ylabel('Art. 5 (rad)')

subplot(336), plot(t,q(:,6),'k'), ylabel('Art. 6 (rad)')

xlabel('Tiempo (s)');

figure

plot(p560, q)
```

ANEXO 3: PROGRAMA EN MATLAB DEL MODELO CINEMÁTICO DEL ROBOT TIPO PUMA DE CUATRO GRADOS DE LIBERTAD EN EL ESPACIO 3D.

```
function puma3d

% GUI kinematic demo for the Puma Robot.

% Robot geometry uses the CAD2MATDEMO code in the Mathworks file exchange

%%

loaddata

InitHome

% Create the push buttons: pos is: [left bottom width height]

demo = uicontrol(fig_1,'String','Demo','callback',@demo_button_press,...

    'Position',[20 5 60 20]);

rnd_demo = uicontrol(fig_1,'String','Random Move','callback',@rnd_demo_button_press,...

    'Position',[100 5 80 20]);

clr_trail = uicontrol(fig_1,'String','Clr Trail','callback',@clr_trail_button_press,...

    'Position',[200 5 60 20]);

%

home = uicontrol(fig_1,'String','Home','callback',@home_button_press,...

    'Position',[280 5 70 20]);

%
```

```
% Kinematics Panel
%
K_p = uipanel(fig_1,...
   'units','pixels',...
   'Position',[20 45 265 200],...
   'Title','Kinematics','FontSize',11);
%
%    Angle    Range              Default Name
%    Theta 1: 320 (-160 to 160)    90       Waist Joint
%    Theta 2: 220 (-110 to 110)   -90       Shoulder Joint
%    Theta 3: 270 (-135 to 135)   -90       Elbow Joint
%    Theta 4: 532 (-266 to 266)     0       Wrist Roll
%    Theta 5: 200 (-100 to 100)     0       Wrist Bend
%    Theta 6: 532 (-266 to 266)     0       Wrist Swivel
t1_home = 90; % offset to define the "home" position as UP.
t2_home = -90;
t3_home = -90;
LD = 105; % Left, used to set the GUI.
HT = 18;  % Height
BT = 156; % Bottom
```

```
%%  GUI buttons for Theta 1.  pos is: [left bottom width height]
t1_slider = uicontrol(K_p,'style','slider',...
    'Max',160,'Min',-160,'Value',0,...
    'SliderStep',[0.05 0.2],...
    'callback',@t1_slider_button_press,...
    'Position',[LD BT 120 HT]);
t1_min = uicontrol(K_p,'style','text',...
    'String','-160',...
    'Position',[LD-30 BT+1 25 HT-4]); % L, from bottom, W, H
t1_max = uicontrol(K_p,'style','text',...
    'String','+160',...
    'Position',[LD+125 BT+1 25 HT-4]); % L, B, W, H
t1_text = uibutton(K_p,'style','text',...  % Nice program Doug. Need this
    'String','\theta_1',...               % due to no TeX in uicontrols.
    'Position',[LD-100 BT 20 HT]); % L, B, W, H
% t1_text = uicontrol(K_p,'style','text',... % when matlab fixes uicontrol
%     'String','t1',...                    % for TeX, then I can use this.
%     'Position',[LD-100 BT 20 HT]); % L, B, W, H
t1_edit = uicontrol(K_p,'style','edit',...
    'String',0,...
```

```
    'callback',@t1_edit_button_press,...
    'Position',[LD-75 BT 30 HT]); % L, B, W, H
%
%%  GUI buttons for Theta 2.
BT = 126;   % Bottom
t2_slider = uicontrol(K_p,'style','slider',...
    'Max',115,'Min',-115,'Value',0,...        % Mech. stop limits !
    'SliderStep',[0.05 0.2],...
    'callback',@t2_slider_button_press,...
    'Position',[LD BT 120 HT]);
t2_min = uicontrol(K_p,'style','text',...
    'String','-110',...
    'Position',[LD-30 BT+1 25 HT-4]); % L, from bottom, W, H
t2_max = uicontrol(K_p,'style','text',...
    'String','+110',...
    'Position',[LD+125 BT+1 25 HT-4]); % L, B, W, H
t2_text = uibutton(K_p,'style','text',...
    'String','\theta_2',...
    'Position',[LD-100 BT 20 HT]); % L, B, W, H
t2_edit = uicontrol(K_p,'style','edit',...
```

```
    'String',0,...
    'callback',@t2_edit_button_press,...
    'Position',[LD-75 BT 30 HT]); % L, B, W, H
%
%%  GUI buttons for Theta 3.
BT = 96;   % Bottom
t3_slider = uicontrol(K_p,'style','slider',...
    'Max',135,'Min',-135,'Value',0,...
    'SliderStep',[0.05 0.2],...
    'callback',@t3_slider_button_press,...
    'Position',[LD BT 120 HT]);
t3_min = uicontrol(K_p,'style','text',...
    'String','-135',...
    'Position',[LD-30 BT+1 25 HT-4]); % L, from bottom, W, H
t3_max = uicontrol(K_p,'style','text',...
    'String','+135',...
    'Position',[LD+125 BT+1 25 HT-4]); % L, B, W, H
t3_text = uibutton(K_p,'style','text',...
    'String','\theta_3',...
    'Position',[LD-100 BT 20 HT]); % L, B, W, H
```

```
t3_edit = uicontrol(K_p,'style','edit',...
    'String',0,...
    'callback',@t3_edit_button_press,...
    'Position',[LD-75 BT 30 HT]); % L, B, W, H
%%  GUI buttons for Theta 4.
BT = 66;   % Bottom
t4_slider = uicontrol(K_p,'style','slider',...
    'Max',266,'Min',-266,'Value',0,...
    'SliderStep',[0.05 0.2],...
    'callback',@t4_slider_button_press,...
    'Position',[LD BT 120 HT]);
t4_min = uicontrol(K_p,'style','text',...
    'String','-266',...
    'Position',[LD-30 BT+1 25 HT-4]); % L, from bottom, W, H
t4_max = uicontrol(K_p,'style','text',...
    'String','+266',...
    'Position',[LD+125 BT+1 25 HT-4]); % L, B, W, H
t4_text = uibutton(K_p,'style','text',...
    'String','\theta_4',...
    'Position',[LD-100 BT 20 HT]); % L, B, W, H
```

```
t4_edit = uicontrol(K_p,'style','edit',...
    'String',0,...
    'callback',@t4_edit_button_press,...
    'Position',[LD-75 BT 30 HT]); % L, B, W, H
%%  GUI buttons for Theta 5.
BT = 36;   % Bottom
t5_slider = uicontrol(K_p,'style','slider',...
    'Max',100,'Min',-100,'Value',0,...
    'SliderStep',[0.05 0.2],...
    'callback',@t5_slider_button_press,...
    'Position',[LD BT 120 HT]);
t5_min = uicontrol(K_p,'style','text',...
    'String','-100',...
    'Position',[LD-30 BT+1 25 HT-4]); % L, from bottom, W, H
t5_max = uicontrol(K_p,'style','text',...
    'String','+100',...
    'Position',[LD+125 BT+1 25 HT-4]); % L, B, W, H
t5_text = uibutton(K_p,'style','text',...
    'String','\theta_5',...
    'Position',[LD-100 BT 20 HT]); % L, B, W, H
```

```
t5_edit = uicontrol(K_p,'style','edit',...
    'String',0,...
    'callback',@t5_edit_button_press,...
    'Position',[LD-75 BT 30 HT]); % L, B, W, H
%%  GUI buttons for Theta 6.
BT = 6;   % Bottom
t6_slider = uicontrol(K_p,'style','slider',...
    'Max',266,'Min',-266,'Value',0,...
    'SliderStep',[0.05 0.2],...
    'callback',@t6_slider_button_press,...
    'Position',[LD BT 120 HT]);
t6_min = uicontrol(K_p,'style','text',...
    'String','-266',...
    'Position',[LD-30 BT+1 25 HT-4]); % L, from bottom, W, H
t6_max = uicontrol(K_p,'style','text',...
    'String','+266',...
    'Position',[LD+125 BT+1 25 HT-4]); % L, B, W, H
t6_text = uibutton(K_p,'style','text',...
    'String','\theta_6',...
    'Position',[LD-100 BT 20 HT]); % L, B, W, H
```

```
t6_edit = uicontrol(K_p,'style','edit',...
    'String',0,...
    'callback',@t6_edit_button_press,...
    'Position',[LD-75 BT 30 HT]); % L, B, W, H
%% Slider for Theta 1 motion.
    function t1_slider_button_press(h,dummy)
        slider_value = round(get(h,'Value'));
        set(t1_edit,'string',slider_value);
        T_Old = getappdata(0,'ThetaOld');
        t2old = T_Old(2); t3old = T_Old(3); t4old = T_Old(4);
        t5old = T_Old(5); t6old = T_Old(6);
        pumaANI(slider_value+t1_home,t2old,t3old,t4old,t5old,t6old,10,'n')
    end
%% Slider for Theta 2 motion.
    function t2_slider_button_press(h,dummy)
        slider_value = round(get(h,'Value'));
        set(t2_edit,'string',slider_value);
        T_Old = getappdata(0,'ThetaOld');
        t1old = T_Old(1); t3old = T_Old(3); t4old = T_Old(4);
        t5old = T_Old(5); t6old = T_Old(6);
```

```
        pumaANI(t1old,slider_value+t2_home,t3old,t4old,t5old,t6old,10,'n')
    end
%% Slider for Theta 3 motion.
    function t3_slider_button_press(h,dummy)
        slider_value = round(get(h,'Value'));
        set(t3_edit,'string',slider_value);
        T_Old = getappdata(0,'ThetaOld');
        t1old = T_Old(1); t2old = T_Old(2); t4old = T_Old(4);
        t5old = T_Old(5); t6old = T_Old(6);
        pumaANI(t1old,t2old,slider_value+t3_home,t4old,t5old,t6old,10,'n')
    end
%% Slider for Theta 4 motion.
    function t4_slider_button_press(h,dummy)
        slider_value = round(get(h,'Value'));
        set(t4_edit,'string',slider_value);
        T_Old = getappdata(0,'ThetaOld');
        t1old = T_Old(1); t2old = T_Old(2); t3old = T_Old(3);
        t5old = T_Old(5); t6old = T_Old(6);
        pumaANI(t1old,t2old,t3old,slider_value,t5old,t6old,10,'n')
    end
```

```
%% Slider for Theta 5 motion.
    function t5_slider_button_press(h,dummy)
        slider_value = round(get(h,'Value'));
        set(t5_edit,'string',slider_value);
        T_Old = getappdata(0,'ThetaOld');
        t1old = T_Old(1); t2old = T_Old(2); t3old = T_Old(3);
        t4old = T_Old(4); t6old = T_Old(6);
        pumaANI(t1old,t2old,t3old,t4old,slider_value,t6old,10,'n')
    end
%% Slider for Theta 6 motion.
    function t6_slider_button_press(h,dummy)
        slider_value = round(get(h,'Value'));
        set(t6_edit,'string',slider_value);
        T_Old = getappdata(0,'ThetaOld');
        t1old = T_Old(1); t2old = T_Old(2); t3old = T_Old(3);
        t4old = T_Old(4); t5old = T_Old(5);
        pumaANI(t1old,t2old,t3old,t4old,t5old,slider_value,10,'n')
    end
%% Edit box for Theta 1 motion.
    function t1_edit_button_press(h,dummy)
```

```
        user_entry = check_edit(h,-160,160,0,t1_edit);
        set(t1_slider,'Value',user_entry);  % slider = text box.
        T_Old = getappdata(0,'ThetaOld');   % Current pose
        %
        t2old = T_Old(2); t3old = T_Old(3); t4old = T_Old(4);
        t5old = T_Old(5); t6old = T_Old(6);
        %
        pumaANI(user_entry+t1_home,t2old,t3old,t4old,t5old,t6old,10,'n')
    end
%% Edit box for Theta 2 motion.
    function t2_edit_button_press(h,dummy)
        user_entry = check_edit(h,-110,110,0,t2_edit);
        set(t2_slider,'Value',user_entry);  % slider = text box.
        T_Old = getappdata(0,'ThetaOld');   % Current pose
        %
        t1old = T_Old(1); t3old = T_Old(3); t4old = T_Old(4);
        t5old = T_Old(5); t6old = T_Old(6);
        pumaANI(t1old,user_entry+t2_home,t3old,t4old,t5old,t6old,10,'n')
    end
%% Edit box for Theta 3 motion.
```

```
    function t3_edit_button_press(h,dummy)
        user_entry = check_edit(h,-135,135,0,t3_edit);
        set(t3_slider,'Value',user_entry);  % slider = text box.
        T_Old = getappdata(0,'ThetaOld');   % Current pose
        %
        t1old = T_Old(1); t2old = T_Old(2); t4old = T_Old(4);
        t5old = T_Old(5); t6old = T_Old(6);
        pumaANI(t1old,t2old,user_entry+t3_home,t4old,t5old,t6old,10,'n')
    end
%% Edit box for Theta 4 motion.
    function t4_edit_button_press(h,dummy)
        user_entry = check_edit(h,-266,266,0,t4_edit);
        set(t4_slider,'Value',user_entry);  % slider = text box.
        T_Old = getappdata(0,'ThetaOld');   % Current pose
        %
        t1old = T_Old(1); t2old = T_Old(2); t3old = T_Old(3);
        t5old = T_Old(5); t6old = T_Old(6);
        pumaANI(t1old,t2old,t3old,user_entry,t5old,t6old,10,'n')
    end
%% Edit box for Theta 5 motion.
```

```
    function t5_edit_button_press(h,dummy)
        user_entry = check_edit(h,-100,100,0,t5_edit);
        set(t5_slider,'Value',user_entry);  % slider = text box.
        T_Old = getappdata(0,'ThetaOld');   % Current pose
        %
        t1old = T_Old(1); t2old = T_Old(2); t3old = T_Old(3);
        t4old = T_Old(4); t6old = T_Old(6);
        %
        pumaANI(t1old,t2old,t3old,t4old,user_entry,t6old,10,'n')
    end
%% Edit box for Theta 6 motion.
    function t6_edit_button_press(h,dummy)
        user_entry = check_edit(h,-266,266,0,t6_edit);
        set(t6_slider,'Value',user_entry);  % slider = text box.
        T_Old = getappdata(0,'ThetaOld');   % Current pose
        t1old = T_Old(1); t2old = T_Old(2); t3old = T_Old(3);
        t4old = T_Old(4); t5old = T_Old(5);
        %
        pumaANI(t1old,t2old,t3old,t4old,t5old,user_entry,10,'n')
    end
```

```
%%
   function user_entry = check_edit(h,min_v,max_v,default,h_edit)
      % This function will check the value typed in the text input box
      % against min and max values, and correct errors.
      %
      % h: handle of gui
      % min_v min value to check
      % max_v max value to check
      % default is the default value if user enters non number
      % h_edit is the edit value to update.
      %
      user_entry = str2double(get(h,'string'));
      if isnan(user_entry)
         errordlg(['You must enter a numeric value, defaulting to ',num2str(default),'.'],'Bad Input','modal')
         set(h_edit,'string',default);
         user_entry = default;
      end
      %
      if user_entry < min_v
```

```
        errordlg(['Minimum limit is ',num2str(min_v),' degrees, using ',num2str(min_v),'.'],'Bad Input','modal')
        user_entry = min_v;
        set(h_edit,'string',user_entry);
    end
    if user_entry > max_v
        errordlg(['Maximum limit is ',num2str(max_v),' degrees, using ',num2str(max_v),'.'],'Bad Input','modal')
        user_entry = max_v;
        set(h_edit,'string',user_entry);
    end
  end
%
%% Demo button's callback
  function demo_button_press(h,dummy)
    %
    % disp('pushed demo bottom');
    %       R = 500;
    %       x = 1000;
    n = 2;    % demo ani steps
```

```
num = 30; % home to start, and end to home ani steps
%       j = 1;
%       M = 1000;
for t = 0:.1:7*pi
   Px = 30*t*cos(t);
   Py = 1200-300*t*(t)/(50*pi);
   Pz = 30*t*sin(t);
   [theta1,theta2,theta3,theta4,theta5,theta6] = PumaIK(Px,Py,Pz);
   if t==0 %move to start of demo
      pumaANI(theta1,theta2,theta3-180,0,0,0,num,'n')
   end
   % Theta 4, 5 & 6 are zero due to plotting at wrist origen.
   pumaANI(theta1,theta2,theta3-180,0,0,0,n,'y')
   set(t1_edit,'string',round(theta1)); % Update slider and text.
   set(t1_slider,'Value',round(theta1));
   set(t2_edit,'string',round(theta2));
   set(t2_slider,'Value',round(theta2));
   set(t3_edit,'string',round(theta3-180));
   set(t3_slider,'Value',round(theta3-180));
end
```

```
        gohome
%       pumaANI(90,-90,-90,0,0,0,num,'n')
    end
%
%
%%
    function home_button_press(h,dummy)
        %disp('pushed home bottom');
        gohome
    end
%
%%
    function clr_trail_button_press(h,dummy)
        %disp('pushed clear trail bottom');
        handles = getappdata(0,'patch_h');          %
        Tr = handles(9);
        %
        setappdata(0,'xtrail',0); % used for trail tracking.
        setappdata(0,'ytrail',0); % used for trail tracking.
        setappdata(0,'ztrail',0); % used for trail tracking.
```

```
        %
        set(Tr,'xdata',0,'ydata',0,'zdata',0);
    end
%%
    function rnd_demo_button_press(h, dummy)
        %disp('pushed random demo bottom');
        % a = 10; b = 50; x = a + (b-a) * rand(5)
        %    Angle    Range              Default Name
        %    Theta 1: 320 (-160 to 160)    90       Waist Joint
        %    Theta 2: 220 (-110 to 110)    -90      Shoulder Joint
        %    Theta 3: 270 (-135 to 135)    -90      Elbow Joint
        %    Theta 4: 532 (-266 to 266)    0      Wrist Roll
        %    Theta 5: 200 (-100 to 100)    0      Wrist Bend
        %    Theta 6: 532 (-266 to 266)    0      Wrist Swival
        t1_home = 90; % offsets to define the "home" postition as UP.
        t2_home = -90;
        t3_home = -90;
        theta1 = -160 + 320*rand(1); % offset for home
        theta2 = -110 + 220*rand(1); % in the UP pos.
        theta3 = -135 + 270*rand(1);
```

```
        theta4 = -266 + 532*rand(1);
        theta5 = -100 + 200*rand(1);
        theta6 = -266 + 532*rand(1);
        n = 50;
        pumaANI(theta1+t1_home,theta2+t2_home,theta3+t3_home,theta4,theta5,theta6,n,'y')
        set(t1_edit,'string',round(theta1)); % Update slider and text.
        set(t1_slider,'Value',round(theta1));
        set(t2_edit,'string',round(theta2));
        set(t2_slider,'Value',round(theta2));
        set(t3_edit,'string',round(theta3));
        set(t3_slider,'Value',round(theta3));
        set(t4_edit,'string',round(theta4));
        set(t4_slider,'Value',round(theta4));
        set(t5_edit,'string',round(theta5));
        set(t5_slider,'Value',round(theta5));
        set(t6_edit,'string',round(theta6));
        set(t6_slider,'Value',round(theta6));
    end
        %%
%Here are the functions used for this robot example:
```

```
%
%%
% When called this function will simply initialize a plot of the Puma 762
% robot by plotting it in it's home orientation and setting the current
% angles accordingly.
   function gohome()
      pumaANI(90,-90,-90,0,0,0,20,'n') % show it animate home
      %PumaPOS(90,-90,-90,0,0,0)  %drive it home, no animate.
      set(t1_edit,'string',0);
      set(t1_slider,'Value',0);  %At the home position, so all
      set(t2_edit,'string',0);   %sliders and input boxes = 0.
      set(t2_slider,'Value',0);
      set(t3_edit,'string',0);
      set(t3_slider,'Value',0);
      set(t4_edit,'string',0);
      set(t4_slider,'Value',0);
      set(t5_edit,'string',0);
      set(t5_slider,'Value',0);
      set(t6_edit,'string',0);
      set(t6_slider,'Value',0);
```

```
        setappdata(0,'ThetaOld',[90,-90,-90,0,0,0]);
    end
%%
% This function will load the 3D CAD data.
%
function loaddata
% Loads all the link data from file linksdata.mat.
% This data comes from a Pro/E 3D CAD model and was made with cad2matdemo.m
% from the file exchange.  All link data manually stored in linksdata.mat
[linkdata]=load('linksdata.mat','s1','s2', 's3','s4','s5','s6','s7','A1');
%Place the robot link 'data' in a storage area
setappdata(0,'Link1_data',linkdata.s1);
setappdata(0,'Link2_data',linkdata.s2);
setappdata(0,'Link3_data',linkdata.s3);
setappdata(0,'Link4_data',linkdata.s4);
setappdata(0,'Link5_data',linkdata.s5);
setappdata(0,'Link6_data',linkdata.s6);
setappdata(0,'Link7_data',linkdata.s7);
setappdata(0,'Area_data',linkdata.A1);
end
```

```
%
%%
% Use forward kinematics to place the robot in a specified configuration.
%
   function PumaPOS(theta1,theta2,theta3,theta4,theta5,theta6)
      s1 = getappdata(0,'Link1_data');
      s2 = getappdata(0,'Link2_data');
      s3 = getappdata(0,'Link3_data');
      s4 = getappdata(0,'Link4_data');
      s5 = getappdata(0,'Link5_data');
      s6 = getappdata(0,'Link6_data');
      s7 = getappdata(0,'Link7_data');
      A1 = getappdata(0,'Area_data');
      %
      a2 = 650;
      a3 = 0;
      d3 = 190;
      d4 = 600;
      Px = 5000;
      Py = 5000;
```

```
Pz = 5000;

t1 = theta1;

t2 = theta2;

t3 = theta3 %-180;

t4 = theta4;

t5 = theta5;

t6 = theta6;

%

% Forward Kinematics

T_01 = tmat(0, 0, 0, t1);

T_12 = tmat(-90, 0, 0, t2);

T_23 = tmat(0, a2, d3, t3);

T_34 = tmat(-90, a3, d4, t4);

T_45 = tmat(90, 0, 0, t5);

T_56 = tmat(-90, 0, 0, t6);


%T_01 = T_01;

T_02 = T_01*T_12;

T_03 = T_02*T_23;

T_04 = T_03*T_34;
```

```
T_05 = T_04*T_45;

T_06 = T_05*T_56;

%

Link1 = s1.V1;

Link2 = (T_01*s2.V2')';

Link3 = (T_02*s3.V3')';

Link4 = (T_03*s4.V4')';

Link5 = (T_04*s5.V5')';

Link6 = (T_05*s6.V6')';

Link7 = (T_06*s7.V7')';

handles = getappdata(0,'patch_h');          %

L1 = handles(1);

L2 = handles(2);

L3 = handles(3);

L4 = handles(4);

L5 = handles(5);

L6 = handles(6);

L7 = handles(7);

%

set(L1,'vertices',Link1(:,1:3),'facec', [0.717,0.116,0.123]);
```

```
        set(L1, 'EdgeColor','none');
        set(L2,'vertices',Link2(:,1:3),'facec', [0.216,1,.583]);
        set(L2, 'EdgeColor','none');
        set(L3,'vertices',Link3(:,1:3),'facec', [0.306,0.733,1]);
        set(L3, 'EdgeColor','none');
        set(L4,'vertices',Link4(:,1:3),'facec', [1,0.542,0.493]);
        set(L4, 'EdgeColor','none');
        set(L5,'vertices',Link5(:,1:3),'facec', [0.216,1,.583]);
        set(L5, 'EdgeColor','none');
        set(L6,'vertices',Link6(:,1:3),'facec', [1,1,0.255]);
        set(L6, 'EdgeColor','none');
        set(L7,'vertices',Link7(:,1:3),'facec', [0.306,0.733,1]);
        set(L7, 'EdgeColor','none');
    end
%%
% This function computes the Inverse Kinematics for the Puma 762 robot
% given X,Y,Z coordinates for a point in the workspace. Note: The IK are
% computed for the origin of Coordinate systems 4,5 & 6.
    function [theta1,theta2,theta3,theta4,theta5,theta6] = PumaIK(Px,Py,Pz)
        theta4 = 0;
```

```
theta5 = 0;

theta6 = 0;

sign1 = 1;

sign3 = 1;

nogo = 0;

noplot = 0;

% Because the sqrt term in theta1 & theta3 can be + or - we run through

% all possible combinations (i = 4) and take the first combination that

% satisfies the joint angle constraints.

while nogo == 0;

   for i = 1:1:4

      if i == 1

         sign1 = 1;

         sign3 = 1;

      elseif i == 2

         sign1 = 1;

         sign3 = -1;

      elseif i == 3

         sign1 = -1;

         sign3 = 1;
```

```
else
   sign1 = -1;
   sign3 = -1;
end
a2 = 650;
a3 = 0;
d3 = 190;
d4 = 600;
rho = sqrt(Px^2+Py^2);
phi = atan2(Py,Px);
K = (Px^2+Py^2+Pz^2-a2^2-a3^2-d3^2-d4^2)/(2*a2);
c4 = cos(theta4);
s4 = sin(theta4);
c5 = cos(theta5);
s5 = sin(theta5);
c6 = cos(theta6);
s6 = sin(theta6);
theta1 = (atan2(Py,Px)-atan2(d3,sign1*sqrt(Px^2+Py^2-d3^2)));
c1 = cos(theta1);
s1 = sin(theta1);
```

```
theta3 = (atan2(a3,d4)-atan2(K,sign3*sqrt(a3^2+d4^2-K^2)));

c3 = cos(theta3);

s3 = sin(theta3);

t23 = atan2((-a3-a2*c3)*Pz-(c1*Px+s1*Py)*(d4-a2*s3),(a2*s3-d4)*Pz+(a3+a2*c3)*(c1*Px+s1*Py));

theta2 = (t23 - theta3);

c2 = cos(theta2);

s2 = sin(theta2);

s23 = ((-a3-a2*c3)*Pz+(c1*Px+s1*Py)*(a2*s3-d4))/(Pz^2+(c1*Px+s1*Py)^2);

c23 = ((a2*s3-d4)*Pz+(a3+a2*c3)*(c1*Px+s1*Py))/(Pz^2+(c1*Px+s1*Py)^2);

r13 = -c1*(c23*c4*s5+s23*c5)-s1*s4*s5;

r23 = -s1*(c23*c4*s5+s23*c5)+c1*s4*s5;

r33 = s23*c4*s5 - c23*c5;

theta4 = atan2(-r13*s1+r23*c1,-r13*c1*c23-r23*s1*c23+r33*s23);

r11 = c1*(c23*(c4*c5*c6-s4*s6)-s23*s5*c6)+s1*(s4*c5*c6+c4*s6);

r21 = s1*(c23*(c4*c5*c6-s4*s6)-s23*s5*c6)-c1*(s4*c5*c6+c4*s6);

r31 = -s23*(c4*c5*c6-s4*s6)-c23*s5*c6;

s5 = -(r13*(c1*c23*c4+s1*s4)+r23*(s1*c23*c4-c1*s4)-r33*(s23*c4));

c5 = r13*(-c1*s23)+r23*(-s1*s23)+r33*(-c23);

theta5 = atan2(s5,c5);
```

```
s6 = -r11*(c1*c23*s4-s1*c4)-r21*(s1*c23*s4+c1*c4)+r31*(s23*s4);

c6 = r11*((c1*c23*c4+s1*s4)*c5-c1*s23*s5)+r21*((s1*c23*c4-c1*s4)*c5-s1*s23*s5)-r31*(s23*c4*c5+c23*s5);

theta6 = atan2(s6,c6);

theta1 = theta1*180/pi;

theta2 = theta2*180/pi;

theta3 = theta3*180/pi;

theta4 = theta4*180/pi;

theta5 = theta5*180/pi;

theta6 = theta6*180/pi;

if theta2>=160 && theta2<=180

   theta2 = -theta2;

end

if theta1<=160 && theta1>=-160 && (theta2<=20 && theta2>=-200) && theta3<=45 && theta3>=-225 && theta4<=266 && theta4>=-266 && theta5<=100 && theta5>=-100 && theta6<=266 && theta6>=-266

   nogo = 1;

   theta3 = theta3+180;

   break

end

if i == 4 && nogo == 0
```

```
                h = errordlg('Point unreachable due to joint angle constraints.','JOINT ERROR');
                waitfor(h);
                nogo = 1;
                noplot = 1;
                break
            end
        end
    end
%
%%
    function pumaANI(theta1,theta2,theta3,theta4,theta5,theta6,n,trail)
        % This function will animate the Puma 762 robot given joint angles.
        % n is number of steps for the animation
        % trail is 'y' or 'n' (n = anything else) for leaving a trail.
        %
        %disp('in animate');
        a2 = 650; %D-H paramaters
        a3 = 0;
        d3 = 190;
```

```
d4 = 600;

% Err2 = 0;

%

ThetaOld = getappdata(0,'ThetaOld');

%

theta1old = ThetaOld(1);

theta2old = ThetaOld(2);

theta3old = ThetaOld(3);

theta4old = ThetaOld(4);

theta5old = ThetaOld(5);

theta6old = ThetaOld(6);

%

t1 = linspace(theta1old,theta1,n);

t2 = linspace(theta2old,theta2,n);

t3 = linspace(theta3old,theta3,n);% -180;

t4 = linspace(theta4old,theta4,n);

t5 = linspace(theta5old,theta5,n);

t6 = linspace(theta6old,theta6,n);

n = length(t1);

for i = 2:1:n
```

```
        % Forward Kinematics
        %
        T_01 = tmat(0, 0, 0, t1(i));
        T_12 = tmat(-90, 0, 0, t2(i));
        T_23 = tmat(0, a2, d3, t3(i));
        T_34 = tmat(-90, a3, d4, t4(i));
        T_45 = tmat(90, 0, 0, t5(i));
        T_56 = tmat(-90, 0, 0, t6(i));
%
%           %     T_67 = [    1           0      0 0
%           %                 0           1      0 0
%           %                 0           0      1 188
%           %                 0           0      0 1];
        %T_01 = T_01;  % it is, but don't need to say so.
        T_02 = T_01*T_12;
        T_03 = T_02*T_23;
        T_04 = T_03*T_34;
        T_05 = T_04*T_45;
        T_06 = T_05*T_56;
        %     T_07 = T_06*T_67;
```

```
%

s1 = getappdata(0,'Link1_data');

s2 = getappdata(0,'Link2_data');

s3 = getappdata(0,'Link3_data');

s4 = getappdata(0,'Link4_data');

s5 = getappdata(0,'Link5_data');

s6 = getappdata(0,'Link6_data');

s7 = getappdata(0,'Link7_data');

%A1 = getappdata(0,'Area_data');

Link1 = s1.V1;

Link2 = (T_01*s2.V2')';

Link3 = (T_02*s3.V3')';

Link4 = (T_03*s4.V4')';

Link5 = (T_04*s5.V5')';

Link6 = (T_05*s6.V6')';

Link7 = (T_06*s7.V7')';

%    Tool = T_07;

%    if sqrt(Tool(1,4)^2+Tool(2,4)^2)<514

%        Err2 = 1;

%        break
```

```
%     end
%
handles = getappdata(0,'patch_h');          %
L1 = handles(1);
L2 = handles(2);
L3 = handles(3);
L4 = handles(4);
L5 = handles(5);
L6 = handles(6);
L7 = handles(7);
Tr = handles(9);
%
set(L1,'vertices',Link1(:,1:3),'facec', [0.717,0.116,0.123]);
set(L1, 'EdgeColor','none');
set(L2,'vertices',Link2(:,1:3),'facec', [0.216,1,.583]);
set(L2, 'EdgeColor','none');
set(L3,'vertices',Link3(:,1:3),'facec', [0.306,0.733,1]);
set(L3, 'EdgeColor','none');
set(L4,'vertices',Link4(:,1:3),'facec', [1,0.542,0.493]);
set(L4, 'EdgeColor','none');
```

```
set(L5,'vertices',Link5(:,1:3),'facec', [0.216,1,.583]);

set(L5, 'EdgeColor','none');

set(L6,'vertices',Link6(:,1:3),'facec', [1,1,0.255]);

set(L6, 'EdgeColor','none');

set(L7,'vertices',Link7(:,1:3),'facec', [0.306,0.733,1]);

set(L7, 'EdgeColor','none');

% store trail in appdata

if trail == 'y'

   x_trail = getappdata(0,'xtrail');

   y_trail = getappdata(0,'ytrail');

   z_trail = getappdata(0,'ztrail');

   %

   xdata = [x_trail T_04(1,4)];

   ydata = [y_trail T_04(2,4)];

   zdata = [z_trail T_04(3,4)];

   %

   setappdata(0,'xtrail',xdata); % used for trail tracking.

   setappdata(0,'ytrail',ydata); % used for trail tracking.

   setappdata(0,'ztrail',zdata); % used for trail tracking.

   %
```

```
            set(Tr,'xdata',xdata,'ydata',ydata,'zdata',zdata);
        end
        drawnow
    end
    setappdata(0,'ThetaOld',[theta1,theta2,theta3,theta4,theta5,theta6]);
  end
%%
%
%
%%
        function InitHome
    % Use forward kinematics to place the robot in a specified
    % configuration.
    % Figure setup data, create a new figure for the GUI
    set(0,'Units','pixels')
    dim = get(0,'ScreenSize');
    fig_1 = figure('doublebuffer','on','Position',[0,35,dim(3)-200,dim(4)-110],...
        'MenuBar','none','Name',' 3D Puma Robot Graphical Demo',...
        'NumberTitle','off','CloseRequestFcn',@del_app);
    hold on;
```

```
%light('Position',[-1 0 0]);
light                          % add a default light
daspect([1 1 1])               % Setting the aspect ratio
view(135,25)
xlabel('X'),ylabel('Y'),zlabel('Z');
title('WWU Robotics Lab PUMA 762');
axis([-1500 1500 -1500 1500 -1120 1500]);
plot3([-1500,1500],[-1500,-1500],[-1120,-1120],'k')
plot3([-1500,-1500],[-1500,1500],[-1120,-1120],'k')
plot3([-1500,-1500],[-1500,-1500],[-1120,1500],'k')
plot3([-1500,-1500],[1500,1500],[-1120,1500],'k')
plot3([-1500,1500],[-1500,-1500],[1500,1500],'k')
plot3([-1500,-1500],[-1500,1500],[1500,1500],'k')
s1 = getappdata(0,'Link1_data');
s2 = getappdata(0,'Link2_data');
s3 = getappdata(0,'Link3_data');
s4 = getappdata(0,'Link4_data');
s5 = getappdata(0,'Link5_data');
s6 = getappdata(0,'Link6_data');
s7 = getappdata(0,'Link7_data');
```

```
A1 = getappdata(0,'Area_data');

%

a2 = 650;

a3 = 0;

d3 = 190;

d4 = 600;

Px = 5000;

Py = 5000;

Pz = 5000;

%The 'home' position, for init.

t1 = 90;

t2 = -90;

t3 = -90;

t4 = 0;

t5 = 0;

t6 = 0;

    % Forward Kinematics

T_01 = tmat(0, 0, 0, t1);

T_12 = tmat(-90, 0, 0, t2);

T_23 = tmat(0, a2, d3, t3);
```

```
T_34 = tmat(-90, a3, d4, t4);

T_45 = tmat(90, 0, 0, t5);

T_56 = tmat(-90, 0, 0, t6);

% Each link fram to base frame transformation

T_02 = T_01*T_12;

T_03 = T_02*T_23;

T_04 = T_03*T_34;

T_05 = T_04*T_45;

T_06 = T_05*T_56;

 % Actual vertex data of robot links

Link1 = s1.V1;

Link2 = (T_01*s2.V2')';

Link3 = (T_02*s3.V3')';

Link4 = (T_03*s4.V4')';

Link5 = (T_04*s5.V5')';

Link6 = (T_05*s6.V6')';

Link7 = (T_06*s7.V7')';

% points are no fun to watch, make it look 3d.

L1 = patch('faces', s1.F1, 'vertices' ,Link1(:,1:3));

L2 = patch('faces', s2.F2, 'vertices' ,Link2(:,1:3));
```

```
L3 = patch('faces', s3.F3, 'vertices' ,Link3(:,1:3));

L4 = patch('faces', s4.F4, 'vertices' ,Link4(:,1:3));

L5 = patch('faces', s5.F5, 'vertices' ,Link5(:,1:3));

L6 = patch('faces', s6.F6, 'vertices' ,Link6(:,1:3));

L7 = patch('faces', s7.F7, 'vertices' ,Link7(:,1:3));

A1 = patch('faces', A1.Fa, 'vertices' ,A1.Va(:,1:3));

Tr = plot3(0,0,0,'b.'); % holder for trail paths

%

setappdata(0,'patch_h',[L1,L2,L3,L4,L5,L6,L7,A1,Tr])

%

setappdata(0,'xtrail',0); % used for trail tracking.

setappdata(0,'ytrail',0); % used for trail tracking.

setappdata(0,'ztrail',0); % used for trail tracking.

%

set(L1, 'facec', [0.717,0.116,0.123]);

set(L1, 'EdgeColor','none');

set(L2, 'facec', [0.216,1,.583]);

set(L2, 'EdgeColor','none');

set(L3, 'facec', [0.306,0.733,1]);

set(L3, 'EdgeColor','none');
```

```
        set(L4, 'facec', [1,0.542,0.493]);
        set(L4, 'EdgeColor','none');
        set(L5, 'facec', [0.216,1,.583]);
        set(L5, 'EdgeColor','none');
        set(L6, 'facec', [1,1,0.255]);
        set(L6, 'EdgeColor','none');
        set(L7, 'facec', [0.306,0.733,1]);
        set(L7, 'EdgeColor','none');
        set(A1, 'facec', [.8,.8,.8],'FaceAlpha',.25);
        set(A1, 'EdgeColor','none');
        %
        setappdata(0,'ThetaOld',[90,-90,-90,0,0,0]);
        %
    end
%%
    function T = tmat(alpha, a, d, theta)
        % tmat(alpha, a, d, theta) (T-Matrix used in Robotics)
        % The homogeneous transformation called the "T-MATRIX"
        % as used in the Kinematic Equations for robotic type
        % systems (or equivalent).
```

```
        %
        % This is equation 3.6 in Craig's "Introduction to Robotics."
        % alpha, a, d, theta are the Denavit-Hartenberg parameters.
        %
        % (NOTE: ALL ANGLES MUST BE IN DEGREES.)
        %
        alpha = alpha*pi/180;    %Note: alpha is in radians.
        theta = theta*pi/180;    %Note: theta is in radians.
        c = cos(theta);
        s = sin(theta);
        ca = cos(alpha);
        sa = sin(alpha);
        T = [c -s 0 a; s*ca c*ca -sa -sa*d; s*sa c*sa ca ca*d; 0 0 0 1];
    end
%%
    function del_app(varargin)
        %This is the main figure window close function, to remove any
        % app data that may be left due to using it for geometry.
        %CloseRequestFcn
        % here is the data to remove:
```

```
%     Link1_data: [1x1 struct]
%     Link2_data: [1x1 struct]
%     Link3_data: [1x1 struct]
%     Link4_data: [1x1 struct]
%     Link5_data: [1x1 struct]
%     Link6_data: [1x1 struct]
%     Link7_data: [1x1 struct]
%      Area_data: [1x1 struct]
%        patch_h: [1x9 double]
%       ThetaOld: [90 -182 -90 -106 80 106]
%         xtrail: 0
%         ytrail: 0
%         ztrail: 0
% Now remove them.
rmappdata(0,'Link1_data');
rmappdata(0,'Link2_data');
rmappdata(0,'Link3_data');
rmappdata(0,'Link4_data');
rmappdata(0,'Link5_data');
rmappdata(0,'Link6_data');
```

```
        rmappdata(0,'Link7_data');
        rmappdata(0,'ThetaOld');
        rmappdata(0,'Area_data');
        rmappdata(0,'patch_h');
        rmappdata(0,'xtrail');
        rmappdata(0,'ytrail');
        rmappdata(0,'ztrail');
        delete(fig_1);
    end
%%
    function [hout,ax_out] = uibutton(varargin)
        %uibutton: Create pushbutton with more flexible labeling than uicontrol.
        % Usage:
        %   uibutton accepts all the same arguments as uicontrol except for the
        %   following property changes:
        %
        %    Property      Values
        %    -----------   -------------------------------------------------------
        %    Style         'pushbutton', 'togglebutton' or 'text', default =
        %                  'pushbutton'.
```

```
%     String        Same as for text() including cell array of strings and
%                   TeX or LaTeX interpretation.
%     Interpreter   'tex', 'latex' or 'none', default = default for text()
%
% Syntax:
%   handle = uibutton('PropertyName',PropertyValue,...)
%   handle = uibutton(parent,'PropertyName',PropertyValue,...)
%   [text_obj,axes_handle] = uibutton('Style','text',...
%       'PropertyName',PropertyValue,...)
%
% uibutton creates a temporary axes and text object containing the text to
% be displayed, captures the axes as an image, deletes the axes and then
% displays the image on the uicontrol.  The handle to the uicontrol is
% returned.  If you pass in a handle to an existing uicontol as the first
% argument then uibutton will use that uicontrol and not create a new one.
%
% If the Style is set to 'text' then the axes object is not deleted and the
% text object handle is returned (as well as the handle to the axes in a
% second output argument).
%
```

```
% See also UICONTROL.

% Version: 1.6, 20 April 2006

% Author:  Douglas M. Schwarz

% Email:   dmschwarz=ieee*org, dmschwarz=urgrad*rochester*edu

% Real_email = regexprep(Email,{'=','*'},{'@','.'})

% Detect if first argument is a uicontrol handle.

keep_handle = false;

if nargin > 0

   h = varargin{1};

   if isscalar(h) && ishandle(h) && strcmp(get(h,'Type'),'uicontrol')

      keep_handle = true;

      varargin(1) = [];

   end

end

% Parse arguments looking for 'Interpreter' property.  If found, note its

% value and then remove it from where it was found.

interp_value = get(0,'DefaultTextInterpreter');

arg = 1;

remove = [];

while arg <= length(varargin)
```

```
v = varargin{arg};
if isstruct(v)
   fn = fieldnames(v);
   for i = 1:length(fn)
      if strncmpi(fn{i},'interpreter',length(fn{i}))
         interp_value = v.(fn{i});
         v = rmfield(v,fn{i});
      end
   end
   varargin{arg} = v;
   arg = arg + 1;
elseif ischar(v)
   if strncmpi(v,'interpreter',length(v))
      interp_value = varargin{arg+1};
      remove = [remove,arg,arg+1];
   end
   arg = arg + 2;
elseif arg == 1 && isscalar(v) && ishandle(v) && ...
      any(strcmp(get(h,'Type'),{'figure','uipanel'}))
   arg = arg + 1;
```

```
    else
        error('Invalid property or uicontrol parent.')
    end
end
varargin(remove) = [];
% Create uicontrol, get its properties then hide it.
if keep_handle
    set(h,varargin{:})
else
    h = uicontrol(varargin{:});
end
s = get(h);
if ~any(strcmp(s.Style,{'pushbutton','togglebutton','text'}))
    delete(h)
    error('"Style" must be pushbutton, togglebutton or text.')
end
set(h,'Visible','off')
% Create axes.
parent = get(h,'Parent');
ax = axes('Parent',parent,...
```

```
    'Units',s.Units,...
    'Position',s.Position,...
    'XTick',[],'YTick',[],...
    'XColor',s.BackgroundColor,...
    'YColor',s.BackgroundColor,...
    'Box','on',...
    'Color',s.BackgroundColor);
% Adjust size of axes for best appearance.
set(ax,'Units','pixels')
pos = round(get(ax,'Position'));
if strcmp(s.Style,'text')
    set(ax,'Position',pos + [0 1 -1 -1])
else
    set(ax,'Position',pos + [4 4 -8 -8])
end
switch s.HorizontalAlignment
    case 'left'
        x = 0.0;
    case 'center'
        x = 0.5;
```

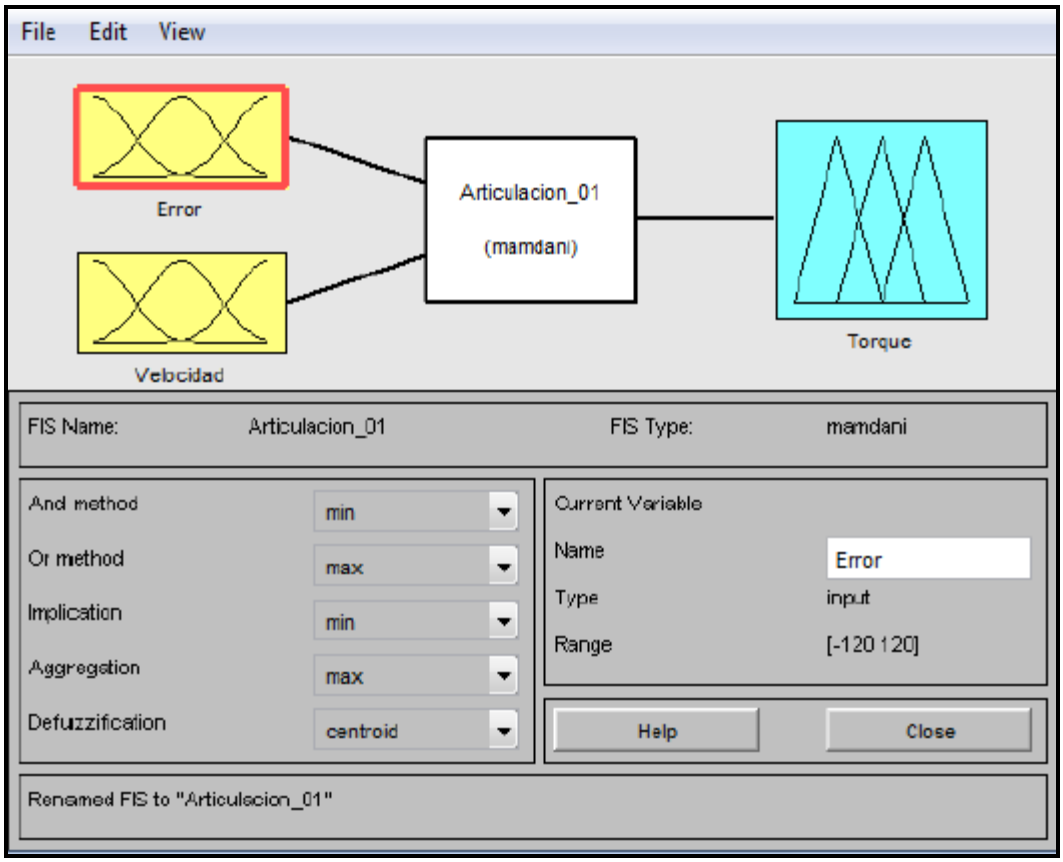
```
    case 'right'
        x = 1;
end
% Create text object.
text_obj = text('Parent',ax,...
    'Position',[x,0.5],...
    'String',s.String,...
    'Interpreter',interp_value,...
    'HorizontalAlignment',s.HorizontalAlignment,...
    'VerticalAlignment','middle',...
    'FontName',s.FontName,...
    'FontSize',s.FontSize,...
    'FontAngle',s.FontAngle,...
    'FontWeight',s.FontWeight,...
    'Color',s.ForegroundColor);
% If we are creating something that looks like a text uicontrol then we're
% all done and we return the text object and axes handles rather than a
% uicontrol handle.
if strcmp(s.Style,'text')
    delete(h)
```

```
    if nargout
        hout = text_obj;
        ax_out = ax;
    end
    return
end
% Capture image of axes and then delete the axes.
frame = getframe(ax);
delete(ax)
% Build RGB image, set background pixels to NaN and put it in 'CData' for
% the uicontrol.
if isempty(frame.colormap)
    rgb = frame.cdata;
else
    rgb = reshape(frame.colormap(frame.cdata,:),[pos([4,3]),3]);
end
size_rgb = size(rgb);
rgb = double(rgb)/255;
back = repmat(permute(s.BackgroundColor,[1 3 2]),size_rgb(1:2));
isback = all(rgb == back,3);
```

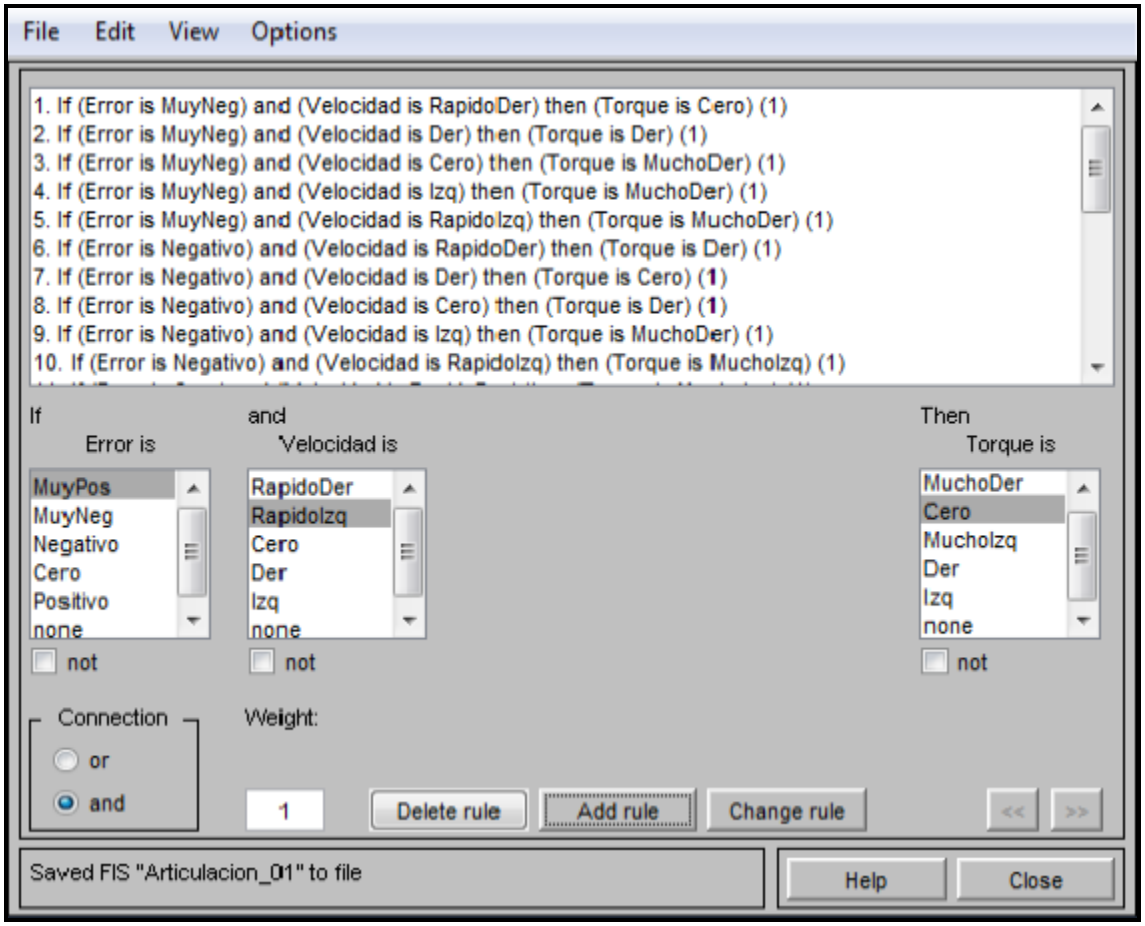
```
        rgb(repmat(isback,[1 1 3])) = NaN;
        set(h,'CData',rgb,'String','','Visible',s.Visible)
        % Assign output argument if necessary.
        if nargout
            hout = h;
        end
%%
    end
end
% Finally
```

ANEXO 4: CONFIGURACIÓN DEL CONTROLADOR DIFUSO PARA CADA UNA DE LAS ARTICULACIONES DEL ROBOT TIPO PUMA.

Configuración del controlador difuso de la articulación 1 en Fuzzy Logic Toolbox.



Definición de las variables de la articulación 01.



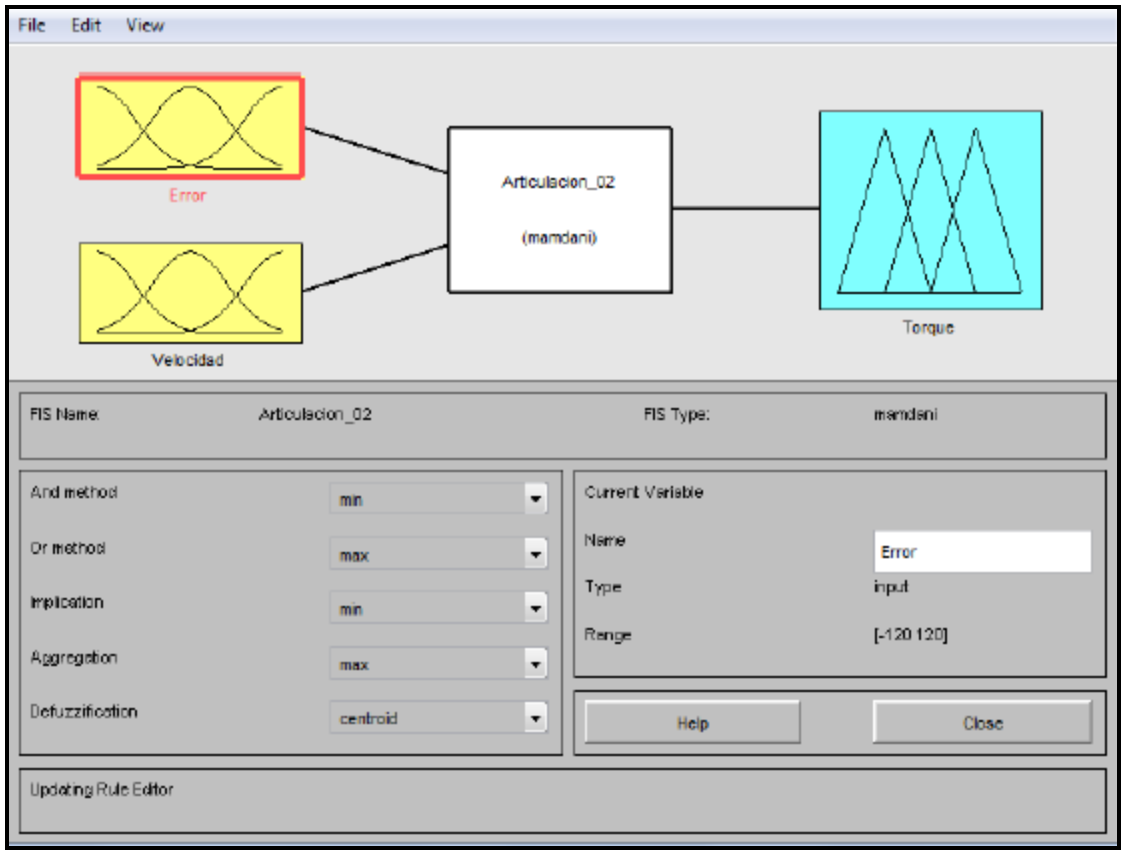
Ventana Rule Editor de la articulación 01.

```
[System]

Name='Articulacion_01'

Type='mamdani'

Version=2.0

NumInputs=2

NumOutputs=1

NumRules=25

AndMethod='min'

OrMethod='max'

ImpMethod='min'

AggMethod='max'

DefuzzMethod='centroid'

[Input1]

Name='Error'

Range=[-120 120]

NumMFs=5

MF1='MuyPos':'trapmf',[50 90 120 120]

MF2='MuyNeg':'trapmf',[-120 -120 -90 -50]

MF3='Negativo':'trimf',[-90 -45 0]

MF4='Cero':'trimf',[-20 0 20]

MF5='Positivo':'trimf',[0 45 90]
```

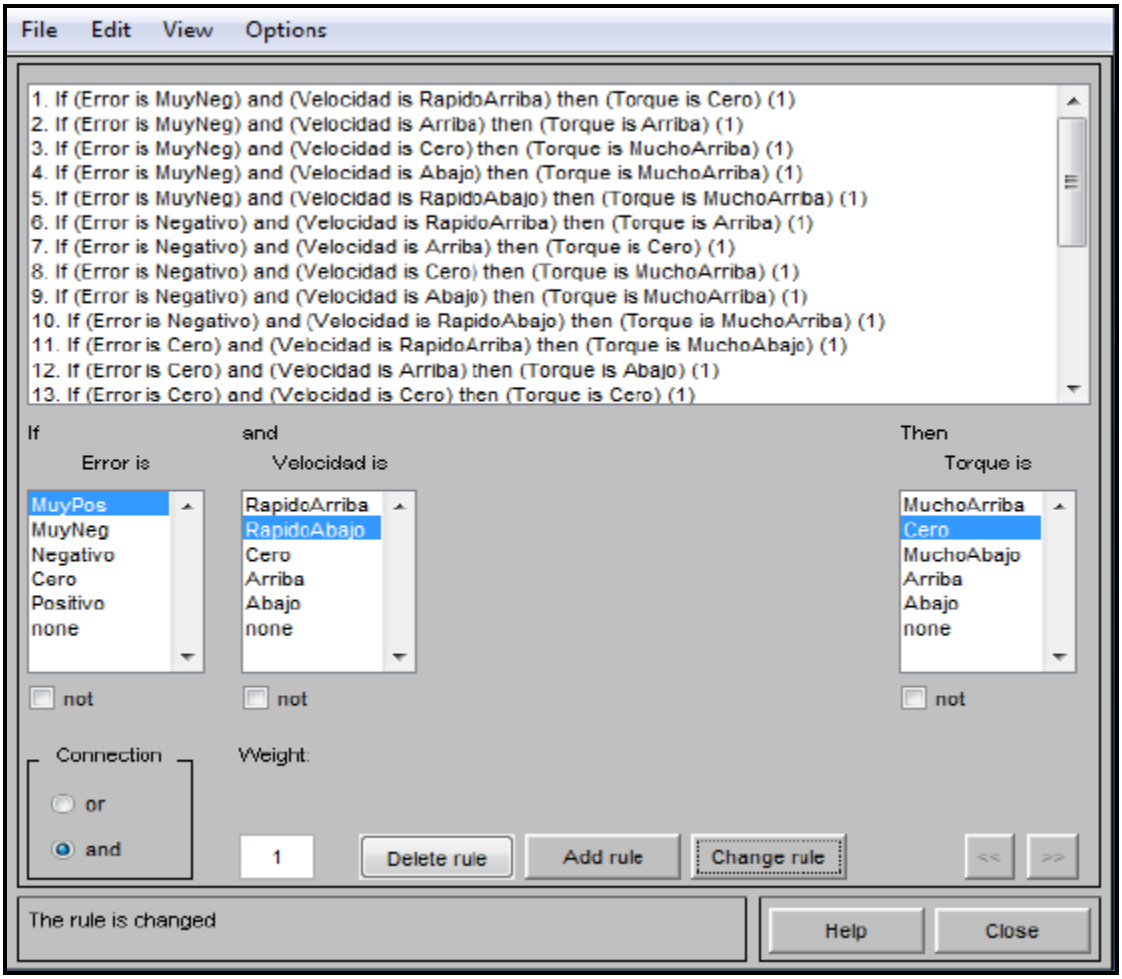
```
[Input2]

Name='Velocidad'

Range=[-50 50]

NumMFs=5

MF1='RapidoDer':'trapmf',[-50 -50 -35 -22]

MF2='RapidoIzq':'trapmf',[22 36 50 50]

MF3='Cero':'trimf',[-14 0 14]

MF4='Der':'trapmf',[-38 -24 -14 0]

MF5='Izq':'trapmf',[0 14 24 38]

[Output1]

Name='Torque'

Range=[-2 2]

NumMFs=5

MF1='MuchoDer':'trapmf',[-2 -2 -1 -0.5]

MF2='Cero':'trimf',[-0.5 0 0.5]

MF3='MuchoIzq':'trapmf',[0.5 1 2 2]

MF4='Der':'trimf',[-1 -0.5 0]

MF5='Izq':'trimf',[0 0.5 1]

[Rules]

2 1, 2 (1): 1

2 4, 4 (1): 1

2 3, 1 (1): 1
```

```
2 5, 1 (1): 1

2 2, 1 (1): 1

3 1, 4 (1): 1

3 4, 2 (1): 1

3 3, 4 (1): 1

3 5, 1 (1): 1

3 2, 3 (1): 1

4 1, 3 (1): 1

4 4, 5 (1): 1

4 3, 2 (1): 1

4 5, 4 (1): 1

4 2, 1 (1): 1

5 1, 3 (1): 1

5 4, 3 (1): 1

5 3, 5 (1): 1

5 5, 2 (1): 1

5 2, 5 (1): 1

1 1, 3 (1): 1

1 4, 3 (1): 1

1 3, 3 (1): 1

1 5, 5 (1): 1

1 2, 2 (1): 1
```

Configuración del controlador difuso de la articulación 02 en Fuzzy Logic Toolbox.



Definición de las variables de la articulación 02.



Ventana Rule Editor de la articulación 02.

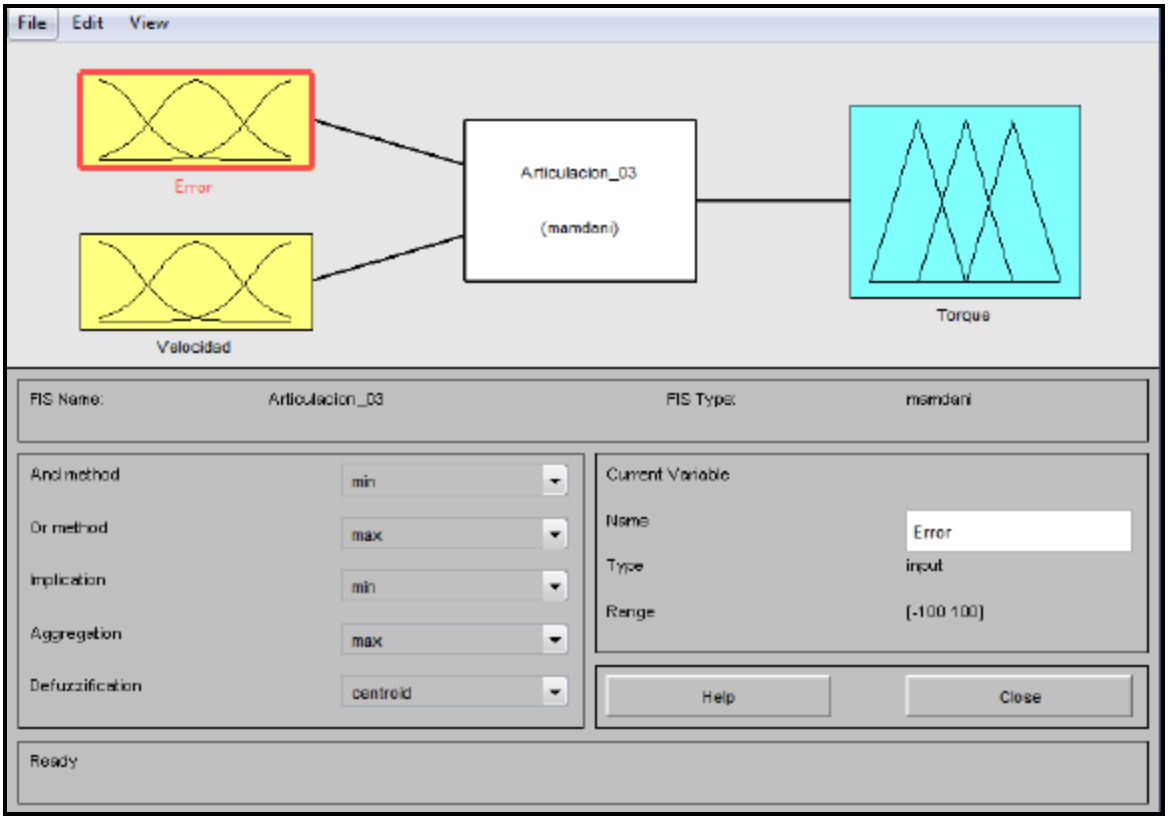
```
[System]

Name='Articulacion_02'

Type='mamdani'

Version=2.0

NumInputs=2

NumOutputs=1

NumRules=25

AndMethod='min'
```

```
OrMethod='max'

ImpMethod='min'

AggMethod='max'

DefuzzMethod='centroid'

[Input1]

Name='Error'

Range=[-100 100]

NumMFs=5

MF1='MuyPos':'trapmf',[40 75 100 100]

MF2='MuyNeg':'trapmf',[-100 -100 -75 -40]

MF3='Negativo':'trimf',[-75 -37.5 0]

MF4='Cero':'trimf',[-16 0 16]

MF5='Positivo':'trimf',[0 37.5 75]

[Input2]

Name='Velocidad'

Range=[-50 50]

NumMFs=5

MF1='RapidoDer':'trapmf',[-50 -50 -35 -22]

MF2='RapidoIzq':'trapmf',[22 36 50 50]

MF3='Cero':'trimf',[-14 0 14]

MF4='Der':'trapmf',[-38 -24 -14 0]

MF5='Izq':'trapmf',[0 14 24 38]
```

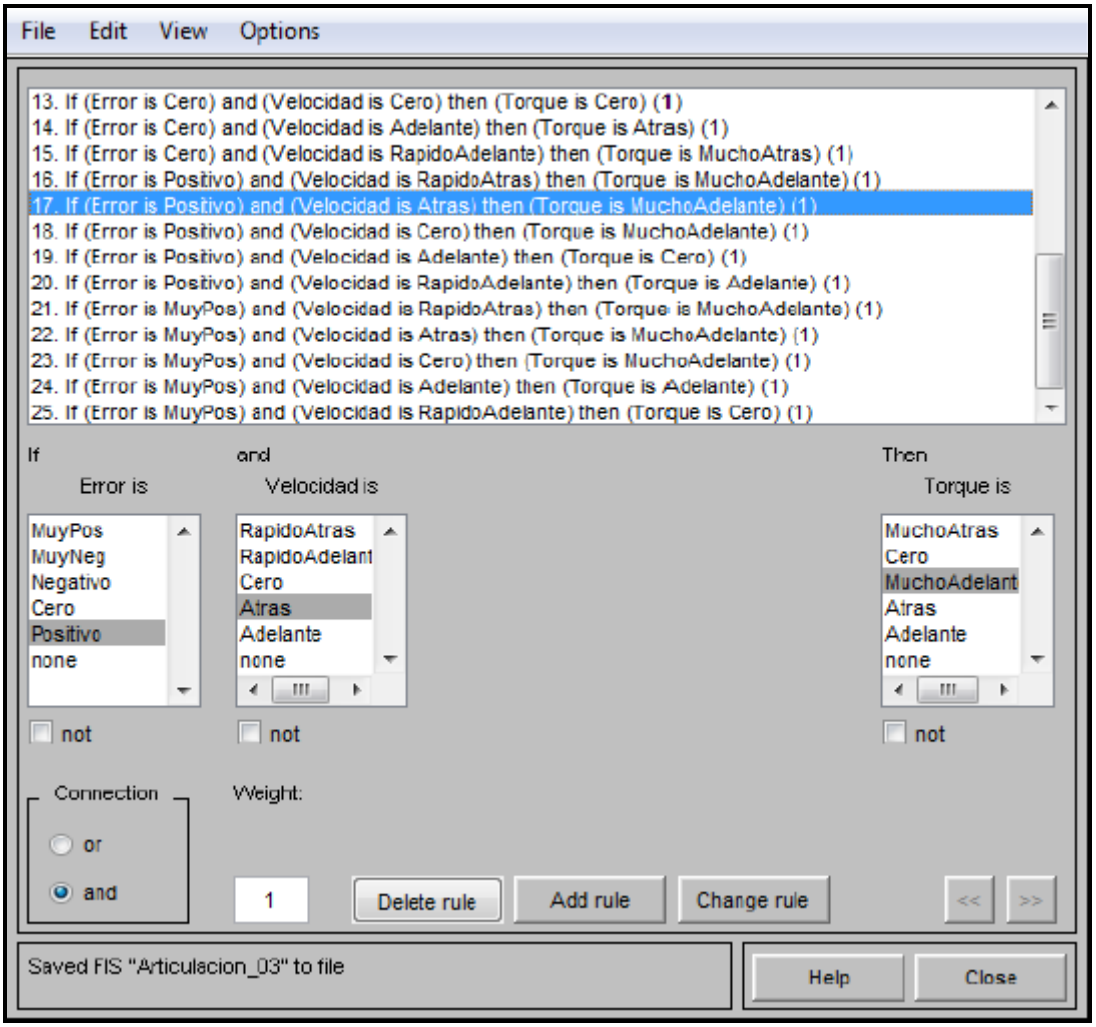
```
[Output1]

Name='Torque'

Range=[-6 6]

NumMFs=5

MF1='MuchoDer':'trapmf',[-6 -6 -3 -1.5]

MF2='Cero':'trimf',[-1.5 0 1.5]

MF3='MuchoIzq':'trapmf',[1.5 3 6 6]

MF4='Der':'trimf',[-3 -1.5 0]

MF5='Izq':'trimf',[0 1.5 3]

[Rules]

2 1, 2 (1): 1

2 4, 4 (1): 1

2 3, 1 (1): 1

2 5, 1 (1): 1

2 2, 1 (1): 1

3 1, 4 (1): 1

3 4, 2 (1): 1

3 3, 4 (1): 1

3 5, 1 (1): 1

3 2, 3 (1): 1

4 1, 3 (1): 1

4 4, 5 (1): 1
```

```
4 3, 2 (1): 1

4 5, 4 (1): 1

4 2, 1 (1): 1

5 1, 3 (1): 1

5 4, 3 (1): 1

5 3, 5 (1): 1

5 5, 2 (1): 1

5 2, 5 (1): 1

1 1, 3 (1): 1

1 4, 3 (1): 1

1 3, 3 (1): 1

1 5, 5 (1): 1

1 2, 2 (1): 1
```

Configuración del controlador difuso de la articulación 03 en Fuzzy Logic Toolbox.



Definición de las variables de la articulación 03.



Ventana Rule Editor de la articulación 03.

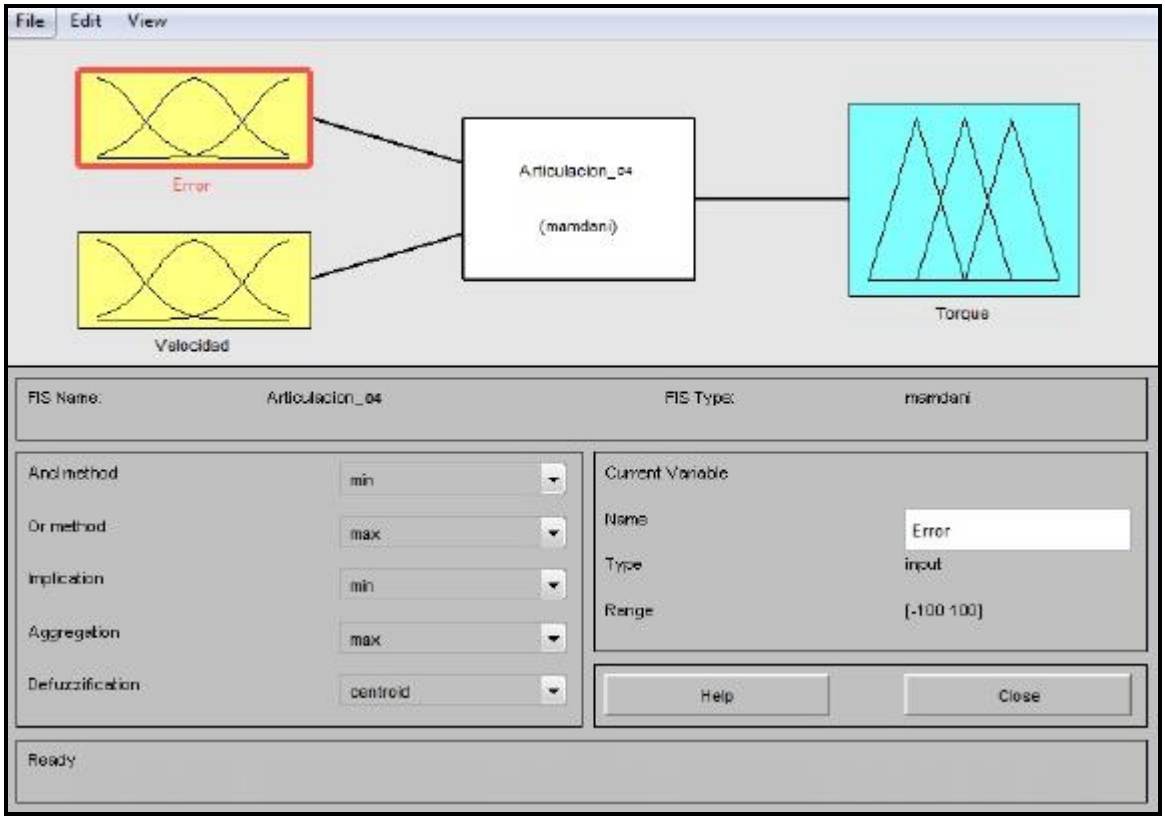
```
[System]

Name='Articulacion_03'

Type='mamdani'

Version=2.0

NumInputs=2

NumOutputs=1

NumRules=25
```

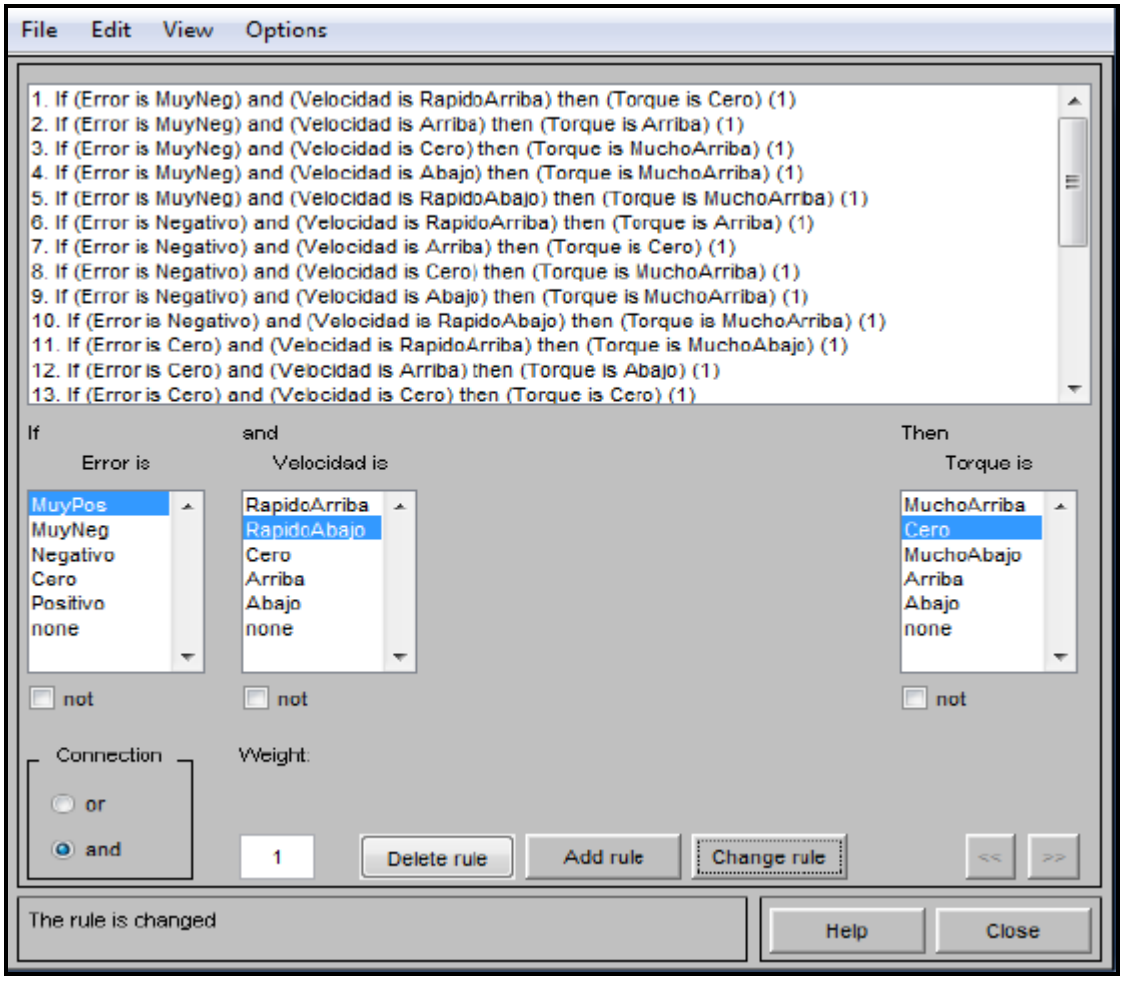
```
AndMethod='min'

OrMethod='max'

ImpMethod='min'

AggMethod='max'

DefuzzMethod='centroid'

[Input1]

Name='Error'

Range=[-120 120]

NumMFs=5

MF1='MuyPos':'trapmf',[48 90 120 120]

MF2='MuyNeg':'trapmf',[-120 -120 -90 -48]

MF3='Negativo':'trimf',[-90 -45 0]

MF4='Cero':'trimf',[-19 0 19]

MF5='Positivo':'trimf',[0 45 90]

[Input2]

Name='Velocidad'

Range=[-50 50]

NumMFs=5

MF1='RapidoAtras':'trapmf',[-50 -50 -35 -22]

MF2='RapidoAdelante':'trapmf',[22 36 50 50]

MF3='Cero':'trimf',[-14 0 14]

MF4='Atras':'trapmf',[-38 -24 -14 0]
```

```
MF5='Adelante':'trapmf',[0 14 24 38]

[Output1]

Name='Torque'

Range=[-2 2]

NumMFs=5

MF1='MuchoAtras':'trapmf',[-2 -2 -1 -0.5]

MF2='Cero':'trimf',[-0.5 0 0.5]

MF3='MuchoAdelante':'trapmf',[0.5 1 2 2]

MF4='Atras':'trimf',[-1 -0.5 0]

MF5='Adelante':'trimf',[0 0.5 1]

[Rules]

2 1, 2 (1): 1

2 4, 4 (1): 1

2 3, 1 (1): 1

2 5, 1 (1): 1

2 2, 1 (1): 1

3 1, 4 (1): 1

3 4, 2 (1): 1

3 3, 1 (1): 1

3 5, 1 (1): 1

3 2, 1 (1): 1

4 1, 3 (1): 1
```

```
4 4, 3 (1): 1

4 3, 2 (1): 1

4 5, 4 (1): 1

4 2, 1 (1): 1

5 1, 3 (1): 1

5 4, 3 (1): 1

5 3, 3 (1): 1

5 5, 2 (1): 1

5 2, 5 (1): 1

1 1, 3 (1): 1

1 4, 3 (1): 1

1 3, 3 (1): 1

1 5, 5 (1): 1

1 2, 2 (1): 1
```

Configuración del controlador difuso de la articulación 04 en Fuzzy Logic Toolbox.


File
Edit
View
Error
Velocidad
Articulacion_04
(mamdani)
Torque
FIS Name:
Articulacion_04
FIS Type:
mamdani
And method
min
Or method
max
Implication
min
Aggregation
max
Defuzzification
centroid
Current Variable
Name
Error
Type
input
Range
[-100 100]
Help
Close
Ready



Ventana Rule Editor de la articulación 04.

```
[System]

Name='Articulacion_04'

Type='mamdani'

Version=2.0

NumInputs=2

NumOutputs=1

NumRules=25

AndMethod='min'
```

```
OrMethod='max'

ImpMethod='min'

AggMethod='max'

DefuzzMethod='centroid'

[Input1]

Name='Error'

Range=[-140 140]

NumMFs=5

MF1='MuyPos':'trapmf',[56 115 140 140]

MF2='MuyNeg':'trapmf',[-140 -140 -115 -56]

MF3='Negativo':'trimf',[-115 -52.5 0]

MF4='Cero':'trimf',[-21 0 21]

MF5='Positivo':'trimf',[0 52.5 115]

[Input2]

Name='Velocidad'

Range=[-50 50]

NumMFs=5

MF1='RapidoDer':'trapmf',[-50 -50 -35 -22]

MF2='RapidoIzq':'trapmf',[22 36 50 50]

MF3='Cero':'trimf',[-14 0 14]

MF4='Der':'trapmf',[-38 -24 -14 0]

MF5='Izq':'trapmf',[0 14 24 38]
```

```
 [Output1]
Name='Torque'
Range=[-1 1]
NumMFs=5
MF1='MuchoAtras':'trapmf',[-1 -1 -0.5 -0.25]
MF2='Cero':'trimf',[-0.25 0 0.25]
MF3='MuchoAdelante':'trapmf',[0.25 0.5 1 1]
MF4='Atras':'trimf',[-0.5 -0.25 0]
MF5='Adelante':'trimf',[0 0.25 0.5]
 [Rules]
2 1, 2 (1): 1
2 4, 4 (1): 1
2 3, 1 (1): 1
2 5, 1 (1): 1
2 2, 1 (1): 1
3 1, 4 (1): 1
3 4, 2 (1): 1
3 3, 1 (1): 1
3 5, 1 (1): 1
3 2, 1 (1): 1
4 1, 3 (1): 1
4 4, 3 (1): 1
```

```
4 3, 2 (1): 1

4 5, 4 (1): 1

4 2, 1 (1): 1

5 1, 3 (1): 1

5 4, 3 (1): 1

5 3, 3 (1): 1

5 5, 2 (1): 1

5 2, 5 (1): 1

1 1, 3 (1): 1

1 4, 3 (1): 1

1 3, 3 (1): 1

1 5, 5 (1): 1

1 2, 2 (1): 1
```