SIMULADOR DE UN ROBOT SCARA DE 4 GRADOS DE LIBERTAD BASADO EN REALIDAD VIRTUAL

DIANA CAROLINA LUNA DIAGO
DIEGO ARMANDO CHECA ROJAS

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
DEPARTAMENTO DE ELECTRÓNICA, INSTRUMENTACIÓN Y CONTROL
LÍNEA DE INVESTIGACIÓN EN ROBÓTICA INDUSTRIAL
POPAYÁN
2007

SIMULADOR DE UN ROBOT SCARA DE 4 GRADOS DE LIBERTAD BASADO EN REALIDAD VIRTUAL

Diana Carolina Luna Diago
Diego Armando Checa Rojas

Monografía Para Optar al Titulo de
Ingeniero en Automática Industrial

Director
Victor Hugo Mosquera Leyton
Especialista en Informática Industrial
Ingeniero en Electrónica y Telecomunicaciones

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERIA ELECTRONICA Y TELECOMUNICACIONES
DEPARTAMENTO DE ELECTRONICA INSTRUMENTACION Y CONTROL
LINEA DE INVESTIGACION EN ROBOTICA INDUSTRIAL
POPAYAN
2007

TABLA DE CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

RESUMEN

En el presente trabajo se diseña e implementa una simulación virtual de un robot de 4 grados de libertad tipo SCARA, utilizando el entorno de simulación virtual, para realizar prácticas educativas de diseño y sintonización de controladores aplicados al brazo robótico.

Inicialmente se investiga sobre las herramientas de simulación que permiten crear un modelo virtual de manera fácil y con requerimientos computacionales mínimos. Uno de los objetivos es crear las bases para implementar un laboratorio virtual a través de internet. Para realizar las pruebas de simulación se utilizará un modelo del robot SCARA en estado continuo, 4 trayectorias y 2 tipos de controles, PID y CTC.

Al final se obtiene una simulación virtual no inmersiva de un robot SCARA utilizando la herramienta Easy Java Simulations. Se entrega una aplicación que se puede distribuir por red o ejecutarla en línea. La simulación fue validada con base en otras simulaciones ya probadas del SCARA usando como parámetro de comparación el promedio del índice del cuadrado del error.

PALABRAS CLAVES: Realidad virtual, Easy Java Simulations, 20sim, Control PID, Control CTC, *Applets*, Laboratorio Virtual.

INTRODUCCIÓN

La realidad virtual es una de las nuevas tecnologías que ha permitido alcanzar horizontes antes no imaginados. Ahora es posible entrenar pilotos de aviación sin siquiera dejar el suelo [1], manejar maquinaria en ambientes peligrosos mediante control remoto, realizar una operación quirúrgica en un paciente virtual [2], etc. Con el presente trabajo se busca aportar en la aplicación de la realidad virtual en el campo educativo de la robótica.

Las herramientas computacionales de simulación matemática hasta ahora utilizadas en la Universidad del Cauca, específicamente en el programa de Automática Industrial, permiten diseñar y probar los robots, ante diferentes señales de entrada, bajo ciertas condiciones de control, dando como resultado gráficas que representan la trayectoria final y los errores tanto articulares como cartesianos, entre otras opciones; sin embargo, durante el transcurso de este estudio el estudiante no se familiariza con la apariencia física del robot, únicamente lo hace con el modelo matemático del mismo; pues bien, el presente proyecto facilita al estudiante la comprensión de su estudio al relacionarlo visualmente de una forma fácil e intuitiva con el robot SCARA de 4 grados de libertad, las trayectorias y los métodos de control, generando la sensación de tener el robot presente mediante el uso de la realidad virtual no inmersiva.

En el primer capítulo se habla sobre la realidad virtual, ambientes virtuales, laboratorios virtuales y sus aplicaciones. El segundo capítulo describe de manera general el modelo del SCARA, los controladores PID y CTC, las trayectorias que tiene la simulación. Además se describe la simulación que servirá como punto de comparación en la validación de la herramienta. El tercer capítulo muestra la investigación de la herramienta y los criterios de selección. El diseño y la implementación son tratadas en este mismo capítulo así como también las posibilidades de construir un laboratorio virtual con la aplicación obtenida. La validación de la simulación construida y los resultados obtenidos son analizados en el Capítulo 4. Por último de presentan las conclusiones que se obtuvieron luego de la culminación del presente trabajo.

1. REALIDAD VIRTUAL Y AMBIENTES VIRTUALES

1.1. ESTADO DEL ARTE

La realidad virtual ha tenido grandes avances y cada vez hay más áreas de la ciencia en donde tiene aplicabilidad. Una de las áreas en donde se están concentrando los mayores esfuerzos por utilizar la realidad virtual es en la medicina, específicamente la cirugía robótica. Se plantea como una necesidad tener salas de operación tele controladas que permitan realizar cirugías a grandes distancias entre el cirujano y el paciente [2]. Para investigar y aprender esta práctica se está utilizando realidad virtual y retroalimentación hápticas por medio de MEMICA (MEMICA es una técnica que utiliza un liquido el cual varía su viscosidad de acuerdo a la cantidad de energía eléctrica que reciba, de esta manera se puede simular el contacto de un bisturí con un tejido blando ó al cortar un hueso, utilizando el mismo dispositivo de control para el cirujano)[2] La retroalimentación háptica la cual consiste en permitir que el sentido humano del tacto pueda interactuar con mundos generados por computadora. La háptica se puede dividir en dos subgrupos, retroalimentación de fuerza o retroalimentación táctil. La primera hace referencia a la interacción de los mundos generados por computador con los músculos y tendones humanos y la segunda a la interacción con los nervios de la piel [3]. La realimentación háptica permite que el cirujano además de observar la simulación en realidad virtual, también pueda sentir al paciente por medio del instrumento de operación, como un bisturí que este conectado al PC, de esta manera el médico percibirá cuando corta al paciente imaginario como si fuera un paciente real [1], [4].

Otros de los avances importantes se encuentran en el diseño de robots. Cyberbotics Ltd. [4] es una de las empresas líderes en el mercado de software para simulación de robots móviles, creada en 1998 por Oliver Michel, esta empresa ha tenido un gran auge, ya que el medio científico ha reconocido la importancia de este tipo de herramientas de diseño. Esta empresa se especializa en diseño de robots móviles y su software permite, además de simular el funcionamiento del robot, definir el entorno para el cual fue creado el robot o

en el cual va a desempeñar su trabajo permitiendo de esta manera observar el comportamiento del robot en dicho entorno [4].

En Europa actualmente se encuentran desarrollando el proyecto MARVEL (Mechatronics Training in Real and Virtual Environments) [5]. Alemania e Inglaterra son algunos de los países que se encuentran dentro del proyecto que busca fomentar la realidad virtual para efectos de aprendizaje, en este caso en especial: entrenamiento de habilidades Mecatrónicas [5].

En la Universidad Central de Venezuela, se cuenta con un brazo robot de cinco grados de libertad (INITIUM) para el Postgrado de Instrumentación. Dada la necesidad de una herramienta alterna que permita la simulación de INITIUM para prolongar su vida útil, y además permitir la simulación de robots con los que no cuenta actualmente dicha universidad, surgió el proyecto "Plataforma virtual Interactiva para la Modelación y Simulación de Robots bajo el ambiente de programación de Matlab" basado en un software que permite la visualización virtual de los modelos simulados en Matlab [6].

En Mayo de 2006 se llevo acabo en México el "1er Congreso Nacional de Mecatrónica y Tecnologías Inteligentes". En este evento fueron expuestos algunos trabajos en torno a la realidad virtual. Para resaltar se encuentran dos ponencias. "Realidad virtual" y "Visualización de soluciones numéricas" las cuales fueron autoría del Dr. Herbert Lara Ordaz, docente del Instituto Técnico Superior de Huichapan, México; la primera trata sobre la visualización del comportamiento de robots mediante técnicas de programación gráfica. La segunda, por su parte, utiliza la herramienta de simulación Matlab para resolver modelos matemáticos y su observación en ambientes virtuales. Estas dos ponencias ponen de manifiesto el deseo de la comunidad de investigadores en robótica en disminuir el tiempo y costo de desarrollo [7].

Existen algunas herramientas software, como el robot virtual RV-M1, que son comercializadas en Internet, las cuales permiten al usuario entregar la trayectoria y ver el comportamiento del robot, antes de adquirirlo físicamente. El RV-M1 es comercializado por la empresa TecnoEdu [8].

Hay otros usos de la realidad virtual en este momento como: entrenamiento para pilotos de aviones comerciales y militares [1], entrenamiento de las fuerzas especiales de los Estados Unidos [9] y hasta tratamientos sicológicos para pacientes con fobias, el cual muestra que tanto ha avanzado la realidad virtual en el mundo [10].

En el ámbito nacional, la realidad virtual está un poco atrasada con respecto a los avances mundiales [11], [12]. Mientras en el mundo se realizan aplicaciones y desarrollos de realidad virtual, en Colombia todavía se está discutiendo e informando de las ventajas que trae este tipo de técnicas para la formación y desarrollo de la región [11], [12]. Sin embargo ya hay algunos proyectos pioneros como el de “Realidad virtual distribuida para la educación a larga distancia”, que busca crear ambientes no inmersivos para que los estudiantes puedan acceder a salones de clase virtuales, hablar con otros estudiantes, leer de un tablero virtual y escuchar las clases del profesor [13]. En Colombia hay un laboratorio virtual en la Universidad Militar Nueva Granada que permite hacer simulaciones de gran escala y posee varias terminales para que 10 estudiantes o 10 grupos puedan desarrollar sus trabajos de diseño y simulación al mismo tiempo. Uno de los ejemplos de desarrollo de dicho laboratorio es el control de manipuladores neumáticos virtuales por medio de señales discretas usando Controladores Lógicos Programables (PLC) reales [14].

En la Universidad del Bosque, específicamente la facultad de medicina, se afronta la necesidad de construir un laboratorio de realidad virtual que permita al estudiante aprender de forma autónoma [11], como lo sugiere el Dr. Hernando Camacho, Decano de dicha facultad “La simulación cibernética es un algo más en la educación del estudiante, que debe fortalecer la relación paciente-médico y no debilitarla. La simulación cibernética en medicina basa la concepción pedagógica de la enseñanza en el alumno y no en el profesor, desarrolla en el alumno su capacidad de descubrimiento y autoaprendizaje y auto instrucción de sus propios objetivos de conocimiento y no en la tradicional pedagogía magisterial paternalista basada en la instrucción. De esta manera el estudiante desarrolla su propia manera de pensar y de aprender. La informática y la simulación, son un nuevo medio de instrucción para el docente y nuevo medio de aprendizaje al servicio del

estudiante. Esto implica que las universidades deben tener su departamento de informática académica que permita desarrollar estas tecnologías" [11].

Existen otros adelantos como lo muestra el artículo desarrollado por Byron Pérez en medicina, específicamente endoscopias virtuales. El artículo presenta muchas especificaciones técnicas de la aplicación de la realidad virtual en medicina [12].

Actualmente el grupo de investigación en Automática Industrial, de la Universidad del Cauca realiza un proyecto a nivel de maestría sobre el desarrollo de una interfaz virtual de robots quirúrgicos [15], a su vez, trabaja en un proyecto para la implementación de una prótesis de mano virtual, el cual esta soportado por subproyectos como: modelado y control de una mano robótica, agarre estable de objetos con una prótesis de mano robótica, diseño de una prótesis de mano en un ambiente asistido por computador y extracción de características e identificación de movimiento a partir de señales electromiográficas [15].

1.2. REALIDAD VIRTUAL

La exploración en el campo de la *realidad virtual* se inició con los experimentos de un grupo de investigación de la Universidad de Harvard encabezado por Ian Sutherland. Este grupo de investigadores diseñó el primer casco, conocido como *incredible helmet (casco increíble),* un dispositivo bastante rudimentario con dos tubos de rayos catódicos que, aunque eran bastante pequeños para la época, no dejaban de ser pesados y voluminosos [16].

El sensor de posición del *incredible helmet* se hallaba fijado en el techo por una barra rígida, que servía para traducir los movimientos de la cabeza a desplazamientos de unos potenciómetros, cuya posición era detectada por el computador. Por su construcción el dispositivo presentaba numerosos problemas de movilidad y comodidad [16].

Las limitaciones del hardware existente en aquella época ocasionaron que las primeras tecnologías resultaran poco convincentes. Los ambientes se creaban usando el sistema de generación de gráficos vectoriales más avanzado del momento sin embargo, solo se logró producir la sensación de estar en un mundo de objetos que parecían estar hechos de alambre, y la ilusión de inmersión era insuficiente [16], [17].

Si bien las primeras exploraciones en el campo de la *realidad virtual* no fueron exitosas, sirvieron para demostrar que era posible llegar a una mayor evolución a futuro. Como consecuencia, numerosas empresas y centros de investigaciones civiles y militares mostraron interés en su desarrollo [16].

La primera adaptación de un dispositivo que permitiera simular reacciones táctiles de fuerza fue lograda en 1968 por un grupo de investigadores de la Universidad de Carolina del Norte, dirigido por el profesor Frederick Rooks. Para esto se sirvieron de un dispositivo robótico similar a los que se usan para la manipulación remota de materiales radiactivos, de forma que ofreciese mayor o menor resistencia al movimiento según fuese necesario. Un dispositivo similar a éste, aunque mucho más desarrollado es empleado en la actualidad para numerosas tareas de la *realidad virtual* [17].

Una de las figuras más destacadas en este campo es Jaron Lanier, quien creó el guante de datos y fundó la empresa VPL Research junto con Thomas Zimmerman. Su invento empezó a venderse en poco tiempo a organismos como la NASA y el Pentágono. Poco después VPL fue absorbido por la multinacional Thompson y Lanier es despedido durante el proceso, sin que por ello dejara de ser uno de los personajes claves en el desarrollo de la *realidad virtual* [16], [17].

En el Reino Unido el desarrollo de ambientes sintéticos de inmersión fue protagonizado por Jonathon Waldern fundador de W Industries y de Virtuality, quien lanzó al mercado el primer producto de basado en el uso del casco, cuyo diseño se inició en 1981, completándose el primer prototipo en 1988. En julio de 1991 apareció Dactyl Nightmare el primer juego en el que varios usuarios pueden interactuar en un mismo espacio [16], [17].

La historia de la *realidad virtual* en Internet se inició con el *GopherVR*, un navegador que creaba una interfaz al *gopherespacio* generando mundos virtuales al vuelo. El interés en este sistema decayó en 1993, con la llegada del World Wide Web (WWW) [16].

El lenguaje de *realidad virtual* empleado actualmente en la red es el *VRML (Virtual Reality Modeling Language),* cuya historia se inició en 1994, con la Primera Conferencia Internacional en el World Wide Web realizada en Mayo de ese año. En ella Mark Pesce y Tony Parisi presentaron una herramienta de visualización llamada Labyrinth [18].

A partir de ese momento se propuso un ciberespacio consistente y definido por el uso de *VRML* para mejorar la navegación en la red sin embargo, la discusión y la actividad que siguieron resultaron en la especificación de un lenguaje común para definir las escenas tridimensionales más que en la generación de una interfaz [18].

En Mayo de 1995 se presentó *VRML 1.0,* un lenguaje para definir mundos virtuales estáticos con la anchura de la red, basados en el formato de archivo *OpenInventor* ideado por Silicon Graphics [19]. En Agosto de ese mismo año se introdujo *VRML 2.0,* un lenguaje mucho más poderoso para definir mundos virtuales dinámicos, con animación, interacción con el usuario y scripts para programas. Posteriormente apareció *VRML 97*, una revisión del *VRML 2.0*, que aún hoy en día, 10 años después, es completamente operacional y esta vigente [18].

1.2.1. Definición de realidad virtual

A pesar de las múltiples aplicaciones o usos que se le han dado a la realidad virtual, aún no se ha establecido una definición universalmente aceptada que permita formalizar el concepto de realidad virtual. Esto es debido a la gran versatilidad que tiene esta tecnología de poder aplicarse a casi cualquier ciencia (por no decir cualquiera). Cada aplicación crea una definición parcial que al mismo tiempo excluye algunos detalles que plantea la realidad virtual en otros campos. Una de las primeras definiciones fue escrita por Aukstalkanis y Blatner, “la realidad virtual es una forma humana de visualizar, manipular e interactuar con ordenadores y datos complejos”. Esta definición es algo

sencilla sin embargo representa el tiempo en que fue escrita, un periodo donde la imaginación estaba mucho más allá de la capacidad real de los computadores (1993) [17], [20].

Otra definición un poco más completa es de A Rowell. (y la que se adopta en el presente proyecto) “la simulación virtual es una simulación interactiva por computador desde el punto de vista del participante, en la cual se sustituye o se aumenta la información sensorial” [21], [22].

Esta definición rescata los tres puntos importantes que debe cumplir la realidad virtual [21].

- Sistema interactivo
- Interacción implícita
- Inmersión sensorial

El sistema interactivo se refiere a que el usuario es “libre” de desplazarse por el mundo virtual sin necesidad de haber programado la trayectoria en que se quiere desplazar, el sistema responde según la voluntad del usuario. Esto representa que el usuario pueda tomar decisiones en “tiempo real” para así observar la escena desde el punto de vista seleccionado [18], [21].

En cuanto a interacción implícita se refiere a que el usuario no debe aprender comandos o algún procedimiento para realizar alguna acción en el mundo virtual. Por el contrario, el usuario realiza movimientos que son naturales a los utilizados en el mundo real para desplazarse. Se busca entonces que el computador se adapte a la naturaleza humana y no al contrario, asegurando así que la experiencia en el mundo virtual sea lo más cercana posible a la experiencia en el mundo real [18], [21].

Por ultimo la inmersión sensorial se refiere a la desconexión de los sentidos del mundo real y la conexión de los mismos al mundo virtual [21].

1.2.2. Clasificación de realidad virtual

Existen diversas formas de clasificar los actuales sistemas de realidad virtual. A continuación se presenta una, basada en el tipo de interfaz con el usuario [23]:

- Sistemas de Ventanas (*Window on World Systems*): Se han definido como sistemas de Realidad Virtual sin Inmersión. Estos sistemas son conocidos como WOW (*Window on a World*) y también como realidad virtual de escritorio. Estos sistemas tratan de hacer que la imagen que aparece en la pantalla luzca real y que los objetos, en ella representada, actúen con realismo [24], [25].

- Sistemas de Mapeo por Video: Este enfoque se basa en la filmación, mediante cámaras de vídeo, de una o más personas y la incorporación de dichas imágenes a la pantalla del computador, donde podrán interactuar - en tiempo real – con otros usuarios o con imágenes gráficas generadas por el computador [24], [25].

- Sistemas Inmersivos: Los sistemas más perfeccionados de realidad virtual permiten que el usuario pueda sentirse "sumergido" en el interior del mundo virtual. Estos sistemas inmersivos se encuentran generalmente equipados con un casco-visor que contiene dos pantallas miniaturas coordinadas para producir visión estereoscópica y recursos acústicos de efectos tridimensionales [24], [25].

- Sistemas de Telepresencia (Telepresence): Esta tecnología vincula sensores remotos en el mundo real con los sentidos de un operador humano. De esta forma el usuario puede operar el equipo como si fuera parte de él [24], [25].

  La "telepresencia" constituye un aspecto que puede ser de gran importancia para la investigación, ya que permite al ser humano llegar a sitios peligrosos sin necesidad de correr riegos o realizar acciones físicamente imposibles para cualquier operario. Los sistemas teleoperados pueden proveer mapas virtuales y

ayudar en diversas tareas como construcción y mantenimiento de plantas nucleares [24], [25].

- Sistemas de Realidad Mixta Aumentada: Al fusionar los sistemas de telepresencia y sistemas por ventanas ó realidad virtual no inmersiva se obtienen los denominados sistemas de Realidad Mixta. En este tipo de realidad virtual las entradas generadas por el computador (objetos virtuales y sus interacciones) se mezclan con entradas de telepresencia y/o la visión de los usuarios del mundo real (objetos reales digitalizados por cámaras de video por ejemplo) [26].

1.3. AMBIENTES VIRTUALES

Un ambiente virtual es una interfaz que permite a los humanos visualizar e interactuar con ambientes generados por medio de computadores en tiempo real, a través de los canales sensoriales humanos. Dicho de otra manera es el mundo que se ha creado en donde es posible realizar acciones con los objetos virtuales.

1.3.1. Clasificación Zeltzers para ambientes virtuales

Zeltzers propone tres conceptos para la clasificación de los ambientes de realidad virtual, cada uno de estos conceptos está dentro de un eje de un cubo donde el punto (0,0,0) son las aplicaciones más primitivas de realidad virtual y en el punto (1,1,1) se sitúan las aplicaciones más avanzadas donde no se podrá distinguir entre la realidad y el mundo virtual, ver Figura 1[27].

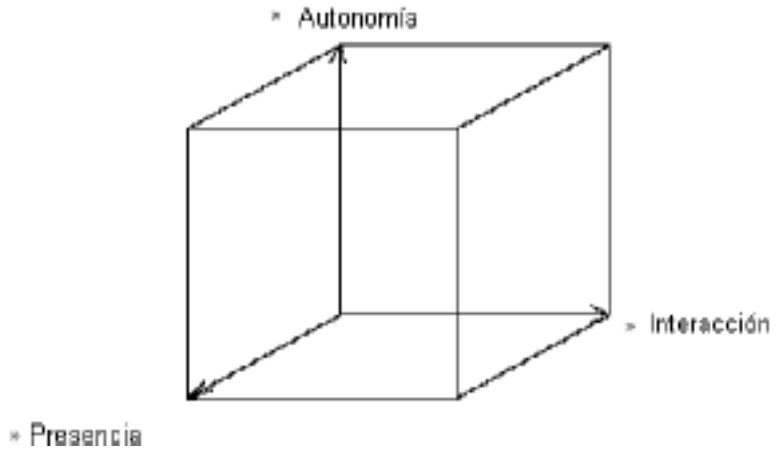


Figura 1. Características de Ambientes Virtuales según Zeltzers [27].

Las tres características propuestas por Zeltzers son [27]:

- Presencia. Provee una medida de la fidelidad de los sensores de entrada y de los canales de salida. Es la medida en la que el ambiente se sienta real.

- Interacción. Se refiere al nivel de acceso de los parámetros o variables de un objeto. Es la medida en la que el usuario pueda explorar libremente el mundo virtual.

- Autonomía. Se refiere a la medida en la que los objetos virtuales reaccionan a eventos y estímulos. Es la medida de la fidelidad con la que reaccionan los objetos virtuales frente a los estímulos.

1.4. LABORATORIO VIRTUAL

El Laboratorio Virtual acerca y facilita la realización de experiencias a un mayor número de alumnos, aunque alumno y laboratorio no coincidan en el espacio. Permite simular fenómenos y modelos físicos, conceptos abstractos, mundos hipotéticos, controlar la

escala de tiempo, etc., ocultando el modelo matemático y mostrando el fenómeno simulado de forma interactiva [28].

Un laboratorio virtual (LV) es un sistema computacional que pretende aproximar el ambiente de un laboratorio tradicional (LT) [28]. Los experimentos se realizan paso a paso, siguiendo un procedimiento similar al de un LT: se visualizan instrumentos y fenómenos mediante objetos dinámicos (*Applets* de Java o Flash, cgi-bin, javascripts,...), imágenes o animaciones. Se obtienen resultados numéricos y gráficos, tratándose éstos matemáticamente para la obtención de los objetivos perseguidos en la planificación docente de las asignaturas [28].

1.4.1. Tipos de laboratorios virtuales

La gran variedad de tipos de laboratorio virtual que puedan existir, dependiendo de su estructura, el tipo de tecnología o software que usen, pueden ser clasificados, de manera general, en las tres clases siguientes [29]:

- *Laboratorios virtuales software*. Son laboratorios virtuales desarrollados como un programa de software independiente destinado a ejecutarse en la máquina del usuario, y cuyo servicio no requiere de un servidor Web. Es el caso de programas con instalación propia, que pueden estar destinados a plataformas Unix, Linux, M.S. Windows e incluso necesitar que otros componentes de software estén instalados previamente, pero que no necesitan los recursos de un servidor determinado (como bases de datos o módulos de software de servidor) para funcionar. También laboratorios virtuales pensados inicialmente como aplicaciones Java accesibles a través de un servidor Web se pueden considerar de este tipo si funcionan localmente y no necesitan recursos de un servidor en concreto [28], [29].

- *Laboratorios virtuales Web o distribuidos*. En contrate con los anteriores, este tipo de laboratorios se basa en un software que depende de los recursos de un servidor. Esos recursos pueden ser determinadas bases de datos, software que

requiere ejecutarse en su servidor, la exigencia de determinado hardware para ejecutarse, esto es, no son programas que un usuario pueda descargar en su equipo para ejecutar localmente de forma independiente, y para que funcionen el usuario debe conectarse al servidor y ejecutar la aplicación [28], [29].

- *Laboratorios remotos*. Se trata de laboratorios remotos que permiten operar remotamente cierto equipamiento, bien sea didáctico como maquetas especificas, o industrial, además de poder ofrecer capacidades de laboratorio virtual. En general, estos laboratorios requieren de equipos servidores específicos que les den acceso a las máquinas a operar de forma remota, y no pueden ofrecer su funcionalidad ejecutándose de forma local. Otro motivo que hace dependientes estos laboratorios de sus servidores es la habitual gestión de usuarios en el servidor [28], [29].

Después de comprender las clasificaciones de Realidad virtual y Laboratorios Virtuales, es de notar que el presente proyecto se caracteriza por ser un Sistema de Ventana (*Window on World Systems*) ya que uno de los objetivos es que el usuario visualice el robot SCARA de una forma muy real. El proyecto también se encuentra enmarcado dentro de los Laboratorios Virtuales Web o Distribuidos por sus características técnicas, según las cuales el usuario deberá conectarse al servidor para de esta forma correr la simulación del SCARA.

2. SIMULACION DE UN ROBOT SCARA DE 4 GRADOS DE LIBERTAD

En este capítulo se presenta el estudio detallado de las características del robot SCARA, como los son el modelo matemático y físico del mismo, para después realizar la simulación en Matlab 7.0. Además se describen las técnicas de control utilizadas para la simulación, control PID y CTC.

2.1. DESCRIPCIÓN GENERAL DEL ROBOT SCARA

Tras un estudio realizado por la Comisión Económica para Europa de las Naciones Unidas (CEE-ONU) y por la Federación Internacional de Robótica (FIR), con sede en Alemania, se concluyó que en la actualidad hay aproximadamente 350.000 unidades robóticas en Japón, 233.000 en la Unión Europea y unos 104.000 en Estados Unidos [30]. Se puede decir que en Europa cerca del 54% de los robots son utilizados para tareas de manipulación. Uno de los robot mas utilizados para estas tareas es el SCARA (*Selective Compliance Arm for Robotic*) desarrollado en 1979 en la universidad de Yamanashi en Japón e introducido comercialmente en el año de 1981 [30], [31]. Teniendo en cuenta la amplia acogida a nivel industrial del robot SCARA, se decidió trabajar con este tipo de manipulador en el proyecto.

Los manipuladores tipo SCARA (ver Figura 2) son robots dotados de libertad total de movimientos en los ejes X e Y pero limitados severamente en sus desplazamientos en el eje Z. Es decir, se comportan de forma parecida al brazo humano, permitiendo localizar el extremo de la mano en cualquier ubicación pero siempre sobre el plano. En el eje vertical solo realizan manipulaciones simples que habitualmente consisten en presionar y desplazarse unos pocos centímetros. Debido a estas características se usan comúnmente en la fabricación de electrónica de consumo y en la clasificación de artículos para su empaquetado [32].

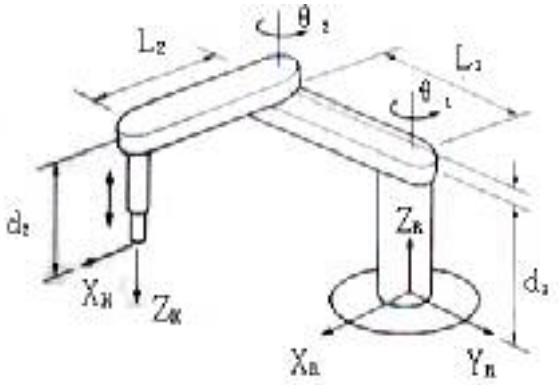


Figura 2. Manipulador tipo SCARA [8].

2.2. MODELO MATEMÁTICO DEL ROBOT SCARA

El modelo matemático del robot utilizado en el presente trabajo fue tomado del trabajo de Grado Titulado "Control Difuso de un Manipulador de Cuatro Grados de Libertad" realizado por Andrés Meneses y Andrea López de la Universidad del Cauca, situación por la cual no se incluye la información sobre la obtención de éste. Para mayor información sobre el tema dirigirse al trabajo mencionado [33].

2.2.1. Modelo geométrico directo

El Modelo Geométrico Directo (mgd), permite obtener la posición cartesiana de la articulación final en función de las posiciones articulares. Este modelo se presenta en la ecuación (2.1).

Esta relación entre el efector final y las articulaciones es realizada por la matriz de transformación ${}^{0}T_{4}$, la cual expresa la posición del órgano terminal con respecto a la base [34].

$$ {}^{0}T_{4} = \begin{bmatrix} C123 & -S123 & 0 & C1C2D3 + C1D2 - S1S2D3 \\ S123 & C123 & 0 & S1C2D3 + S1D2 + C1S2D3 \\ 0 & 0 & 1 & r4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (2.1) $$

Donde,

$C123 = \cos(q_1 + q_2 + q_3)$ (2.2)

$S123 = \text{sen}(q_1 + q_2 + q_3)$ (2.3)

q_i: Posición de la articulación i.

D2, D3 son parámetros del robot que representan la distancia entre la articulación 1-2 y 2-3 respectivamente y r4 es la distancia de la cuarta articulación, ver Figura 3.

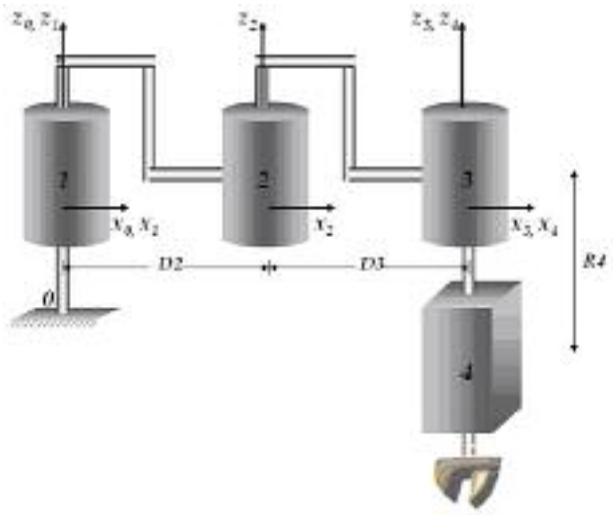


Figura 3. Parámetros del manipulador SCARA.

2.2.2. Modelo geométrico inverso

El mgi permite obtener las coordenadas articulares en función de una coordenada cartesiana en la que se quiere ubicar al órgano terminal. El método de Paul [34] permite

obtener un resultado concreto por medio de unas ecuaciones especificadas para cada tipo de modelo.

La solución a esta ecuación se basa en multiplicar a ambos lados por la matriz ${}^{j}\boldsymbol{T}_{j-1}$ así:

$$\begin{aligned} {}^{1}T_0 U_0 &= {}^{1}T_2\,{}^{2}T_3\,{}^{3}T_4 \ldots {}^{n-1}T_n = U_1 \\ {}^{2}T_1 U_1 &= {}^{2}T_3\,{}^{3}T_4 \ldots {}^{n-1}T_n = U_2 \\ &\vdots \\ {}^{n-1}T_{n-2} U_4 &= {}^{n-1}T_n \end{aligned} \tag{2.4}$$

La solución de la ecuación (2.4) es el mgi y se presenta en las siguientes ecuaciones

$$r_4 = P_z \tag{2.5}$$

$$\theta_1 = a\tan 2(Sen(\theta_1), Cos(\theta_1)) \tag{2.6}$$

$$\theta_2 = a\tan 2(\pm\sqrt{1 - Cos(\theta_2)^2}, Cos(\theta_2)) \tag{2.7}$$

$$\theta_3 = a\tan 2(s_y, s_x) - \theta_2 - \theta_1 \tag{2.8}$$

Con:

$$Cos(\theta_2) = \frac{(P_x^2 + P_y^2) - (D2)^2 - (D3)^2}{2D2D3} \tag{2.9}$$

$$Sen(\theta_1) = \frac{(D2 + D3Cos(\theta_2))P_y - D3Sen(\theta_2)P_x}{P_x^2 + P_y^2} \tag{2.10}$$

$$Cos(\theta_1) = \frac{(D2 + D3Cos(\theta_2))P_x + D3Sen(\theta_2)P_y}{P_x^2 + P_y^2} \tag{2.11}$$

2.3. MODELO DINÁMICO DEL ROBOT SCARA

El modelo dinámico del robot permite expresar el comportamiento del manipulador ante las señales de entrada. Tiene dos partes las cuales son el modelo dinámico inverso y el directo [34].

2.3.1. Parámetros Inerciales de Base

Los parámetros inerciales de base ó parámetros identificables constituyen el conjunto mínimo de parámetros con los cuales se puede calcular el modelo dinámico de un robot y pueden ser calculados a partir de los parámetros inerciales estándar M_j (masa de la articulación j), ${}^{j}MS_j$ (vector columna del primer momento de la articulación j con respecto a la referencia R_j), ${}^{j}J_j$ (matriz del tensor de inercia de la articulación j con respecto a la referencia R_j) y Ia_j (matriz diagonal de las inercias de los actuadores de la articulación j), por la eliminación de aquellos que no tienen efecto en el modelo dinámico y por agrupamiento de otros a partir del algoritmo para la determinación práctica de los parámetros de base bajo la metodología expuesta en, que nos permite trabajar sólo con 11 parámetros inerciales de base, ya que los 33 restantes se hacen cero debido a la reagrupación, tal como se puede observar en la Tabla 1 [33].

Tabla 1. Parámetros inerciales de base del robot SCARA

j	XXj	XYj	XZj	YYj	YZj	ZZj	MXj	MYj	MZj	Mj	Iaj
1	0	0	0	0	0	$ZZR1$	0	0	0	0	0
2	0	0	0	0	0	$ZZR2$	$MXR2$	$MY2$	0	0	Ia_2
3	0	0	0	0	0	$ZZR3$	$MXR3$	$MYR3$	0	0	Ia_3
4	0	0	0	0	0	0	0	0	0	$M4$	Ia_4

2.3.2. Modelo dinámico inverso

Por medio de este es posible expresar los torques (Γ) del sistema en función de la posición (**q**), velocidad ($\dot{\boldsymbol{q}}$) y aceleración articular ($\ddot{\boldsymbol{q}}$).

$$\Gamma = f(q, \dot{q}, \ddot{q}, f_e) \qquad (2.12)$$

Para obtener el modelo existen 2 métodos comúnmente conocidos, el de Lagrange y el de Newton-Euler. Para el presente trabajo se utilizará el método de Lagrange [34].

2.3.3. Modelo dinámico directo

Este modelo es el que normalmente se utiliza para realizar las simulaciones del robot ya que entrega la aceleración en función de los torques entregados al sistema [34].

$$\ddot{q} = A(q)^{-1}[\Gamma - Q(q)] \qquad (2.13)$$

La matriz A(q) representa la matriz de inercias del sistema y Q(q) el vector de fuerzas gravitacionales. Así al calcular la inversa de la matriz A y multiplicarla por la diferencia entre el torque y el vector Q se obtiene la aceleración del sistema. Al integrar la aceleración obtenida es posible obtener la velocidad y una segunda integración la trayectoria en coordenadas articulares.

2.4. CARACTERÍSTICAS DEL MODELO SCARA DE ESTUDIO

De igual manera que en la descripción del modelo matemático, las características técnicas del robot también fueron tomadas del trabajo de grado “Control Difuso para un Manipulador de 4 Grados de Libertad” [33], ver Figura 4. El modelo referencia en dicho escrito es el Adept Cobra 800i.

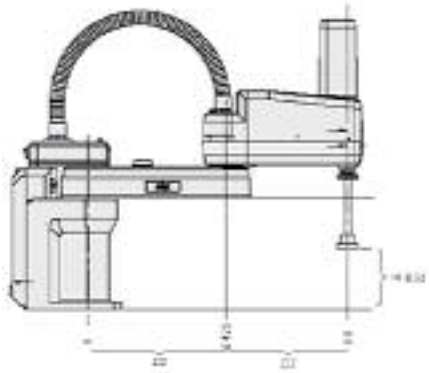


Figura 4. Modelo SCARA de referencia [33].

El robot mencionado anteriormente presenta las siguientes características técnicas. Todas ellas son necesarias para poder realizar una simulación más próxima del comportamiento del robot real.

Tabla 2. Parámetros geométricos del robot SCARA en estudio.

j	σ_j	α_j	d_j	θ_j	r_j
1	0	0	0	θ_1	0
2	0	0	$0.425m$	θ_2	0
3	0	0	$0.375m$	θ_3	0
4	1	0	0	0	r_4

Tabla 3. Valores máximos de posición articular.

Articulación	**Posición articular máxima**
1	$\pm 105^{\circ}$
2	$\pm 175.5^{\circ}$
3	$\pm 360^{\circ}$
4	$0.21m$

Tabla 4. Valores máximos de velocidad articular.

Articulación	Velocidad articular máxima
1	$386^{\circ}/\sec$
2	$720^{\circ}/\sec$
3	$1200^{\circ}/\sec$
4	$1.1m/\sec$

Tabla 5. Parámetros inerciales de base del robot SCARA en estudio [35].

Parámetro	Valor
$ZZR1$	3.38
$ZZR2$	0.063
$ZZR3$	0.1
$MXR2$	0.242
$MXR3$	0.2
$MY2$	0.001
$MYR3$	0.1
Ia_2	0.045
Ia_3	0.045
Ia_4	0.045
$M4$	0.5

Las unidades para los elementos del tensor de inercia son $Kg.m^2$, para el primer momento de inercia son $Kg.m$, para la inercia del accionador $Kg.m^2$ y para las masas Kg.

2.5. TRAYECTORIAS

El robot SCARA es normalmente utilizado para trabajar sobre un solo plano. Esto significa que su efector final es desplazado en los ejes X y Y, dentro del espacio de trabajo.

Las trayectorias pueden ser entregadas al modelo matemático como una secuencia de coordenadas articulares o cartesianas. Se decidió trabajar en espacio cartesiano, ya que su similitud con el mundo real ayuda a la rápida comprensión por parte de los estudiantes.

Las trayectorias que se usaron para realizar las pruebas fueron 2: lineal y la circular.

2.5.1. Trayectoria lineal

Para este recorrido se uso una línea que inicia, tanto en X como en Y, desde la coordenada cartesiana 0.35, hasta la posición final, igualmente para los dos ejes, en 0.45, ver Figura 5.

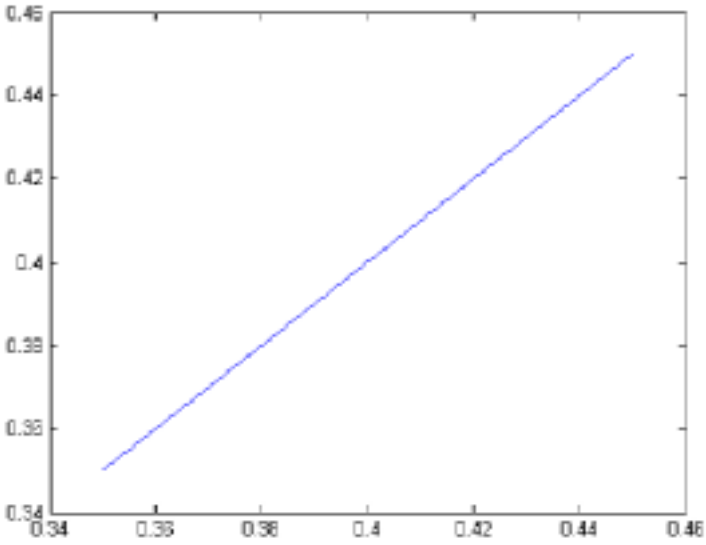


Figura 5. Trayectoria lineal.

La distancia total recorrida es calculada a partir de la formula (2.14):

$$H^2 = A^2 + B^2 \qquad (2.14)$$

Donde H: Hipotenusa
A: Cateto adyacente
B: Cateto opuesto

Reemplazando A y B por 0.1m el resultado de la distancia es de 14.1cm. El tiempo de simulación para este trayecto es de 1 segundo, por lo que el robot deberá moverse a una velocidad de 14.1cm/seg.

2.5.2. Trayectoria circular

Esta trayectoria es un poco más lenta pero presenta una "mayor" dificultad ya que tiene que vencer más concretamente las inercias del robot. En la anterior, el motor no cambia de dirección por lo que después de iniciarse el movimiento, él no debe parar. El recorrido circular es distinto por que debe ser más preciso, el robot esta cambiando de dirección constantemente por lo que el manejo de las inercias requiere mayor acción de control, ver Figura 6.

Para realizar un círculo se utilizaron las siguientes ecuaciones.

$$x=0.02*\sin(2.0943951*t) \quad (2.15)$$

$$y=0.02*\cos(2.0943951*t) \quad (2.16)$$

La amplitud es 0.02, que representa el radio del círculo. La frecuencia 2.0943851 representa un tercio de giro por segundo. Dado que el tiempo de simulación de esta trayectoria es de 3s, el valor de frecuencia se ajusta perfectamente.

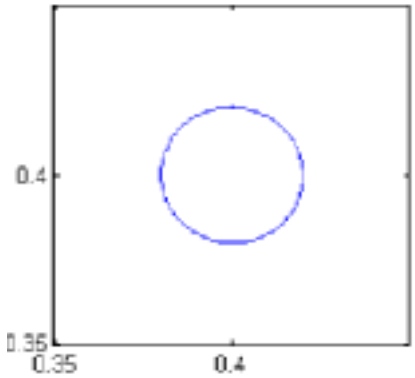


Figura 6. Trayectoria circular.

2.6. CONTROL DE ROBOTS

El robot tiene como principales características: ser un sistema no lineal, altamente acoplado y variable en el tiempo (la dinámica del brazo cambia con el peso de la carga) [36]. La ecuación (2.17) presenta el modelo dinámico inverso simplificado del robot, el cual relaciona los torques aplicados a cada articulación con las aceleraciones, velocidades y posiciones de las mismas [33], [34].

$$\Gamma = A(q)\ddot{q} + Q(q) \qquad (2.17)$$

Donde Γ es el par aplicado, $A(q)$ es la matriz de inercia simétrica y definida positiva y $Q(q)$ es el vector de fuerzas gravitacionales. Así, el problema de control consiste en: dados unos valores deseados de posición q_d y velocidad $\dot{q}_d$, determinar los pares Γ necesarios para llevarlos a cabo [33], [37], [38].

Existen diferentes maneras de diseñar el sistema de control y de ajustar sus parámetros. Una de ellas es el control independiente de cada articulación mediante un controlador PID convencional [38]:

$$\Gamma = -k_p(q_d - q) - k_v(\dot{q}_d - \dot{q}) - k_i \int (q_d - q)dt \quad (2.18)$$

Donde Γ es el par de control, q_d y $\dot{q}_d$ son la posición y velocidad deseadas, q y $\dot{q}$ son la posición y velocidad medidas, K_p, K_d, K_i son matrices diagonales de $(n \times n)$ cuyos elementos genéricos son las ganancias K_{pj} proporcional, K_{dj} derivativa y K_{ij} integral [33], [34], [39], [40].

Otra técnica de control muy utilizada en el medio industrial es el control por par calculado (CTC) o control dinámico, en el cual la ley de control se define por [36]:

$$\Gamma = \ddot{q}_d - k_v(\dot{q}_d - \dot{q}) - k_p(q_d - q) \quad (2.19)$$

Muchos de los robots manipuladores utilizados actualmente, usan un control local descentralizado “proporcional, integral derivativo” [41] con ganancias constantes en cada articulación, con la limitación de que no presentan la acción adecuada cuando las trayectorias deben actuar con consignas a altas velocidades o aceleraciones, ya que los brazos manipuladores, los cuales se componen de varias articulaciones unidas entre sí, poseen una dinámica altamente no lineal con un fuerte acoplamiento entre sus respectivas articulaciones [34], [42], [43].

Debido a la acción limitada de los controladores clásicos, la estrategia de control por par calculado es frecuentemente deseable en aplicaciones avanzadas. Esta consiste en el cálculo de los torques que podrían cancelar la dinámica natural del lazo mecánico del manipulador, de tal modo, que se requiere un conocimiento preciso del modelo matemático del manipulador [42].

2.6.1. Métodos de control para manipuladores robóticos

En el caso de los manipuladores, muchas aplicaciones usan control de posición. Ejemplos incluyen pintura en spray, y operaciones de soldar y desplazar piezas. En tales tareas, el

objetivo es tener a los manipuladores siguiendo un sendero (posición y orientación) con restricciones de tiempo, llamado *trayectoria* del *efector final* del manipulador [42].

El problema en la actualidad con los controladores industriales de posición es la acción de deterioro sobre sus articulaciones cuando se demandan trayectorias rápidas [42].

2.6.2. PID

El algoritmo PID es el controlador realimentado más popular usado en los procesos industriales. Ha sido exitosamente usado por más de 50 años, en más del 95% de procesos industriales de lazo cerrado [44]. La popularidad de los controladores PID puede ser atribuida en parte a su funcionamiento robusto en un amplio rango de condiciones de operación y en parte a su simplicidad funcional, a pesar de las características dinámicas variables del proceso, lo cual permite a los ingenieros operarlos con facilidad [41], [45], [46].

El algoritmo PID consiste de tres modos básicos: modo proporcional, modo integral y modo derivativo. En la práctica, estos modos se combinan y dan como resultado tres algoritmos generalmente usados P, PI y PID, ver Figura 7 [41].

El controlador PID es expresado por la formula (2.18).

El modo proporcional ajusta la señal de salida en proporción directa a la entrada del controlador (la cual es la señal de error, e). El parámetro ajustable a ser especificado es la ganancia del controlador, K. Un controlador proporcional reduce el error pero no lo elimina (a menos que el proceso tenga naturalmente propiedades de integración).

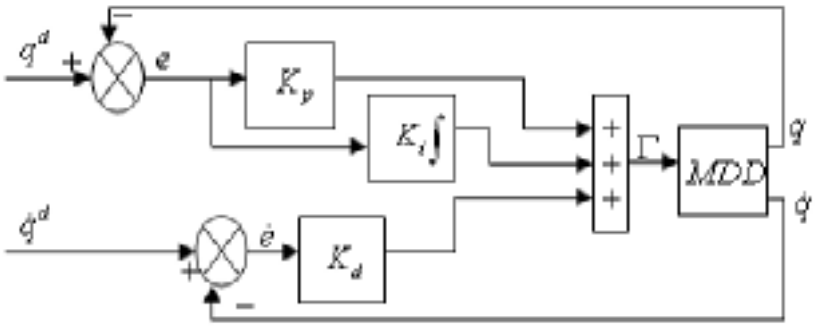


Figura 7. Esquema del control PID en el espacio articular.

El modo integral (frecuentemente llamado reset) corrige cualquier error que puede ocurrir entre el valor deseado (*setpoint*) y la salida del proceso. El parámetro ajustable a ser especificado es el tiempo de integración del controlador Ti.

La acción derivativa *anticipa* el comportamiento de la proporción de tiempo de la variable controlada. T_d es la "proporción de tiempo" y caracteriza la acción derivativa (con unidades de minutos). En teoría la acción derivativa siempre debe mejorar la respuesta dinámica. Sin embargo, el problema de las señales de ruido hace el uso de la acción derivativa, indeseable (diferenciando las señales de ruido, éstas se pueden traducir en movimientos excesivos). La acción derivativa depende de la pendiente del error. Si el error es constante, la acción derivativa no tendrá efecto [41].

2.6.3. CTC (Control por par calculado)

Cuando las tareas a realizar por brazos robóticos industriales requieren rápido movimiento del robot y alta precisión dinámica, es necesario mejorar la acción de control teniendo en cuenta, parcial o totalmente, la interacción dinámica de torques. Tal control es conocido como Control por par Calculado (CTC) o Control Dinámico Inverso basado en la utilización del modelo dinámico del manipulador. Este control es relativamente fácil de implementar y provee resultados satisfactorios en cuanto a errores de seguimiento y robustez. Pero

efectos dinámicos difíciles de obtener en el modelo, dificultan el diseño de un algoritmo eficaz basado en un modelo matemático exacto [42], [43].

Las técnicas de linealización y desacople consisten en transformar un problema de control no lineal en uno lineal usando una ley de realimentación apropiada, proporcionando de cualquier forma una conducta uniforme de la configuración del robot. En el caso de los robots manipuladores rígidos, el diseño de la ley de control de linealización y desacople se facilita por el hecho de que el número de actuadores es igual al numero de variables de las articulaciones, y que el modelo dinámico inverso dando la entrada de control Γ del sistema como una función del vector de estado ($\boldsymbol{q}$, $\dot{\boldsymbol{q}}$) y de $\ddot{\boldsymbol{q}}$ es naturalmente obtenido, ver Figura 8. Estas características aseguran que las ecuaciones del robot definan un así llamado sistema liso, de quien las salidas lisas son las variables de las articulaciones $\boldsymbol{q}$. Subsecuentemente la ley de control solo involucra las variables de estado $\boldsymbol{q}$ y $\dot{\boldsymbol{q}}$, limitadas a una *ley de control estática desacoplada* [34], [42].

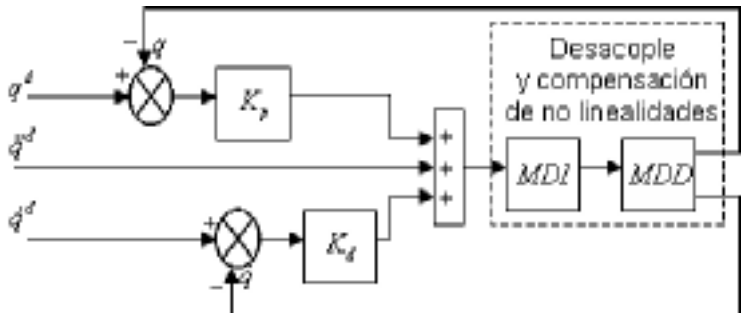


Figura 8. Esquema para el control CTC articular.

2.7. SIMULACIÓN DEL ROBOT SCARA EN MATLAB/SIMULINK

Matlab es un software desarrollado por la compañía MathWorks [47] para realizar cálculos matemáticos de casi cualquier tipo. Esta herramienta ofrece grandes prestaciones en la solución de sistemas de ecuaciones, procesamiento de señales, simulación de sistemas dinámicos, etc.

En los últimos años se ha convertido en una herramienta muy utilizada en el campo científico y educativo gracias a que consta de muchas funciones que dan soporte a diferentes tipos de análisis matemático y diseño. Por ejemplo, existen funciones especializadas para análisis financiero, procesamiento de señales, sistemas lineales, no lineales, etc. Además, posee varios tipos de gráficas muy intuitivas que permiten observar el comportamiento del sistema a simular más fácilmente y de distintas perspectivas con tan solo algunos pequeños cambios.

Matlab posee tanto una ventana de comandos como un editor de archivos para la ejecución de las instrucciones [47].

- La ventana de comandos permite al usuario ejecutar instrucciones luego de ser creadas. En esta ventana cada línea de código es ejecutada una vez se ha introducido al sistema, puede servir para hacer pequeñas pruebas de código pequeño o para obtener datos de verificación. La desventaja, es que para ejecutar otra vez un código hay que volverlo a escribir línea por línea por lo que no es muy agradable en simulaciones con 50 líneas en las que muy seguramente haya que depurar algunas líneas hasta obtener el resultado deseado.

- El editor permite agrupar varias líneas de código y ejecutarlas todas cada vez que se corra el programa según el orden en que estén dispuestas. También se puede guardar los archivos para que en la siguiente sesión no se tenga que escribir de nuevo todo el código.

Simulink es una herramienta, desarrollada también por Mathworks, que permite utilizar diagramas de bloques para construir simulaciones matemáticas. Es muy fácil e intuitivo de utilizar, se puede hacer la inspección del ciclo del programa con solo mirar el diagrama del modelo. Simulink puede comunicarse con Matlab para realizar simulaciones. Esta comunicación permite combinar la potencia de Matlab con la versatilidad de Simulink [47].

Básicamente las simulaciones del robot SCARA en Matlab constan de 3 partes:

- Un archivo de inicio que permite cargar los valores de las variables a utilizar en la simulación en el entorno de Matlab, para que de esta manera las otras partes de la simulación tengan acceso a estos valores. Básicamente es la configuración inicial de la simulación a realizar.

- Conjunto de códigos a manera de funciones que permiten encapsular los modelos del robot, como también, encapsular el código escrito por el usuario para la simulación. Esta funciones sean de Matlab o sean construidas por el usuario pueden ser llamadas por Simulink para utilizarlas en el diagrama de bloques.

- Por último el modelo en Simulink, el cual permite construir el ciclo de la simulación utilizando diagramas de bloques, los cuales pueden llamar a variables y funciones que se encuentren activas en el entorno de Matlab.

Para mejorar la comprensión se hace una pequeña descripción más detallada de cada uno de los componentes.

2.7.1. Archivo de Inicio

Como se dijo anteriormente en este archivo se cargan, al entorno de trabajo Matlab, los parámetros iníciales para la simulación. Estos son:

- Posición inicial del órgano terminal.
- Trayectoria que se quiere seguir con el robot SCARA.
- Parámetros del controlador.
- Tiempo de muestreo.

Al ejecutar este archivo es posible utilizar estos valores desde otros archivos o funciones, inclusive es posible utilizar estos parámetros desde Simulink.

2.7.2. Funciones

Los modelos necesarios para simular el comportamiento del robot SCARA han sido encapsulados en funciones, de esta manera se envía la información necesaria hacia la función y se obtienen los datos que regresa la misma, sin necesidad de conocer a fondo el código que describe la función. Dependiendo de la forma como se quiera simular el comportamiento del SCARA se necesitan algunos archivos, en este caso se quieren enviar coordenadas cartesianas hacia el modelo del robot para que este las siga, para esto es necesario los siguientes archivos:

- El modelo geométrico inverso el cual convierte las coordenadas cartesianas a coordenadas articulares.

- El modelo dinámico del robot, dicho de otra manera este es el modelo de la planta a simular donde se encuentran incluidas características como el tamaño, masa, fuerzas de resistencia, energía potencial y cinemática, etc.

2.7.3. Modelo en Simulink

El modelo en Simulink consta de un diagrama de bloques que posee las siguientes partes:

1. Obtiene las coordenadas cartesianas que se cargaron en el archivo inicio.
2. Convierte las coordenadas cartesianas a articulares por medio de la función que tiene el modelo geométrico inverso del robot. Esta función es puesta dentro de un bloque denominado *Matlab Function* que está diseñado para contener funciones, ya sean escritas por el usuario (como es este caso) o sean parte del sistema.
3. En la tercera parte se encuentra el bloque que representa el bloque PID, el cual recibe información tanto de las coordenadas deseadas como de las coordenadas obtenidas en el momento de simulación inmediatamente anterior.
4. La cuarta parte corresponde al modelo dinámico del robot SCARA. Como se dijo anteriormente este bloque representa la planta del sistema la cual obtiene las

coordenadas deseadas luego de que son controladas por el PID y envía la información de cómo se comporto el sistema nuevamente al controlador.

5. La quinta parte solamente hace referencia al lazo de control, el cual lleva la información necesaria del comportamiento del robot hasta el controlador PID para que haga los cálculos necesarios para que el sistema siga estable.
6. Por último en la Figura 9, se muestran los bloques necesarios para poder obtener información visual del sistema en el tiempo. Realmente no es necesario para realizar la simulación, pero son muy útiles para que el diseñador estudie el comportamiento del sistema, encuentre fácilmente errores y obtenga una descripción rápida del desempeño del sistema simulado.

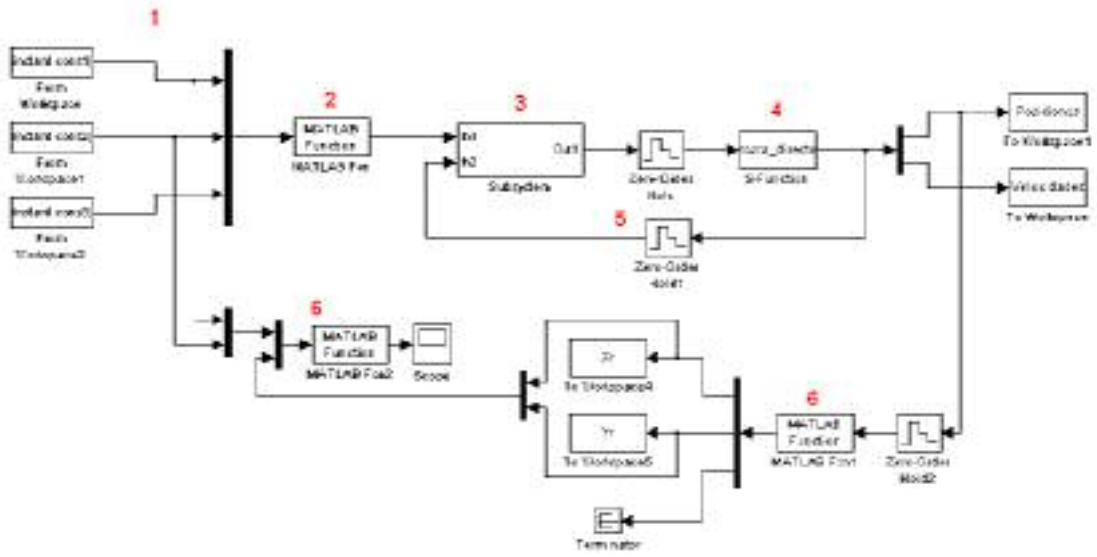


Figura 9. Bloques necesarios para obtener información visual del sistema en el tiempo.

Hasta aquí se ha descrito la forma en qué se realiza una simulación en Matlab, teniendo en cuenta funciones y la comunicación con Simulink, sin embargo para efectos de comparación con otras herramientas de simulación es necesario mostrar la información técnica en cuanto a parámetros y gráficos obtenidos para tener un punto de referencia.

2.8. RESULTADOS OBTENIDOS EN LA SIMULACIÓN DE UN ROBOT SCARA EN MATLAB/SIMULINK

El sistema a simular es el robot SCARA, el cual es un brazo robótico que cuenta con 4 grados de libertad. El modelo y sus características ya fueron descritos anteriormente en este mismo trabajo por lo que no se explicarán a fondo en esta sección.

El controlador que se utilizará para mostrar la forma como se realiza una simulación en Matlab será un PID, el cual es uno de los controladores más conocidos y más utilizados, además de que su funcionamiento es relativamente simple facilitando la comprensión de esta evaluación.

2.8.1. Simulación

Para realizar una simulación de un robot SCARA, controlado por un PID, y que pueda seguir una trayectoria dada son necesarios los siguientes archivos:

- Modelo de geométrico inverso del SCARA.
- Modelo de dinámico del SCARA (modelo de la planta).
- Archivo inicial con parámetros tales como la ubicación inicial del brazo, parámetros del controlador, tiempo de muestreo y la trayectoria a seguir.

Ahora una descripción más detallada del contenido de cada uno de ellos:

2.8.2. Archivo inicio

El código en Matlab que posee este archivo es el siguiente:

```
lineal;
Temp=0.001;
QI=[1.39;-1.89;3.63;0.4];

kp1=210000;
```

```
kp2=80000;
kp3=40000;
kp4=70000;

kv1=1500;
kv2=200;
kv3=60;
kv4=500;

ki1=1000;
ki2=1000;
ki3=1000;
ki4=1000;
```

El programa carga los puntos de la trayectoria a seguir por medio de la instrucción *lineal*, el cual es un archivo creado por el usuario que genera las coordenadas cartesianas y las almacena en 3 vectores bidimensionales llamados cons1, cons2 y cons3. Cada uno de estos vectores tiene una pareja de información que representa la coordenada y el tiempo, ubicando así un punto en el transcurso del tiempo.

El segundo valor es el tiempo de muestreo, es de recordar que se está intentando simular un sistema continuo en un computador que es discreto, es necesario entonces tener un tiempo de muestreo pequeño que permita acercarse a una realidad continua. El valor de este tiempo es de 0.001 ó 1 milisegundo.

A continuación se pueden observar 12 parámetros, los cuales representan las ganancias del controlador PID. Son 4 para cada tipo de ganancia (ganancia proporcional, integral y derivativa) por que el robot tiene 4 articulaciones y para cada una de ellas se necesita un controlador PID sintonizado de forma diferente. Así, para la articulación uno las ganancias de controlador PID son kp1, kv1, ki1, para la segunda articulación kp2, ki2, kv2 y así sucesivamente.

Este archivo es pequeño pero muy importante ya que permite que el usuario configure los parámetros de la simulación tales como la posición inicial, el tiempo de muestreo y la trayectoria deseada.

2.8.3. Archivo con el modelo geométrico inverso

El modelo geométrico inverso se encarga de convertir las coordenadas cartesianas a coordenadas que el modelo del robot pueda entender, esto es, coordenadas articulares.

```
function salida = mgi_scara(X,Y,Z)
D2  =  0.5;
D3  =  0.3;
sx  =  -1;
sy  =  0;

C2  =  (X^2+Y^2-D2^2-D3^2)/(2*D2*D3);

q2  =  atan2((-sqrt(1-C2^2)),C2);

B1  =  D2 + D3*cos(q2);
B2  =  D3*sin(q2);
S1  =  (B1*Y-B2*X)/(B1^2+B2^2);
C1  =  (B1*X+B2*Y)/(B1^2+B2^2);

q1  =  atan2(S1,C1);
q3  =  atan2(sy,sx)-q2-q1;
q4  =  Z;

salida  =  [q1 q2 q3 q4];
```

function salida = mgi_scara(X,Y,Z), indica que este archivo puede ser llamado desde Matlab y pasarle tres datos (coordenadas cartesianas) para que regrese una variable

llamada salida. Salida en realidad es un vector que posee las coordenadas articulares, 4 en total, una para cada articulación.

```
D2 = 0.5;
D3 = 0.3;
sx = -1;
sy = 0;
```

Estos son parámetros del robot. Las dos primeras son las distancias que tienen la articulación 1 hasta la segunda articulación y la distancia entre la segunda y la tercera. Estas medidas cambian para cada tipo de robot, de esta forma, si se quisiera cambiar el tamaño del robot, habría que actualizar estos dos parámetros.

Las siguientes líneas de código sirven para resolver el modelo geométrico inverso. Las ecuaciones pueden ser halladas manualmente o también por medio de algún algoritmo matemático. Cualquiera que sea el método de obtención, se debe obedecer a la siguiente ecuación:

$$Uo = {}^{o}T_1(q_1)\,{}^{1}T_2(q_2) \ldots {}^{n-1}T_n(q_n) \qquad (2.20)$$

Al resolver, se obtiene unas ecuaciones que entregan la relación que tienen las coordenadas cartesianas deseadas y las coordenadas articulares necesarias para realizar la trayectoria deseada. Los tipos de ecuaciones que se pueden obtener se pueden encontrar simplificadas en la literatura [34].

Este archivo entonces tiene codificadas las funciones encontradas para este tipo de robot y lo que hace es tomar los datos entregados a la función (coordenadas cartesianas) y devolver las coordenadas articulares q1, q2, q3 y q4.

2.8.4. Archivo del modelo dinámico del SCARA

Este archivo contiene las ecuaciones de la energía cinética, energía potencial, modelo de frotamientos, etc., en otras palabras todo lo necesario para simular el comportamiento en un ambiente matemático como Matlab.

El código de la función ha sido creado por un software de simulación llamado SYMORO. La función se encapsuló en una *S-Function*, la cual permite simular una planta en tiempo "continuo" por medio de un sistema de variables de estado, a la cual se le añade la posición inicial indicada en el archivo de inicio.

La función recibe los datos de la posición articular deseada, luego de haber sido controlada por el PID, para entregar la velocidad y la posición de cada una de las articulaciones del robot. Estas señales a su vez van hacia el controlador para continuar con el ciclo de control. Pero también se utilizan para obtener las gráficas de desempeño para el diseñador.

Por ultimo se tiene el modelo de la simulación, con todos los componentes necesarios ya cargados, del robot SCARA de 4 grados de libertad, ver Figura 9.

3. DISEÑO Y DESARROLLO DE LA SIMULACIÓN VIRTUAL

En este capítulo se concentrará en el diseño y desarrollo de la simulación virtual. De igual manera se investigará acerca de las herramientas más utilizadas para este tipo de aplicaciones y se escogerá una, de acuerdo a los 4 criterios de selección que plantea el presente trabajo.

3.1. HERRAMIENTAS SOFTWARE UTILIZADAS PARA SIMULACIÓN DE ROBOTS

En el mercado existen muchas herramientas que permiten realizar simulaciones virtuales, cada una tienen sus ventajas y desventajas. Unas de las herramientas más utilizadas son las siguientes.

a. Matlab/simulink

Esta herramienta ya se describió en el capítulo anterior, sin embargo se considera importante nombrarla porque es una de las herramientas de simulación más utilizadas [47].

b. Roboworks

Roboworks es un modelador tridimensional para la simulación de elementos mecánicos. Este programa permite al usuario realizar un modelo en 3D y animarlo desde el teclado, por medio de un archivo .dat creado por el usuario o por medio de archivos ejecutables creados en Matlab, C++, LabView, etc [48].

Roboworks [48] es útil para las personas relacionadas con la Robótica que tengan necesidad de simular movimientos desde un programa de control, ya sea para analizar los resultados ante un estímulo en un Robot o para tener una visión en tiempo real del movimiento de los componentes de un sistema mecánico [48].

La interfaz gráfica del Roboworks consta de las siguientes partes [48]:

- **Vista en 3D:** En esta vista se puede ver en 3 dimensiones el estado en que se encuentra un modelo tridimensional. Desde esta ventana no se puede modificar el modelo, ésta solo sirve para ver la posición de todas las cosas.

- **Vista de Árbol:** En esta vista se crearán todos los modelos. Cada forma visible en la pantalla 3D, se verá referenciada en esta vista, y solo desde esta ventana se pueden modificar sus parámetros.

- **Barra de Tareas:** Ofrece las clásicas tareas de todo programa basado en Windows: Archivo, Editar, Ver, Ventana y Ayuda, además de la opción propia del Roboworks de Animación. Más adelante se explicarán las funciones de cada una de las tareas contenidas bajo estos menús.

- **Barra de Íconos:** En la barra de Íconos solo se encuentran los íconos que pertenecen a las aplicaciones generales de Windows, éstos son: Nuevo, Abrir, Guardar, Cortar, Copiar, Pegar, Acerca del Roboworks y Ayuda. Más adelante se explicaran las funciones de cada una de las tareas contenidas bajo estos íconos.

Otras de las características más importantes de Roboworks es la posibilidad de crear un video con el resultado de la simulación para poder distribuirla o realizar simulaciones sin necesidad de abrir Roboworks. Las trayectorias pueden ser definidas por medio del teclado en tiempo de ejecución, cargarlas desde un archivo .dat en tiempo de compilación o generadas mediante ecuaciones matemáticas escritas en el programa [48].

Los requerimientos en computador son normales, aunque el editor 3D es bueno y las gráficas tienen una buena calidad.

Este es un software propietario y tiene un valor comercial.

c. Microsoft Robotics Studio

Microsoft Robotics Studio [49] es el un nuevo entorno de desarrollo basado en Windows(R) y dedicado a la creación de un software de robótica para diferentes plataformas de hardware. El entorno Microsoft Robotics Studio es una plataforma de desarrollo robótico extensible y completa que incluye lo siguiente [49]:

- Un lenguaje de programación visual que permite a los no programadores programar los robots de forma fácil utilizando los entornos de arrastre y liberación.

- Una herramienta de 3-D que simula las aplicaciones robóticas en los entornos físicos virtuales de simulación, que utiliza el buscador PhysX(TM) de AGEIA(TM) Technologies Inc(ref).

- Un tiempo de utilización de peso ligero y orientado a los servicios que permite las aplicaciones para la comunicación con una amplia variedad de hardware.

Gracias a Microsoft Robotics Studio, las aplicaciones de robótica se pueden desarrollar utilizando una selección de lenguajes de programación, incluyendo los de Microsoft Visual Studio(R) y Microsoft Visual Studio Express (Visual C#(R) y Visual Basic(R)), que son de descarga gratuita, además de Microsoft IronPython. Los lenguajes de las terceras partes que son compatibles con los servicios de Microsoft Robotics Studio también pueden utilizarse [49].

Para las personas que practican un hobby, estudiantes y académicos, Microsoft Robotics Studio está disponible de forma gratuita. Los desarrolladores comerciales de robots interesados en generar ingresos procedentes de las aplicaciones, servicios y robots basados en Microsoft Robotics Studio pueden disponer de la plataforma de desarrollo desde 399 dólares.

Microsoft Robotics Studio ya es compatible con las aplicaciones, servicios y robots de las siguientes compañías: CoroWare Inc., fischertechnik, iRobot, KUKA Robot Group, Larsen & Toubro InfoTech Ltd., LEGO Group, Lynxmotion Inc., Parallax Inc., Phidgets Inc.,

RoboDynamics Corp., Robosoft, RoboticsConnection, Senseta, Sharp Logic, Surveyor y WhiteBox Robotics Inc. Además, muchas de las compañías más importantes de todo el mundo se han unido al Microsoft Robotics Studio Partner Program, y tiene planes de enviar las aplicaciones compatibles, servicios y robots en el futuro. Estos incluyen a Braintech Inc., Camelot Robotics ApS, Cerebellum, ED Co. Ltd., Graupner, Hanulkid Co. Ltd., InTouch Health, JADI Inc., LG CNS, MicroInfinity, Mostitech Inc., RE2 Inc., RidgeSoft LLC, Robo3, SRI, VIA Technologies Inc. y Yujin Robot [49].

Una de las características más importantes de este entorno es que permite simular un robot con elementos hardware actuales, es decir, no se va a simular solo ecuaciones matemáticas de un modelo, en vez de eso se va a simular los elementos reales que conformarían un robot (microprocesadores, sensores, transmisores, protocolos) permitiendo que la simulación sea lo más real posible.

Las gráficas de este entorno son unos de sus fuertes, luces, cámaras, texturas y las demás acciones que tiene un editor de objetos 3D puede ser utilizado.

Una de las ventajas que tiene Microsoft Robotics Studio es la capacidad de interactuar con herramientas de programación como Visual C# y Visual Basic, acelerando la curva de aprendizaje del entorno [49].

Para construir una aplicación de MRS (Microsoft Robotics Studio) se puede realizar en dos formas [49]:

- “A pedal”, que es como normalmente se denomina la programación sin ayuda de ningún editor o wizard.
- O utilizando los scritps de DssNewServices que provee el entorno, el cual ayuda en gran medida a la velocidad de desarrollo.

Sin embargo, detrás de todos los beneficios que ofrece el programa, el entorno de desarrollo tiene unos requerimientos mínimos bastante altos, todas esas grandes

funciones que posee hace que se dificulte la distribución en ambientes educativos, donde los computadores son de capacidad media [49].

MRS posee dos versiones, la 1.0 que es la gratuita y la 1.5 que tiene un costo cercano a los 400 dólares US. El costo es para las personas que quieren hacer negocio con la robótica y necesitan de simular modelos más complejos [49].

d. 20-Sim

20-Sim ("Twente Sim") es un programa de modelado y simulación avanzada que corre bajo Windows. Con 20-Sim el usuario puede simular el desempeño de sistemas dinámicos, tales como eléctricos, mecánicos y sistemas hidráulicos o cualquier combinación de estos sistemas. 20-Sim ha sido desarrollado en el Laboratorio de Control de la Universidad de Twente, como sucesor del famoso paquete TUTSIM [50].

20-Sim soporta completamente modelado grafico, permitiendo diseñar y analizar sistemas dinámicos en una forma amigable e intuitiva, sin compromiso de potencia. 20-Sim está compuesto por dos ventanas [50]:

- Editor
- Simulador

Los sistemas pueden ser modelados en 20-Sim, usando ecuaciones, descripciones en el espacio de estado, enlaces gráficos de diagramas de bloques, y componentes o diagramas icónicos. Estas descripciones pueden ser unidas completamente para crear modelos mixtos. Además, 20-Sim posee muchas funciones y bloques para los diagramas permitiendo reducir el tiempo de codificación de la simulación. El lenguaje sobre el cual ha sido desarrollo este entorno de simulación es SIDOPS+, el cual fue desarrollado exclusivamente para 20-Sim [50].

20-Sim permite crear videos, archivos de datos o hasta generar código en c con los resultados de las simulaciones forma se puede exponer o distribuir el resultado sin

necesidad de compilar el modelo. Otra funcionalidad es que tiene interoperación con Matlab [50].

Este software es bastante liviano y puede ser ejecutado en casi cualquier computador actual. Las gráficas son buenas y bastante amigables. Sin embargo es costoso y el demo no permite realizar muchas pruebas.

e. Easy Java Simulations

EJS, como se nombrará en adelante la herramienta Easy Java Simulations, es una herramienta *open source*, desarrollada por el profesor Francisco Esquembre de la Universidad de Murcia, España. Este software se encuentra dentro del proyecto *Open source* Physics [51], [52].

EJS, esta basado en el potente lenguaje de programación Java y permite realizar casi cualquier simulación de una manera más fácil y didáctica que realizar el programa de simulación en un lenguaje de alto nivel como c++ o java. Con estas características el EJS puede recrear la mayoría de sistema dinámicos sin importar de que tipo. EJS no esta especializado en ningún área [51].

El IDE de EJS se encuentra estructurado en tres partes: el modelo, la vista, y el control [51].

- Modelo: es el espacio destinado para introducir las variables y ecuaciones que describen al fenómeno matemático. De igual manera se encuentra estructurado para que el modelar un sistema sea más sencillo. Cuenta con una sección denominada evolución que permite solucionar sistemas descritos en variables de estado [51].

- Vista: consta de cualquier contenedor (*frames* o diálogos) que permita mostrar la información resultante de la simulación. Gráficas cartesianas, en 3d, textos o modelos virtuales se pueden agregar a estas ventanas [51].

- Control: este componente se encuentra fusionado con los otros dos (vista y modelo). En vista se encuentran los objetos activos que “escuchan” los eventos que los usuarios puedan tener. Pueden ser botones, campos de texto, campos numéricos, *slides*, etc. En modelo se encuentran las funciones que reciben dichos eventos, los procesan y actualizan la vista [51].

Una de las grandes características es que permite crear un ejecutable de la simulación, permitiendo distribuir las simulaciones como un programa y no como un código para ser compilado cada vez que quiera ser simulado o como un video estático que no permite realizar modificaciones. Además, EJS permite crear *Applets*, los cuales son aplicaciones que pueden ser empotradas en una página web y así publicar en internet las simulaciones realizadas [51].

EJS permite interoperar con Matlab y/o Simulink, por lo que facilita la simulación de modelos complejos [51].

Los modelos resultantes no tienen grandes características en cuanto a gráficas, esto es una deficiencia en cuanto a presentación, sin embargo esta “deficiencia” es la que hace que la simulación sea liviana y pueda ser ejecutada en computadores normales, sin tarjetas de video ni procesadores de gran capacidad.

3.2. SOFTWARE PRESELECCIONADOS

3.2.1. Características de selección

Se seleccionaron 2 herramientas entre las opciones expuestas anteriormente de acuerdo a los siguientes criterios:

- Consumo de recursos: Una de las preocupaciones de este proyecto es obtener un sistema de simulación que tenga requerimientos computacionales pequeños. La

idea es que la simulación resultante pueda ser ejecutada en casi cualquier computador.

- Costos del software: Existen herramientas software muy buenas pero al mismo tiempo muy costosas, esto limita el uso de las nuevas tecnologías, sobretodo en ambientes educativos de bajos recursos.

- Compatibilidad: Hace referencia a la facilidad que permite el software de poder instalarse en las diferentes plataformas. Para este estudio se escogieron 4 plataformas que son Windows, Linux, Solaris, Apple (os X). Se considera que este es un criterio de selección por que la idea es que la herramienta generada pueda ser corrida en plataformas diferentes a Windows.

- Fácil distribución: Desde un principio se pensó en implementar un laboratorio virtual por medio de la Web, por lo que el entorno escogido debe ofrecer facilidades para desarrollar esta idea.

3.2.2. Comparación de las herramientas

Se calificará de 1 a 10 el desempeño de cada una de las herramientas con respecto a cada uno de los ítems explicados anteriormente (ver Tabla 6).

Recursos computacionales:	0 requerimientos muy altos, 10 requerimientos muy bajos.
Costos del software:	0 costo muy elevado, 10 sin costo.
Facilidad de distribución:	0 dificultad para distribuir, 10 muy fácil de distribuir.
Compatiblilidad:	2.5 por cada plataforma diferente que acepte (Windows, Linux, Solaris, Apple (OS X))

Tabla 6. Calificación subjetiva de las herramientas de simulación.

Software	Requerimientos computacionales	Costo	Compatibilidad	Facilidad de distribución	Calificación
Microsoft Robotics Studio	2	10	2,5	5	**4,87**
Roboworks	5	6	2,5	6	**4,87**
Matlab	2	6	10	4	**5,5**
Easy Java Simulations	8	10	10	9	**9,25**
20-sim	8	6	2,5	6	**5,62**

Realizando un simple promedio de las calificaciones subjetivas que les fueron asignadas a cada una de las herramientas, se encuentra que las mejores, para el propósito del proyecto, son: 20-Sim y Easy Java Simulations.

En la siguiente sección se desarrollará un estudio más profundo de las dos herramientas preseleccionadas.

3.3. EASY JAVA SIMULATIONS

EJS es una herramienta interactiva desarrollada con el propósito de mejorar el aprendizaje de las ciencias en general a través de la simulación de fenómenos. EJS no ha sido diseñado para programadores profesionales, es dedicado a los profesores y estudiantes quienes quieren crear (o modificar) simulaciones científicas. Con EJS, ellos pueden concentrar sus esfuerzos en escribir y depurar las relaciones en el modelo científico, y dedicar la mínima cantidad de tiempo en técnicas de programación. Profesores sin conocimientos en programación han creado simulaciones para sus cursos luego de una corta introducción al ambiente de trabajo de EJS. Un método alternativo**,** y uno muy prometedor, es permitir a los estudiantes modificar el modelo en una simulación

o crear sus propias simulaciones, y de esta manera, entrar en lo que los investigadores llaman: modelado constructivo.

3.3.1. Descripción de la herramienta

EJS es un software diseñado para la creación de simulaciones por computador. EJS fue desarrollado por un proyecto de física *Open source* [52] en la Universidad de Murcia, España, por el profesor Francisco Esquembre. EJS, y las simulaciones creadas con él, pueden ser usadas como programas independientes bajo diferentes sistemas operativos, o distribuidas por Internet y ejecutadas dentro de paginas *html* por la mayoría de exploradores web.

La diferencia entre EJS y los otros productos es que la herramienta no está diseñada para hacer el trabajo más fácil a los programadores profesionales, fue concebido para el beneficio de los profesores y estudiantes de ciencias que tienen más interés en el contenido de la simulación que los aspectos técnicos de cómo crear una simulación.

La mayoría de simulaciones hechas por computador utilizando EJS pueden ser descritas en términos del paradigma modelo, control y vista, ver Figura 10:

- El modelo, el cual describe el modelo bajo estudio en términos de:
    - Variables, que contienen los posibles diferentes estados del fenómeno, y
    - Relaciones de esas variables (correspondiendo a las leyes que gobiernan el fenómeno), expresados mediante algoritmos.

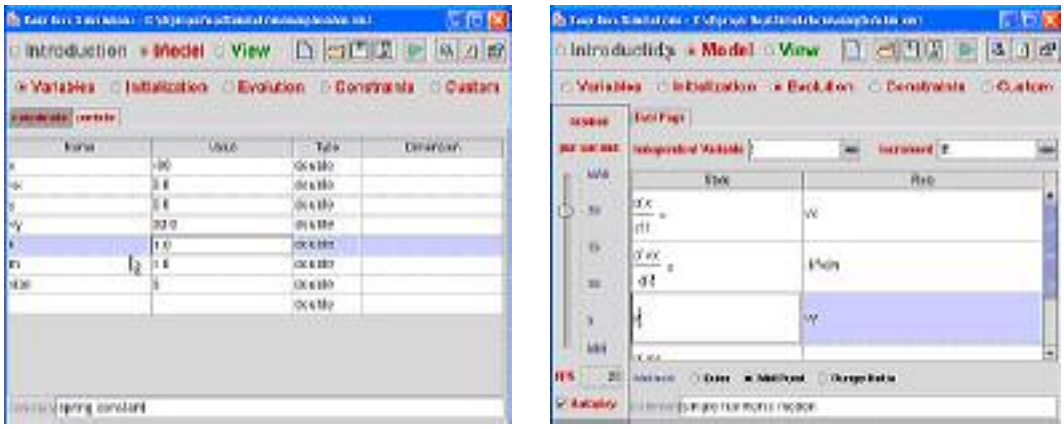


Figura 10. Interfaz Modelo.

- El control, el cual define ciertas acciones que el usuario puede desempeñar en la simulación.

- La vista, la cual muestra una representación gráfica de los diferentes estados que el fenómeno pueda tener. Estas representaciones puede ser hechas en forma real (modelo en 2D o 3D) ó esquemáticas (Gráficas tradicionales), ver Figura 11.

Figura 11. Interfaz Vista.

Las 3 partes están muy relacionadas. El modelo obviamente afecta la vista, a partir de un cambio en el estado del modelo debe ser hecho evidente gráficamente para el usuario. El

control afecta el modelo porque las acciones de control pueden (y usualmente los hacen) modificar el modelo y la vista, porque la interfaz gráfica puede contener componentes que permiten a los usuarios modificar variables o desempeñar las acciones predefinidas.

Para mayor simplicidad en la construcción de una simulación, EJS elimina la parte de control, agregando la mitad de ésta dentro de la vista y la otra mitad dentro del modelo. Actualmente, los programas de computador modernos son interactivos, lo cual significa que los usuarios pueden modificar la lógica del programa haciendo algunas acciones como hacer *clicks* con el ratón, presionando teclas, etc. Así el componente vista puede ser usado para controlar la simulación. Por otro lado, si se quiere que estas acciones tengan efecto en la simulación, debe ser definido dentro del modelo del fenómeno o dentro del componente modelo de la simulación.

Crear una simulación en EJS consiste en definir su modelo y su vista y establecer las relaciones entre ellas necesarias para:

- La correcta visualización del estado del fenómeno que está siendo simulado y
- La apropiada interacción del usuario con la vista (definir las acciones que el usuario pueden desempeñar durante la simulación).

Esta explícita separación en partes refuerza conceptualmente el rol principal del modelo de una simulación. Este es el modelo que define cuál es el programa a simular y cómo hacerlo. Además, los profesores pueden crear varios componentes vista para un mismo modelo, permitiendo así explorar diferentes aspectos de un mismo fenómeno.

EJS posee otra interfaz para cada proyecto de simulación, esta es la introducción, ver Figura 12. Es en esta interfaz donde el creador de la simulación puede agregar información importante acerca del modelo, tipo de control, propósito de simulación y demás información útil que permite entender más fácilmente la simulación. Este componente es muy útil sobre todo si se piensa en crear laboratorios virtuales en donde la presencia del profesor no puede ser permanente.

Figura 12. Interfaz de introducción.

Una de las características más importantes de EJS es que permite crear simulaciones que funcionan como archivos ejecutables u objetos embebidos en una página web independiente de EJS. Esto quiere decir que no es necesario abrir el entorno de trabajo de EJS para poder ver una simulación. Aprovechando esta cualidad se pueden crear laboratorios virtuales en páginas de Internet disponibles a cualquier persona que quiera realizar una práctica sin siquiera conocer que es EJS.

3.3.2. Requerimientos del sistema

EJS es un programa realizado en lenguaje java, debido a esto para poder ejecutar la aplicación se debe tener instalado la maquina virtual de java en el computador. Teniendo en cuenta lo anterior los requerimientos mínimos para *crear* una simulación son:

- Procesador de 1Ghz.
- Memoria Ram de 256MB.
- Windows 2000 o XP.
- Java JDK 1.5 (Maquina virtual de java).

Sin embargo, como se explicó anteriormente, EJS permite crear simulaciones que estén embebidas en páginas web o que sean ejecutables. De este modo los requerimientos mínimos para *ejecutar* una simulación son:

- Procesador de 500Mhz.
- Memoria Ram de 128 Mhz.
- Windows 2000 ó XP.
- Java JDK 1.5.

Estos requerimientos en la actualidad son bastantes básicos, la mayoría de las máquinas actuales son mucho más potentes, haciendo de EJS un software adecuado para la creación de simulaciones virtuales *portables y livianas*.

3.3.3. Easy Java Simulations con Matlab

Una de las grandes ventajas de EJS es la capacidad de comunicarse con herramientas de alto desempeño como Matlab y Simulink. Esta característica permite usar los códigos y diagramas y la gran cantidad de funciones que provee Matlab para hacer el procesamiento matemático de la simulación, ahorrando mucho tiempo en el momento de crear funciones como operaciones con matrices, solución de sistemas de múltiples ecuaciones, derivadas, etc.

Para poder trabajar en EJS con herramientas externas se debe hacer lo siguiente:

- Tener instalado la herramienta en el computador.
- Configurar la consola de java para que trabaje con herramientas externas.

Luego de cumplir con los requerimientos se puede escoger entre los dos tipos de conexión que EJS puede realizar con Matlab. La primera de ellas es la integración de código de Matlab en EJS y llamar archivos .m, la segunda es la capacidad de ejecutar un modelo de Simulink, aunque para tener disponible esta forma debe utilizarse la anterior.

3.3.4. Añadiendo código en las funciones de EJS

Para lograr la comunicación entre Matlab y EJS se deben declarar las variables a utilizar dentro de una página especial llamada página externa, lo cual significa que las variables que han sido declaradas pueden ser accedidas desde otra aplicación, ver Figura 13.

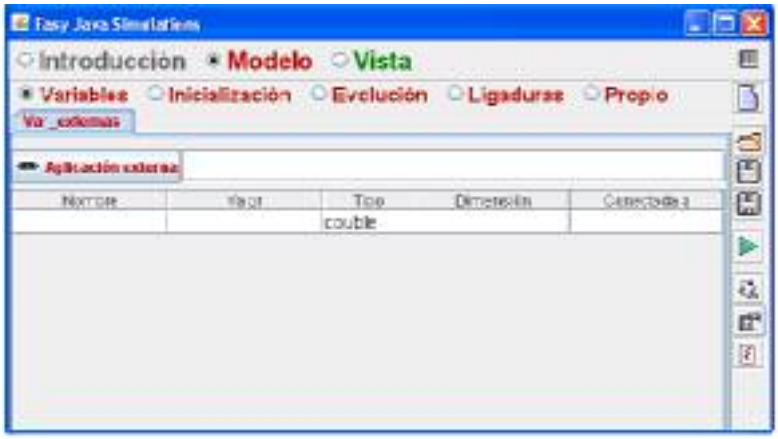


Figura 13. Vista pagina de variables externas.

Esta página es ligeramente diferente a la estándar de declaración normal. La diferencia se encuentra en el botón que aparece en el medio del lado izquierdo con la etiqueta “Aplicación Externa”. Aunque de primer momento no se va a necesitar utilizar ese botón, ahí se puede configurar el archivo .m o .mdl (simulink) que se necesita utilizar.

EJS permite integrar el código de Matlab dentro del código de las simulaciones de java como se puede observar en la Figura 14.

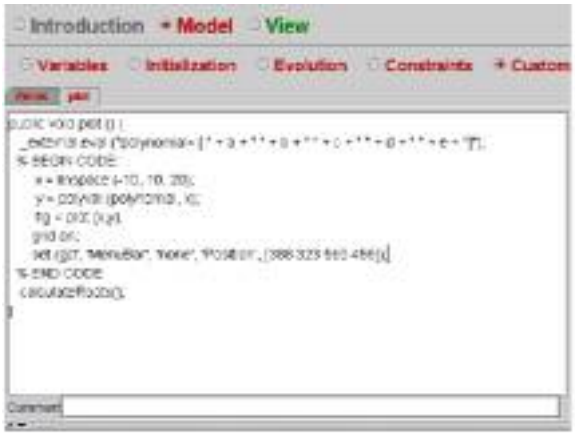


Figura 14. Código de Matlab en EJS.

La pestaña *plot* dentro de las funciones construidas por el usuario, entrega un vector llamado polynomial con los componentes a, b, c, d, e los cuales están dados en EJS.

% BEGIN CODE y % END CODE encierran el código escrito en el lenguaje matemático, usando las funciones *plot* y *linspace* que tienen una declaración en Matlab.

De esta manera podemos utilizar las funciones y características en Matlab para construir simulaciones virtuales de una manera más simple y segura que si se escribiera otra vez el código para definir un espacio lineal o escribir la función *plot* que permite dibujar en pantalla la curva de característica polinomial.

En resumen la forma de usar el código de Matlab en EJS es la siguiente.

- Usar _external para enviar y recibir la información entre las dos aplicaciones.
- Usar %BEGIN CODE y % END CODE para escribir el código de Matlab dentro de la simulación.

Cuando se corre la simulación la ventana de comandos de Matlab emergerá y se comenzará con la simulación.

3.3.5. Utilizando archivos .mdl de Simulink

Usar modelos de Simulink desde EJS es bastante simple, aunque se necesita de algún trabajo preliminar. Los pasos necesarios son:

1. Hacer unos cambios menores al modelo de Simulink para que se pueda comunicar con el espacio de trabajo de Matlab. También se aconseja quitar todos los bloques que permitan visualizar el comportamiento del modelo, en general se desea utilizar las funciones que provee EJS.
2. Crear un archivo de .m sencillo que informa a EJS acerca del modelo a ser usado y las variables y/o parámetros que pueden ser accedidos en éste, y también actúa como un mediador de comunicación.
3. Enlazar las variables en EJS a las correspondientes variables en Matlab. Se debe entender que así como se puede tener la capacidad de obtener datos de un modelo de Simulink también se pueden enviar datos, tales como condiciones iniciales y trayectorias hacia el modelo, como un conexión bilateral que expande las posibilidades de la técnica.
4. Incluir en EJS el modelo necesario para llamar los métodos que controlan la ejecución del modelo de Simulink. Los archivos de Simulink son ejecutados por la interfaz de Simulink. Para poderlos ejecutar desde otra aplicación es necesario algunas entradas desde el espacio de trabajo de Matlab para poder tener control absoluto de la ejecución del modelo.

EJS presenta unas características interesantes y muy fáciles de entender para el desarrollo de simulaciones virtuales para estudiantes de ciencias. La característica principal es que se pueden crear archivos ejecutables (jar) ó archivos visibles desde internet (*Applet*) que no necesitan que se abra la aplicación que crea la simulación. Sin embargo una capacidad fundamental es la de trabajar en conjunto con una herramienta tan poderosa como Matlab. Esta propiedad permite realizar simulaciones de una gran complejidad gracias al soporte para el tratamiento matemático que ofrece Matlab.

3.4. 20-Sim

20-Sim está compuesto por dos ventanas:

- Editor
- Simulador

Una sesión de modelado y simulación en 20-Sim puede ser representada por la Figura 15.

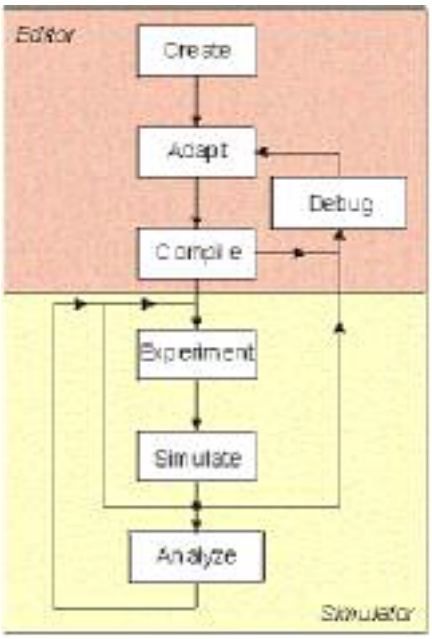


Figura 15. Modelado y Simulación en 20-Sim.

3.4.1. Editor

Los modelos son introducidos y compilados en el editor de 20-Sim. El editor es una herramienta versátil que ayuda al usuario a introducir modelos soportando una amplia variedad de sistemas incluyendo lineales, no lineales, de tiempo discreto, de tiempo continuo y sistemas híbridos, sin restringirlo a un modelo de representación particular. Después de introducir y depurar, el modelo puede ser revisado y compilado. Esto es

hecho automáticamente en el trasfondo, cuando se abre el simulador. El Editor consiste de cuatro subventanas:

- **Jerarquía**: Es posible agrupar partes del modelo como un submodelo, la jerarquía de estos es mostrada en esta ventana.
- **Implementación**: 20-Sim soporta modelos gráficos y de ecuaciones. Se pueden introducir en la subventana de Implementación.
- **Icono**: Cada modelo tiene su propio icono el cual puede ser editado.
- **Tipo**: Se puede definir la interfaz de un modelo definiendo sus puertos y parámetros.

Los sistemas pueden ser modelados en 20-Sim, usando ecuaciones, descripciones en el espacio de estado, enlaces gráficos, diagramas de bloques, y componentes o diagramas icónicos. Estas descripciones pueden ser unidas completamente para crear modelos mixtos.

20-Sim soporta modelado jerárquico ilimitado. Los niveles más altos consisten de modelos gráficos (modelos en espacio de estados, diagramas de bloques, gráficas de enlaces o componentes) y los niveles más bajos están formados por modelos de ecuaciones.

El nivel más bajo en un modelo en 20-Sim está formado por ecuaciones. Las ecuaciones en 20-Sim siguen el estándar de notación matemática y pueden ser cambiados por el usuario. Una gran colección de funciones lineales, no lineales, escalares y matriciales están disponibles para el uso de ecuaciones.

3.4.2. Archivos de Modelos

Todos los modelos 20-Sim son almacenados en archivos con la extensión .em. La información almacenada en estos archivos es basada en el lenguaje SIDOPS+ el cual ha sido desarrollado exclusivamente para 20-Sim.

La parte invisible de SIDOPS+ contiene la descripción del ícono del modelo, definiciones de submodelos, interfaces de submodelos, conexiones de submodelos, etc. El usuario podrá revisar la descripción del modelo en SIDOPS+ cuando lo guarda en formato texto.

3.4.3. Simulador

La ventana de Simulación, permite correr simulaciones y mostrar los resultados en una gráfica, además ofrece un conjunto de herramientas para analizar los resultados de la simulación.

Los resultados de la simulación pueden ser mostrados en gráficas y ventanas animadas. La ventana de Simulación principal es usada para especificar valores de la simulación y correr la simulación. Durante la simulación todas las otras gráficas y ventanas de animación son actualizadas simultáneamente.

Las gráficas son totalmente configurables. Soporta vistas logarítmicas, estilos de líneas, estilos de marcos y fondos. Las gráficas y animaciones pueden estar listas para publicación fácilmente (copiar y pegar en cualquier documento).

3.4.4. Archivos de Datos y Compilación

20-Sim permite almacenar la simulación en un archivo de datos.

Para almacenar o leer los datos de la simulación, primero deben definirse las variables que serán almacenadas y el nombre del archivo que será usado; después de definir estas características, se puede correr la simulación durante la cual los datos son almacenados.

En cuanto a la compilación, 20-Sim puede operar en dos modos: Modo Depurar y Modo Rápido. En el Modo Depurar se harán todas las posibles revisiones y las advertencias serán generadas para los posibles errores del modelo.

En el Modo Rápido se construye la compilación con el tiempo corriendo, el cual compila el modelo de simulación en una plataforma especifica con código de maquina de 32 bits. El resultado es un crecimiento dramático de la velocidad de simulación. El código de maquina de 20-Sim corre más rápido que la compilación equivalente en código C compilando el código de máquina, incluso con grandes modelos. Esto es hecho mientras el usuario pone en marcha el Simulador. El compilador es una parte interna del software 20-Sim. No se requiere un programa o compilador externo.

3.4.5. Toolbox 3D Mechanics y Toolbox Real Time

El Toolbox 3D Mechanics consiste de un editor. Este es un programa independiente para introducir modelos dinámicos 3D de una manera amigable y generar un código 20-Sim.

El Editor 3D Mechanics puede ser iniciado desde el Editor de 20-Sim. Desde el menú **Herramientas**, se debe seleccionar *Toolbox 3D Mechanics* y *Editor 3D Mechanics*.

Con el Editor de 3D Mechanics se pueden crear modelos dinámicos 3D de estructuras mecánicas. Para simular estos modelos, se deben exportar a 20-Sim. En 20-Sim se pueden agrupar el modelo 3D Mechanical con otros componentes.

Se puede hacer uso del Toolbox Animation 3D para mostrar una animación de la estructura mecánica. Es posible exportar la estructura a un archivo escenario, el cual puede ser cargado en el Toolbox Animation 3D.

Por su parte, el Toolbox Real Time de 20-Sim permite crear código C fuera de cualquier modelo 20-Sim para el uso en aplicaciones en tiempo real:

- Crear código ANSI-C para el uso en un ambiente en tiempo real.
- Crear S-Functions en Matlab/Simulink para el uso en el ambiente tiempo real de Matlab.

La generación de código Matlab/Simulink puede ser realizado usando el comando *Exportar* del menú **Editor Archivo**.

3.4.6. Archivos AVI

Para usar Animaciones 3D en aplicaciones Multimedia, es posible crear un archivo AVI. Para esto, es necesario abrir el menú **Archivo** desde la ventana *Propiedades 3D* y seleccionar **Crear Archivo AVI**.

Varias opciones pueden ser elegidas:

- **Filename**: Introduce el nombre del archivo avi.
- **Color Depth**: Selecciona la cantidad de colores que deberán ser usados.
- **Frame Selection**:

  - Real Time: Si esta opción es elegida, los marcos serán mostrados en tiempo real. El número de marcos que son usados desde la simulación, dependen de la rata de marcos que el usuario ha elegido. El factor Speed-Up puede tener cualquier valor para mostrar películas a alta o baja velocidad.

  - **Write All Frames**: Si esta opción es elegida, todos los marcos resultado de los puntos de simulación, serán almacenados. La velocidad de la película depende de la velocidad de simulación (cuántos puntos de simulación y por lo tanto marcos, fueron hechos cada segundo) y la rata de marcos.

- **Video Format**: Los archivos AVI pueden ser creados en formatos de video Standard, el tamaño actual de la ventana o un formato definido por el usuario.
- **Frame Rate**: El número de marcos por segundo.

En el fondo de la ventana se muestra el número de marcos, el tiempo resultante de película, y se da una estimación del número de bytes del archivo completo. El tamaño exacto del archivo depende del tipo de compresión y del factor de compresión.

Cuando se da *click* en el botón Crear AVI, se abre un dialogo que permite elegir el tipo de compresión y el factor de compresión.

3.4.7. Exportar a Matlab

Existen diferentes formas de exportar modelos 20-Sim y datos al paquete Matlab/Simulink.

- **Exportar al entorno Matlab**

Para exportar modelos 20-Sim a Matlab, desde el Editor se debe seleccionar el menú **Archivo** y dar clic en el comando *Exportar a Matlab/Simulink*, esta acción abrirá un dialogo donde se elegirá exportar a Matlab ó Simulink.

Si el deseo del usuario es exportar a Matlab, podrá elegir exportar el modelo completo (main model) o un submodelo. En ambos casos, se generan tres archivos:

- ModelName.m: Contiene el modelo de ecuaciones y las funciones especiales de 20-Sim. El modelo de ecuaciones es trasladado directamente desde las ecuaciones que son mostradas con el comando *Mostrar Ecuaciones* del menú **Modelo**.
- ModelName_run.m: Un archivo ejemplo muestra como correr una simulación con el modelo exportado a 20-Sim.
- ModelName_print.m: Este archivo contiene una función que es usada en el archivo run.

Si se desea simular el modelo, deberá seleccionarse el comando Exportar a Simulink.

Para exportar modelos 20-Sim a Matlab y Simulink, son usadas algunas plantillas especiales. Estas plantillas pueden ser adaptadas para exportar a otros paquetes.

- **Exportar a Simulink**

Los modelos 20-Sim son exportados a *S-Functions* en Simulink. Las *S-Functions* son elementos de diagramas de bloques cuya descripción interna puede ser *m-file* o *dll-file*. 20-Sim puede exportar ambo tipos. *M-files* pueden ser abiertos desde Matlab y por lo tanto son más accesibles. *dll-files* son compilados fuera de Código-C, lo que los hace inaccesibles, pero un poco más rápidos.

Simulink no soporta puertos de potencia. Por lo tanto los puertos de potencia en un modelo 20-Sim son trasladados a puertos de entradas y salidas.

- **Exportar a Simulink (m-file)**

Para exportar modelos 20-Sim a Simulink con una descripción *m-file*, desde el Editor se debe seleccionar el menú **Archivo** y dar clic en el comando Exportar *a* Matlab/Simulink. Esta acción abrirá un dialogo donde se elegirá exportar a Matlab ó Simulink.

Si el deseo del usuario es exportar a Simulink, podrá elegir exportar el modelo completo (main model) o un submodelo. En ambos casos, se generan dos archivos:

- ModelName_.mdl: El modelo 20-Sim exportado, basado en un m-file que describe el modelo.
- ModelName.m: Un m-file que contiene el modelo de ecuaciones y funciones especiales de 20-Sim. El modelo de ecuaciones es directamente trasladado desde las ecuaciones que son mostradas con el comando *Show Equations* del menú **Modelo**.

- **Exportar a Simulink (dll-file)**

Para exportar modelos 20-Sim a Simulink con una descripción m-file, desde el Simulador se debe seleccionar el menú **Herramientas** y dar clic en el comando *Generación Código C*, esta acción abrirá un dialogo donde se elegirá exportar a Simulink.

Para la generación de Código C, el usuario solo puede seleccionar exportar un submodelo 20-Sim a Simulink. Si la opción OK es seleccionada, se generará un Código C ANSI y se compilará en una dll-function. Algunos archivos serán generados de los cuales dos son necesarios en Simulink:

- ModelName_.mdl: El modelo 20-Sim exportado, basado en un dll-file que describe el modelo.
- ModelName.dll: Un dll-file que contiene el modelo de ecuaciones y funciones especiales de 20-Sim. El modelo de ecuaciones es directamente trasladado desde las ecuaciones que son mostradas con el comando *Show Equations* del menú **Modelo**.

3.4.8. Generación de código C

20-Sim tiene un generador de Código C el cual automáticamente convierte un modelo completo o submodelo de 20-Sim en código C. La aplicación puede ser usada para generar S-Functions Matlab/Simulink, para generar ejecutables Stand-Alone o para generar funciones entrada/salida para usar en otro programa C y C++. El generador de código ANSI C, puede ser abierto desde el Simulador (menú **Herramientas**, comando *Generación de código C*)

3.4.9. Versiones de 20 Sim

20-Sim actualmente se encuentra disponible en tres versiones: Viewer, Standard y Professional.

- **Viewer**: Esta es una versión libre que permite al usuario cargar y correr modelos y realizar una evaluación del paquete. No se permite grabar modelos.
- **Standard**: Esta versión no tiene limitaciones pero solo incluye la ToolBox de Dominio en Frecuencia.
- **Professional**: Esta es la versión completa de 20-Sim con todas las Toolbox.

Las Tablas 7 y 8 muestran en detalle las opciones disponibles en las tres versiones:

Tabla 7. Librerías disponibles en las tres versiones de 20-Sim.

Librerías	Viewer	Standard	Professional
Bond Graph	×	√	√
Signal / Block Diagram	×	√	√
Electric	×	√	√
Hydraulics	×	√	√
Translation	×	√	√
Rotation	×	√	√
Thermal	×	√	√

Tabla 8. Toolbox disponibles en las tres versiones de 20-Sim.

Toolbox	Viewer	Standard	Professional
3D Mechanics Toolbox	×	×	√
Animation Toolbox	×	×	√
Control Toolbox	×	×	√
Frequency Domain Toolbox	×	√	√
Mechatronics Toolbox	×	×	√
Real Time Toolbox	×	×	√
Time Domain Toolbox	×	×	√

3.5. SELECCIÓN DE LA HERRAMIENTA SOFTWARE

Las herramientas estudiadas en la sección anterior presentan muchas características parecidas. En cuanto a los requerimientos del proyecto las dos herramientas tienen lo que se necesita. Pero luego de un estudio más profundo de las capacidades que tiene cada software de simulación, teniendo en cuenta los 4 tópicos principales de selección del presente proyecto se tiene las siguientes conclusiones.

- En cuanto a recursos computacionales las dos herramientas son muy buenas. Las dos tienen un bajo uso de memoria Ram como de tiempo de procesador.

- 20-sim es un software propietario por lo que tiene un costo. EJS es parte de un proyecto de software libre y gratis por lo que no hay que pagar ni por su uso ni por su distribución.

- 20-sim es un software que le lleva mucho en cuanto a facilidad de uso a EJS. Es de recordar que 20-sim es el predecesor de TUTSIM, por lo que ya lleva años de desarrollo. Gracias a este tiempo 20-sim es un programa muy completo y eficiente.

- La distribución es una de las características más importantes que hace de EJS una buena herramienta. La posibilidad de crear archivos ejecutables y *Applets* de las simulaciones esta muy por encima de la capacidad de crear videos que tiene 20-Sim.

Las características son muy parecidas pero existen 2 factores que hacen que EJS sea la herramienta elegida. La primera es el costo y la segunda es la capacidad de generar *Applets* lo cual facilita de gran manera el desarrollo de un laboratorio virtual a través de la web.

Por lo tanto, se escoge a **Easy Java Simulations** como la herramienta ideal para el desarrollo de la simulación virtual del robot SCARA del presente proyecto.

3.6. DISEÑO Y DESARROLLO DE LA SIMULACIÓN

3.6.1. Diseño de la simulación

Para el diseño de la simulación se partió del modelo en diagramas de bloques que tiene el sistema el cual es:

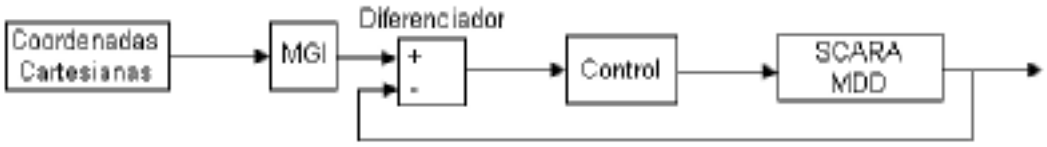


Figura 16. Modelo en diagrama de bloques de la simulación.

El modelo representado en la Figura 16 muestra el típico diagrama de bloques que representa un modelo dinámico y su control. Se escogió un modelo que obtuviera la señal de entrada en coordenadas cartesianas y un controlador para coordenadas articulares, de ahí el bloque MGI. A continuación se da una explicación más profunda de cada uno de los bloques del diagrama:

Coordenadas Cartesianas: Este bloque representa la señal de entrada del sistema, son las coordenadas cartesianas que el programador quiere que el robot reproduzca, es posible que otro diseñador escoja un modelo con señales de entrada articulares, cambia un poco el modelo pero los resultados deben ser muy similares. En otras palabras, este bloque entrega la trayectoria que debe seguir el robot en un espacio de 3 dimensiones.

MGI: Este bloque es necesario para poder convertir la trayectoria deseada de coordenadas cartesianas a coordenadas articulares. Contiene el modelo geométrico inverso del robot SCARA, el cual recibe las coordenadas cartesianas en que se desea tener a la articulación final y entrega las posiciones articulares de cada una de las articulaciones para que se cumplan los requerimientos del diseñador..

- Señal de entrada: Coordenadas cartesianas.

- Señal de salida: Coordenadas articulares.

Diferenciador: Este bloque es fundamental para tener un sistema controlado, ya que permite hacer la diferencia entre las coordenadas que se desean y las coordenadas articulares en las cuales se encuentra actualmente al robot.

- Señal de entrada: Coordenadas articulares deseadas, Coordenadas articulares actuales del robot.

- Señal de salida: Diferencia entre las coordenadas articulares, el error articular. (Señal de referencia).

Control: Para el presente proyecto se escogieron dos tipos de controladores para realizar la simulación; el PID y el CTC. Estos controladores son de los más utilizados en la industria, además de que existe mucha literatura sobre ellos facilitando el trabajo de investigación. Luego de que se aplica el control a la señal de referencia ésta deja de ser un error articular para convertirse en el torque deseado.

- Señal de entrada: Señal de referencia.

- Señal de salida: Señal controlada (Torques).

SCARA MDD: Este bloque representa el modelo dinámico directo del robot. Aquí se recibe la señal de torque deseado enviado por el controlador para que el modelo la procese y entregue la aceleración de cada una de las articulaciones. Para obtener la velocidad y la posición actual del robot se debe integrar la aceleración. Cabe recordar que el controlador recibe la diferencia de las posiciones articulares, por lo que esta conversión es muy necesaria.

- Señal de entrada: Torques.

- Señal de salida: Aceleración articular.

Es importante tener dentro de la vista la información necesaria para seguir y configurar el seguimiento del controlador.

- La principal razón de este proyecto es mostrar un modelo virtual de un robot SCARA que permita observar el comportamiento de éste mientras desarrolla las

trayectorias deseadas. Por lo tanto es muy importante tener en cuenta que muestre el modelo en 3D del robot SCARA.

- Para el diseñador es importante tener información gráfica de las señales de salida, seguimiento de la trayectoria, errores articulares y cartesianos.

- Debe haber unos controles (botones, *slides* o campos numéricos) que permita configurar la constantes de los controladores, configurar la posición inicial y tamaño de las trayectorias, como también poder detener o reiniciar la simulación en cualquier momento.

Todos estos detalles son importantes ya que permitirán que la simulación sea realmente útil y fácil de usar tanto para profesionales en robótica como estudiantes.

Teniendo una descripción de cada uno de los bloques además de tener las señales de entrada y salida ya se puede tener una idea de las funciones necesarias para crear la simulación, de ésta manera se prosigue con el desarrollo de la simulación.

3.6.2. Desarrollo de la simulación

Teniendo en cuenta los módulos que se obtuvieron en el diseño de robot se comenzará con el desarrollo de cada uno de los módulos. Antes de proseguir se explicarán algunos términos necesarios para comprender mejor el desarrollo de la simulación.

Tiempo de ejecución: es el tiempo en que la simulación está corriendo, cuando el sistema se encuentra activo.

Ciclo del programa: Es el ciclo que se repite una y otra vez hasta terminar la simulación. Este ciclo ejecuta el diagrama de bloques de la Figura 19 hasta cumplir el tiempo de simulación.

Tiempo de simulación: es el tiempo total que va desde el inicio de la simulación hasta el final.

Tiempo de muestreo: Valor que representa el tiempo entre cada iteración del ciclo del programa. En otras palabras, cada cuánto se realiza un paso de la simulación. El tiempo de muestreo se fijó en 0.001 segundos (1 milisegundo).

3.6.3. Coordenadas cartesianas

Para generar los puntos de las coordenadas cartesianas que conformarán las trayectorias que el robot va a seguir, se crearán ecuaciones matemáticas que entregarán los puntos en tiempo de ejecución (durante la simulación).

Cada trayectoria tiene su conjunto de ecuaciones agrupadas por funciones, de esta manera es posible utilizar cada recorrido simplemente incluyendo su función dentro del ciclo del programa. Para cambiar el tamaño de cada trayecto se hizo que se pudiera multiplicar por un valor proporcional, esto es, si se deja el tamaño en 1, entonces la trayectoria será del tamaño que esté estipulado en las ecuaciones, pero si es cambiado a 2 el tamaño será el doble del tamaño que este definido en las funciones de trayectoria.

a. Trayectoria lineal

Esta trayectoria es una línea recta con la ecuación Y= X + inix, donde iniX e iniY son las posiciones iniciales en coordenadas cartesianas en las cuales el usuario quiere ubicar al robot (ver implementación en anexo B).

Las variables xent, yent y zent guardan la posición que le entrega la función en cada ciclo del programa (ver anexo1). La variable tamRobot permite cambiar el tamaño del desplazamiento. Temp es la variable que lleva la cuenta del tiempo de simulación. Tal como están expresadas las ecuaciones, esta función representa una trayectoria lineal en 2D (el eje z se encuentra fijo).

Esta función hace que el robot se desplace tanto en el eje X como en el Y, de 0.0001cm cada iteración del ciclo de programa, por lo tanto en 1 segundo, que es el tiempo de simulación se obtendrá un desplazamiento de 14.1 cm (hipotenusa), siempre y cuando el tamaño sea 1.

b. Trayectoria circular

Esta trayectoria genera un círculo a partir del punto entregado como centro y el tamaño del radio que se le ha dado por defecto es de 2cm.

Emplea 3 segundos de simulación para hacer el recorrido total. De igual forma que en la trayectoria anterior, ésta solo mueve al robot en dos dimensiones X y Y, pero no el eje Z (ver implementación en Anexo B).

c. Trayectoria triangular

Para conformar el triangulo se utilizó tres veces la ecuación de la trayectoria lineal modificando solamente el inicio y la dirección para poder conformar el triángulo isósceles.

Se puede apreciar que las trayectorias a seguir se dividieron en tres grupos, dependiendo del tiempo de simulación. El primer grupo desde el inicio hasta el primer segundo de simulación. Durante el siguiente segundo se opera otro grupo de ecuaciones, y, finalmente se cierra el triángulo, desde el segundo 2 hasta el 4.

Solo es una trayectoria de dos dimensiones, no se mueve en el eje Z y el tiempo de simulación es de 4 segundos (ver implementación en Anexo B).

d. Pick-and-Place

Representa el movimiento que haría un SCARA real para recoger un objeto, transportarlo y dejarlo en otro sitio.

Es bastante larga pero el funcionamiento es similar a la triangular, con la excepción de que esta trayectoria si mueve la cuarta articulación en el eje Z.

El funcionamiento que describe el código para recrear este trayecto es el siguiente.

1. Inicialmente el robot baja y se dispone a recoger un objeto.

2. Después, recoge el objeto y lo sube hasta la posición en que estaba inicialmente.

3. Acto seguido el robot trasporta el objeto en el plano X-Y y se detiene.

4. Por último, el SCARA baja su articulación prismática ubicando al objeto en su posición final.

Todo este recorrido toma un tiempo de simulación de 4 segundos (ver implementación en Anexo B).

3.6.4. Conversión de coordenadas cartesianas a coordenadas articulares

Para realizar esta conversión se utilizó el modelo geométrico inverso del robot SCARA, el cual determina las posiciones articulares de acuerdo a la posición cartesiana del efector final.

La función toma las coordenadas cartesianas que se le entregan al sistema. Las ecuaciones utilizadas para desarrollar este modelo se pueden encontrar en la biografía [34].

En la parte final se ve que los resultados son guardados en el vector *q*. Este vector guarda la información de las coordenadas articulares deseadas, necesaria para que funcione el controlador PID.

Un detalle importante de este modelo es el uso de la función *atan2* para calcular el valor de la *arcotangente* de dos valores. Esta función tiene la ventaja de que permite obtener los resultados de la operación trigonométrica en los cuatros cuadrantes y no solo en dos como lo hace *atan* (ver implementación en Anexo B).

3.6.5. Diferenciador

La diferencia de las posiciones articulares se realizó dentro de cada uno de los controladores, por lo que no se realizó función para este bloque.

3.6.6. Controladores

Como se dijo anteriormente se escogieron dos tipos de controladores para desarrollar la simulación. Cada uno de ellos representa una función distinta, pero no confundir con bloques diferentes. Durante la simulación se debe escoger un control.

3.6.7. Controlador PID

La función de este controlador de dividió en 4 partes. La primera de ellas es la diferencia entre las posiciones articulares deseadas y las obtenidas.

```
proporcional[0] = q[0] - x0;
proporcional[1] = q[1] - x1;
proporcional[2] = q[2] - x2;
proporcional[3] = q[3] - x3;
```

Seguido, se realizó el control proporcional multiplicando la diferencia de las señales por las constantes proporcionales mediante un ciclo.

```
for(int a=0;a<=3;a++)
        {
        diferencia[a] = proporcional[a];
```

```
proporcional[a]*= Kp[a];
}
```

Se creó un vector denominado diferencia para guardar este valor para la parte integral.

El control derivativo se realizó derivando las posiciones articulares deseadas (sin realizar la diferencia con las obtenidas), luego se realizó la diferencia con las velocidades obtenidas en el instante inmediatamente anterior (x4, x5, x6, x7), para terminar multiplicando el resultado por las constates derivativas.

```
// Derivativa

derivativa(q); // Se

double[] derivativo=new double[4];

derivativo[0] = qres[0]-x4;
derivativo[1] = qres[1]-x5;
derivativo[2] = qres[2]-x6;
derivativo[3] = qres[3]-x7;

for(int b=0;b<=3;b++)
        {
        derivativo[b]*= Kv[b];
        }
```

Fue necesario crear una función que permitiera derivar las coordenadas articulares para obtener la velocidad deseada. Esta función se explicará más adelante en el punto Funciones de Utilería.

El control integral se desarrolló de la siguiente manera:

```
double[] integral=new double[4];

integral (diferencia);

for(int k=0;k<=3;k++)
        {
        integral[k]=IntRes[k]*Ki[k];
        }
```

De manera muy similar al control derivativo, se integró la diferencia de coordenadas articulares y luego se multiplicaron por las constates del control integral.

Por último se sumaron el resultado de los tres controles, y así obtener el torque requerido para el funcionamiento del modelo (ver implementación en Anexo B).

```
for(int m=0;m<=3;m++)
        {
        Torque[m]=proporcional[m]+integral[m]+derivativo[m];
        }
```

3.6.8. Controlador CTC

La implementación de este control es similar a tener un controlador PD. Inicialmente se encuentra la diferencia entre la trayectoria articular deseada y la obtenida (ver implementación en Anexo B).

```
p[0] = q[0] - x0;
p[1] = q[1] - x1;
p[2] = q[2] - x2;
p[3] = q[3] - x3;
```

Se estima la velocidad deseada derivando las posiciones articulares y se obtiene la diferencia.

```
//Obtengo la velocidad deseada y cálculo la diferencia entre la deseada y la obtenida
derivativa(q);

double[] v=new double[4];

v[0] = qres[0]-x4;
v[1] = qres[1]-x5;
v[2] = qres[2]-x6;
v[3] = qres[3]-x7;
```

Se multiplica las diferencias de las velocidades y posiciones por las constantes. Y se obtiene la aceleración sumando estos dos resultados.

```
//Multiplica a cada uno por los valores de las constantes
for(int a=0;a<4;a++)
{
        v[a]*=Kp[a];
        p[a]*=Kv[a];
}
double[] wt= new double[4];
for(int a=0;a<4;a++)
{
        wt[a] = p[a] + v[a];
}
```

Luego de obtener las señales de control de velocidad, posición y aceleración, se envía esta información al modelo dinámico inverso del robot SCARA. Al igual que el modelo directo éste también ha sido generado por el software SYMORO+ [34].

Después de que el modelo inverso procese la información se obtiene las señales de torque que deben agregarse al modelo dinámico directo.

3.6.9. Modelo

El modelo que se utilizó en esta simulación fue creado utilizando el programa SYMORO+ (ver implementación en Anexo B). Una *S-function* describe el modelo de un sistema por medio de ecuaciones de estado, por lo tanto no fue muy difícil adaptarlo para que funcionara en EJS, ya que éste último también tiene la posibilidad de procesar sistemas expresados en tiempo “continuo”.

La implementación del modelo entrega de una vez la posición y la velocidad en lugar de la aceleración.

El código de este modelo no es muy intuitivo pero es funcional y tiene tantas ecuaciones que se muestra el modelo a manera de información (ver Figura 17).

Para poder calcular en cada instante las variables de estado del modelo se añadieron las igualdades necesarias en la sección evolución de EJS diseñada para este uso específico.

Figura 17. Cálculo de la posición y velocidad del modelo en cada iteración del programa.

3.6.10. Salida del sistema

Luego de obtener el comportamiento del sistema es necesario observar los resultados de manera analítica, es por eso que se hace necesario convertir las posiciones articulares a posiciones cartesianas para poder dibujar la trayectoria que el robot en realidad realizó. Estas funciones fueron tomadas de las matrices de transformación del Modelo Geométrico Directo (mgd) del SCARA.

Esta conversión también es necesaria para poder ubicar los elementos tridimensionales en el espacio de trabajo.

Cada una de las articulaciones tiene dos variables que guardan su posición en los ejes X y Y. Solo la cuarta articulación tiene movimiento en el eje Z, de acuerdo a la naturaleza del robot SCARA.

3.6.11. Funciones de Utilería

Para realizar la simulación fue necesario construir funciones, que aunque no están descritas en el modelo general, eran muy necesarias. Estas funciones son:

a. Función integradora

Implementar este método era necesario para poder calcular el control integral. La función que se utilizó fue la de hallar un promedio entre la señal actual y la señal anterior y dividirla entre el tiempo de muestreo (ver implementación en Anexo B).

$$ci = \frac{qant - qact}{2} \qquad (3.1)$$

Con lo que se encuentra el punto medio de la función en ese instante, el resultado se multiplica por el tiempo de muestreo y se hace la sumatoria de ese valor:

$$\sum ci * delta \qquad (3.2)$$

Donde delta es igual al tiempo de muestreo (0.001).

b. Función derivativa

Este método deriva una señal discreta, necesario para obtener la velocidad deseada de las posiciones deseadas. La ecuación que permite hacer esto es la siguiente:

$$Derivada = \frac{qact - q1ant}{delta} \qquad (3.3)$$

La derivada resulta de restar la posición de la anterior iteración a la posición actual. Este resultado es dividido entre el tiempo de muestreo, y al final se obtiene la derivada o pendiente entre dos puntos (ver implementación en Anexo B).

c. Calcular los ángulos

Para representar el robot en un modelo de 3D es necesario conocer los ángulos entre cada articulación. El modelo dinámico del robot entrega la posición de cada una de las articulaciones, sin embargo, es necesario dibujar elementos que conecten estas articulaciones. Para poder obtener los ángulos que tienen las articulaciones 1 y 2 y 2-3 en determinado momento fue necesario implementar esta función.

d. Función de mensajes

Este método maneja los errores más comunes que pueden suceder durante la simulación. Una posición por fuera del área de trabajo o tamaño de la trayectoria demasiado grande son algunos de los errores que se encuentran codificados.

Esta función es llamada cada vez que se desea iniciar una simulación, por lo tanto, siempre revisa los valores y los errores que pueden suceder antes de iniciar la simulación (ver implementación en Anexo B).

Se han establecido siete errores conocidos, para cada uno de estos errores se envía una notificación al usuario y se posiciona el robot en un tamaño y posición válida, para luego reiniciar la simulación.

e. Función reiniciar

Cada vez que el usuario quiera iniciar la simulación cargando los nuevos datos de configuración la función reiniciar debe ser llamada.

Su funcionamiento principal es configurar en cero el tiempo y tomar los datos de la vista, con la función _initialize() del entorno de simulación, para configurar la nueva simulación (ver implementación en Anexo B).

Todos estos métodos fueron agregados en la sección de Propio de EJS. Esta sección está diseñada para que el usuario pueda agregar sus métodos propios, ver Figura 18.

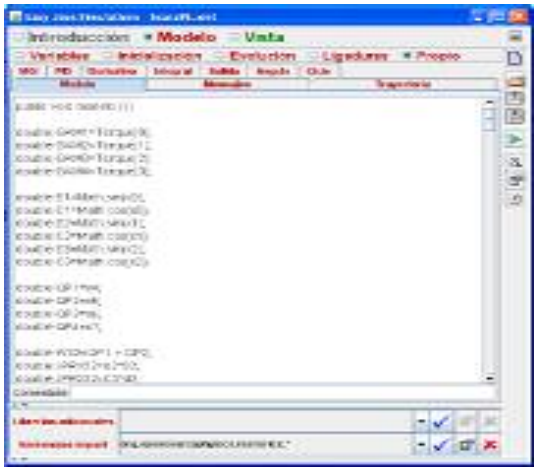


Figura 18. Pestañas de EJS.

Cada una de las pestañas representa por lo menos una función. Hay pestañas como trayectoria que poseen más de una función. La utilidad de usar varias pestañas es la de organizar el contenido de la simulación, de esta manera es mucho más fácil de depurar y de entender para terceros.

3.6.12. Inicialización

EJS permite configurar una página con las condiciones iníciales de la simulación. Sin embargo, este código se fue convirtiendo más en una función que en una simple igualación de variables a valores estáticos. Fue necesario construir una inicialización dinámica, que permitiera al usuario configurar la simulación a sus necesidades (ver implementación en Anexo B).

Antes que nada la inicialización no debe declararse como método, EJS obtiene el código puesto en esta página y lo organiza en un método por defecto. Solamente con escribir las líneas de configuración dentro de la sección inicialización es suficiente.

Esta parte configura la posición inicial que debe tener el robot de acuerdo a la trayectoria y la posición inicial que le indica el usuario.

Cuando el usuario desea utilizar la trayectoria circular y selecciona un punto para que se realice el trayecto, en realidad selecciona el centro del círculo y no la posición inicial desde donde debería iniciar el robot. Debido a este pequeño inconveniente, fue necesario adecuar la posición inicial del robot cuando se quiere que realice una trayectoria circular.

Para los otros recorridos esto no es necesario, así que se pasa la posición inicial directamente.

Con la capacidad de ubicar al robot de acuerdo a la trayectoria deseada se permite que el robot tenga un inicio suave, simulando con si viniera de una trayectoria más larga, y no, como un salto brusco desde cero hasta las posiciones deseadas.

Los estados que representan las velocidades, al igual que las variables que guardan los resultados del control de la función integradora y derivativa, deben ser configurados en cero. Esto asegura que si hubo un error en la simulación anterior, el usuario puede configurar nuevamente la aplicación y continuar con las pruebas.

Cada trayectoria tiene un tiempo de simulación distinto, por eso es importante determinar que tiempo de simulación se va a tener en cuenta cada vez que el usuario quiera cambiar de trayecto.

Las constantes del controlador PID se inicializan dependiendo de los valores que el usuario agregó en la vista. De esta manera la simulación puede ir actualizando su comportamiento.

3.6.13. Desarrollo de la vista

Como se dijo anteriormente, el objetivo principal de utilizar EJS para simular un robot es la poder mostrar un modelo en 3D, el cual permita representar el comportamiento del sistema mientras se encuentra siguiendo alguna trayectoria.

También se habló en el diseño de ofrecer las gráficas analíticas necesarias para el trabajo de diseñadores o estudiantes de robótica. Estas gráficas son el seguimiento de la trayectoria, y los errores tanto articulares como cartesianos.

3.6.14. Ventana principal

La ventana principal es el objeto por el cual la maquina virtual de Java empieza a ejecutar la aplicación. Aunque este comportamiento no es posible observarlo desde el entorno de EJS, se puede inferir de acuerdo al comportamiento de las aplicaciones Java.

La ventana principal posee una barra de menú. Los menús que se encuentran disponibles son Simulación, Trayectoria, Tipo de control y Ventanas. Cada uno con sus respectivos ítems permiten realizar las siguientes acciones sobre la simulación:

Menú Simulación: Aquí se encuentran las opciones que permiten detener, reanudar, reiniciar y detener la simulación.

- Play: Permite proseguir con la simulación cuando ésta se ha detenido manualmente.

- Pause: Para detener la simulación en cualquier momento.

- Actualizar: Cuando el usuario realiza una nueva configuración de la simulación (cambio en las constates del controlador, selección de otra trayectoria), este botón permite reiniciar la simulación teniendo en cuenta la actual configuración.

- Reset: Cuando el usuario desee comenzar desde cero o en caso de que se produzca un error grave, siempre se cuenta con la opción de reset, la cual permite reiniciar la aplicación cargando los parámetros por defecto.

Menú Trayectoria: Permite seleccionar entre las cuatro trayectorias que trae la simulación de manera predeterminada. Estas son:

- Una línea recta.
- Un círculo.
- Un triángulo isósceles.
- Una trayectoria que muestra como el robot transportaría un objeto de un lugar a otro.

Menú Tipo de control: La simulación soporta 2 tipos de control para el robot, una es el clásico controlador PID y la otra es el control CTC, mediante este menú se puede escoger el uso de cualquiera.

Ventanas: En algunas ocasiones es necesario o conveniente cerrar o abrir alguna ventana de información, o por que simplemente se cerró accidentalmente y es necesario volver a revisar los resultados obtenidos. De esta manera este menú permite hacer visible

o invisible los resultados de la simulación, por ejemplo, si se está desarrollando el ajuste con el control PID, no se necesita tener visible la ventana que permite configurar el control CTC.

La ventana principal también contiene la simulación del robot SCARA en un espacio 3D, ver Figura 19. El modelo tridimensional ha sido configurado por medio del editor de 3D de EJS, con solo unos ajustes por medio de código.

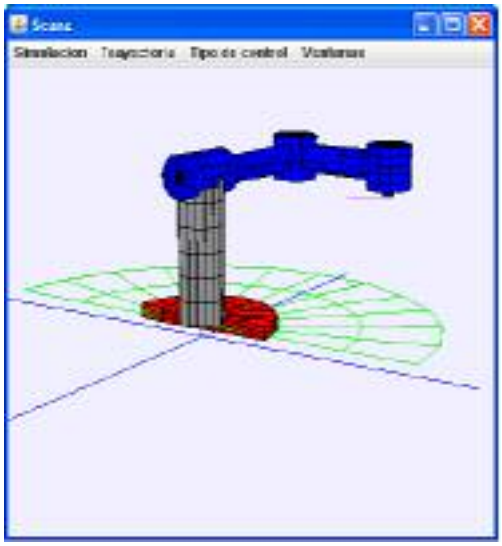


Figura 19. Visualización del SCARA.

El espacio en 3D posee un gráfico el cual señala la dirección de los ejes X, Y y Z. Esto permite que el usuario pueda ubicarse dentro del plano tridimensional mientras intenta cambiar el ángulo desde el cual se observa al robot.

Como la ventana de simulación es la ventana principal de la aplicación, es de esperarse que cada vez que se cierre esta ventana se dará por terminada la simulación, esto no sucede con las demás ventanas y diálogos que posee la simulación.

3.6.15. Ventana de trayectoria

Para determinar si el robot siguió la trayectoria cartesiana deseada, con más precisión que en la vista del modelo 3D, se creó una ventana que gráfica tanto la trayectoria deseada como la obtenida, de esta manera es mucho más rápido determinar si hubo o no éxito en la sintonización de los controladores.

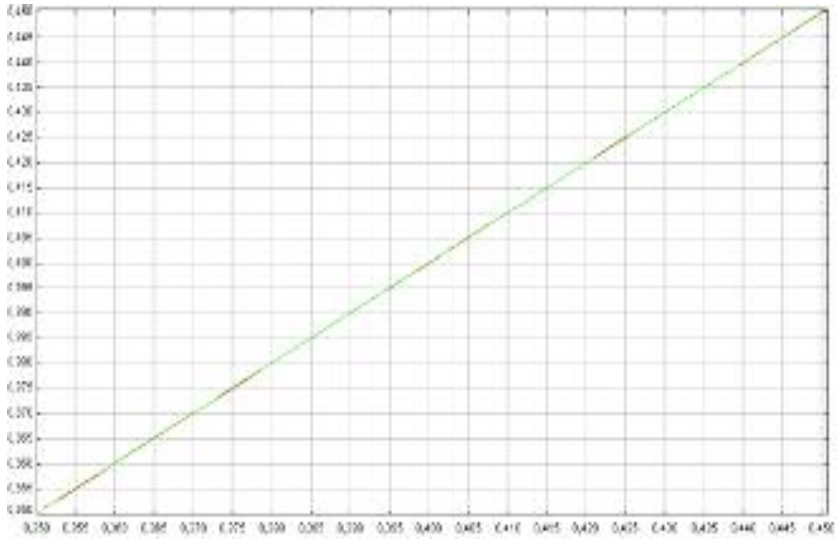


Figura 20. Trayectoria lineal.

La Figura 20 muestra en verde el trayecto que el diseñador quiere realizar con el robot. En rojo se gráfica el resultado del sistema.

Esta ventana puede ser cerrada sin terminar la simulación, además posee un control para ocultar o mostrar, el cual se verá más adelante.

3.6.16. Ventana de errores

Más importante aún que una gráfica que muestre el resultado entre el recorrido deseado y el recorrido obtenido, es la información de los errores o la cuantificación de los errores tanto en el espacio cartesiano como en el articular.

Con esta información, el diseñador puede tomar decisiones importantes sobre la sintonización del controlador y la precisión que tiene la aplicación en desarrollo.

En la Figura 21 se observa el error cartesiano de seguimiento de las trayectorias, mientras que en la Figura 22 se puede ver el Error Articular del SCARA.

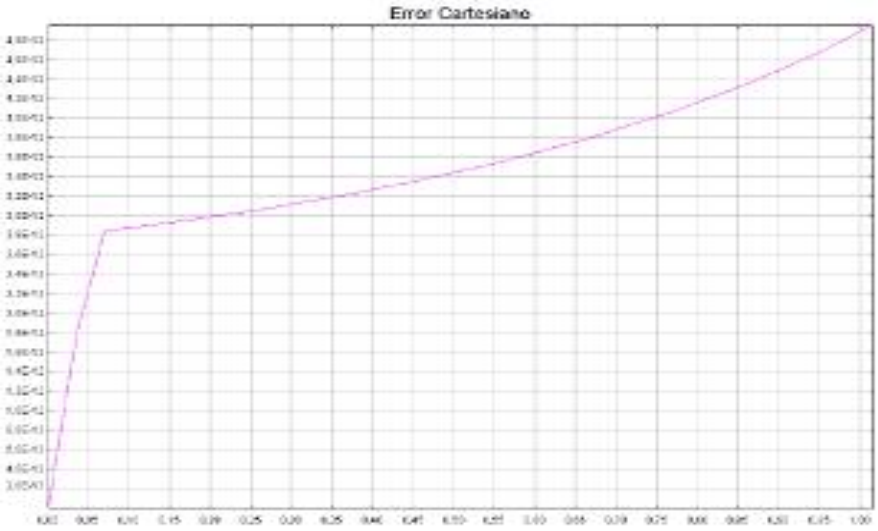


Figura 21. Error Cartesiano.

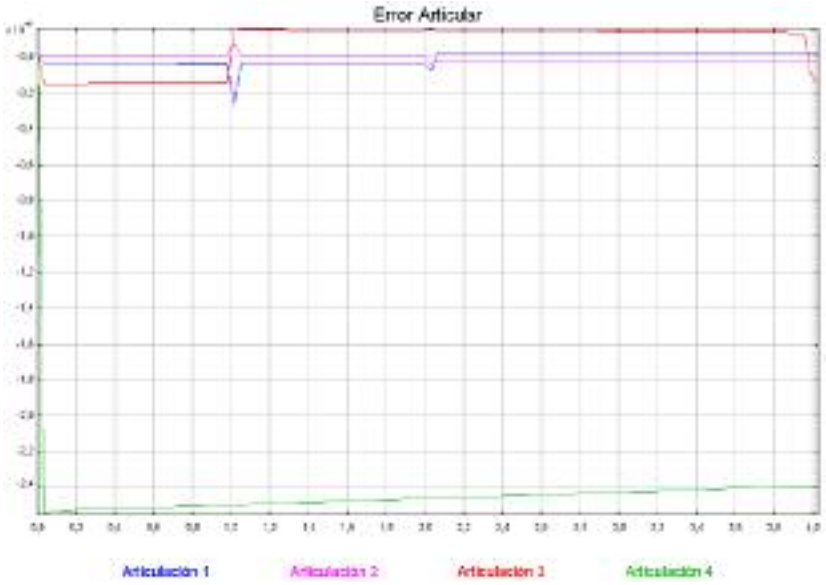


Figura 22. Error Articular.

3.6.17. Dialogo de configuración del PID

En la Figura 23 se observa la ventana de configuración del PID la cual posee 12 campos numéricos para la configuración del PID. Cada campo está relacionado con una variable interna, las cuales permiten modificar las constantes de los tres controles (proporcional, integral y derivativo).

Figura 23. Configuración del PID.

3.6.18. Dialogo de configuración de la trayectoria

Por medio de éste es posible reposicionar en un lugar cualquiera el punto inicial siempre y cuando esté dentro del área de trabajo del robot.

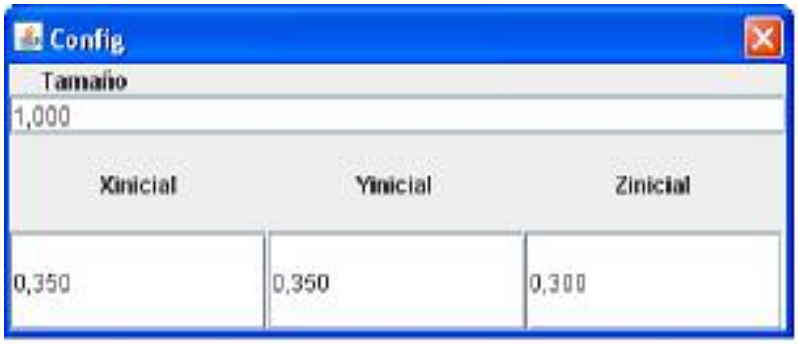


Figura 24. Configuración de la Trayectoria.

Además de la posición inicial, es posible cambiar el tamaño de la trayectoria por medio del control numérico llamado tamaño, como se observa en la Figura 24. La variable tamaño representa un control de escala que se le puede hacer a una trayectoria, pero no representa el tamaño real. Por ejemplo la trayectoria lineal tiene una longitud de 14.1cm, pero modificando la variable tamaño del dialogo de 1 a 2, la nueva longitud de la trayectoria será 28.2cm (2 * 14.1cm), o dos veces más grande que la normal.

3.6.19. Errores de simulación

Existen algunos problemas que podrían presentarse durante la simulación que dejarían en riesgo la estabilidad de la aplicación. Es por eso que se creó una función que se encarga de verificar la viabilidad de obedecer las instrucciones del usuario. Evitar que el usuario quiera posicionar la articulación final del SCARA por fuera del área de trabajo es uno de estos errores posibles. La función determina si la trayectoria a realizar está dentro de los límites, de lo contrario informa al usuario sobre el fallo.

Para advertir se utiliza un dialogo muy sencillo, el cual posee un texto y un botón de aceptar, ver Figura 25. Esta dialogo de información es creado por medio de la función de EJS denominada _alert (“Element”, “title”, “advertencia”).

Figura 25. Ventana de indicación de error.

4. RESULTADOS OBTENIDOS

En este capítulo se muestran los resultados obtenidos luego de la investigación y el diseño de la aplicación. Se comparan los resultados obtenidos con la simulación de referencia para determinar así el desempeño de la nueva herramienta frente a una que ya ha sido aceptada.

Para tener plena seguridad de que la implementación del robot virtual es confiable y que su uso obtiene resultados similares con la simulación de referencia, se realiza una validación teniendo en cuenta el promedio del índice del error al cuadrado en el espacio cartesiano.

Es de esperar que la configuración de cada una de las simulaciones deba ser igual para tener unos resultados confiables. Cada controlador es probado con 2 trayectorias distintas (lineal y circular), además de que cada una se probará con dos diferentes tiempos de muestreo para tener más información del comportamiento de la planta. En total se realizarán 8 pruebas de validación. Como convención se utilizarán los puntos rojos como la trayectoria obtenida y los puntos verdes como la deseada.

4.1. INDICE DE ERROR

Es necesario, para determinar el desempeño de la simulación, el poder deducir de manera concreta qué implementación consigue mejores resultados en cuanto al seguimiento de la trayectoria. Para eliminar los criterios subjetivos que puedan afectar dicha comparación, se utiliza un indicador, un valor fijo que permite hacer comparaciones muy objetivas.

El índice del promedio del error al cuadrado evalúa la diferencia entre el valor deseado y el valor obtenido, ver ecuación (4.1) [33]. De esta manera se puede inferir que a menor valor del índice hubo un mejor seguimiento de la trayectoria.

$$E = \frac{1}{N}\sum_{i=0}^{N}(pd(k) - po(k))^2 \tag{4.1}$$

El error es encontrado para cada uno de los ejes en el espacio cartesiano, principalmente en X y en Y, dada la naturaleza de las trayectorias [33]. Sin embargo, para obtener un solo valor que permita representar el seguimiento de toda la trayectoria de forma general se sumarán los resultados de cada eje y se les hará un promedio.

$$E_x = \frac{1}{N}\sum_{i=0}^{N}(xd(k) - xo(k))^2 \tag{4.2}$$

$$E_y = \frac{1}{N}\sum_{i=0}^{N}(yd(k) - yo(k))^2 \tag{4.3}$$

El error total sería entonces:

$$E_t = \frac{E_x + E_y}{2} \tag{4.4}$$

De esta manera se obtiene un indicador que representa el comportamiento total del robot al tratar de seguir la trayectoria en los dos ejes (X y Y), el cual se denominará el promedio del error al cuadrado. De igual manera que los índices para cada eje, el *Et* representa el error del robot al tratar de seguir la trayectoria deseada, por lo que, a menor índice de error mejor será el seguimiento de la trayectoria. Cabe anotar que este índice carece de unidades de medición, no se puede establecer que sean centímetros o milímetros, es solo un valor.

4.2. VALIDACION DE LA SIMULACION CON CONTROL PID

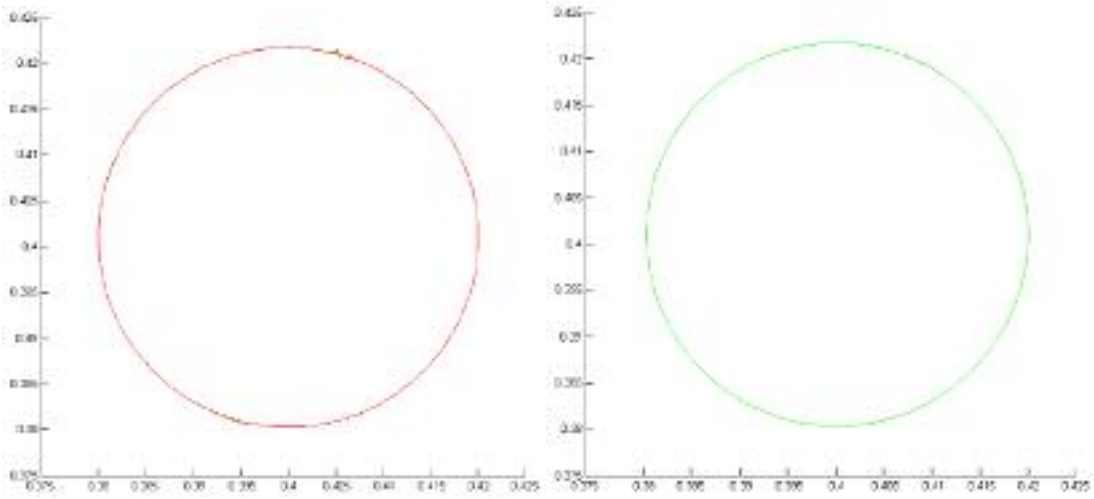
Para la implementación del PID, se sintonizará el controlador con las constantes, [33]:

```
kp1=210000;
kp2=80000;
kp3=40000;
kp4=70000;

kv1=1500;
kv2=200;
kv3=60;
kv4=500;

ki1=1000;
ki2=1000;
ki3=1000;
ki4=1000;
```

Para ver la implementación del controlador tanto en Matlab como en EJS ir al Anexo A y B respectivamente.

- **Validación de la Trayectoria Circular con Control PID**

En primer lugar se pondrá a prueba la trayectoria circular con el controlador PID, con dos valores para el Tiempo de Muestreo *h*, ver Figura 26.



a) b)

Figura 26. Trayectoria circular con un controlador PID, h=0.001 a) en Matlab, b) en EJS.

El índice para estas dos simulaciones se muestra en la Figura 27:

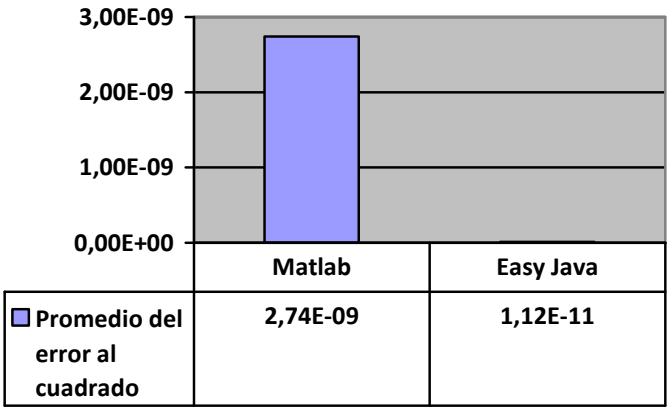
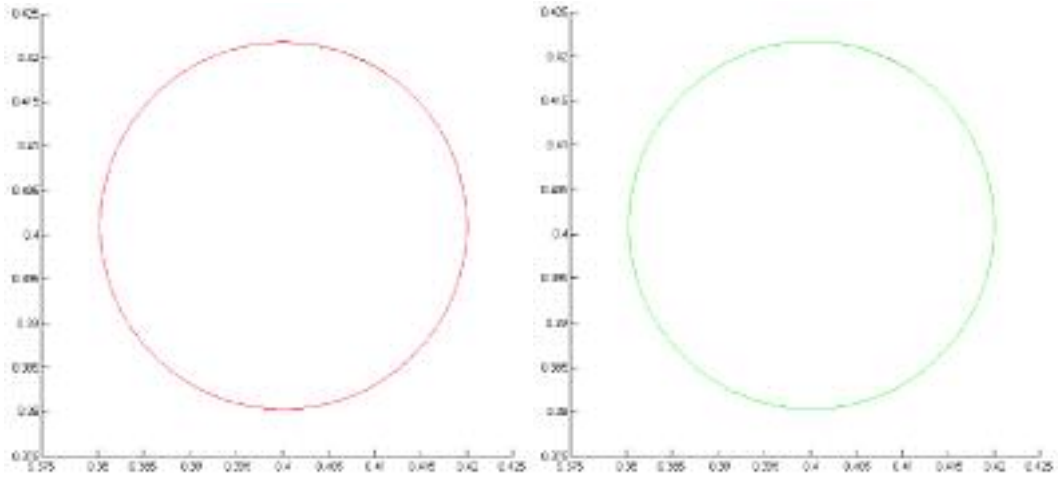


Figura 27. Promedio de error al cuadrado para la trayectoria circular utilizando un controlador PID con h=0.001.

Se repite el experimento pero esta vez el tiempo de muestreo se fija en 0.0001, ver Figura 28.



a) b)

Figura 28. Trayectoria circular con un controlador PID h=0.0001, a) en Matlab, b) en EJS.

El índice para estas dos simulaciones se muestra en la Figura 29:

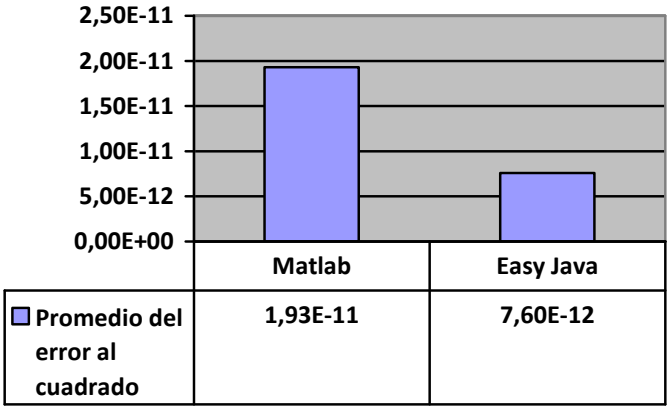
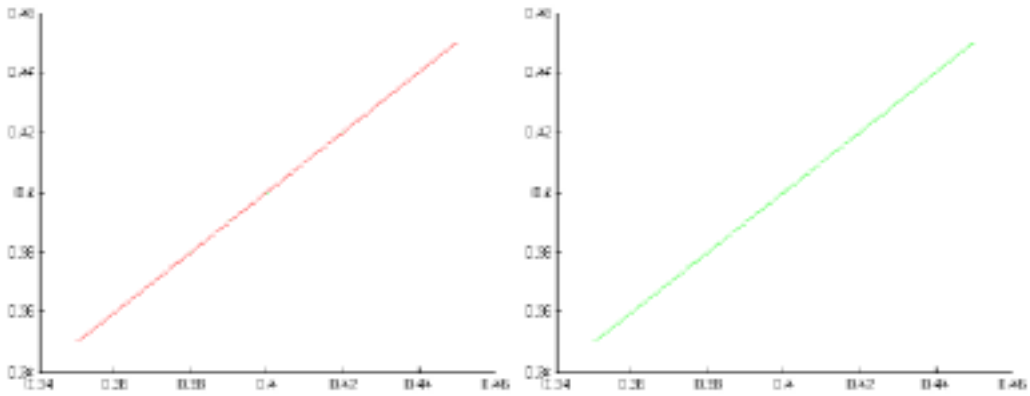


Figura 29. Promedio de error al cuadrado para la trayectoria circular utilizando un controlador PID con h=0.0001.

Las pruebas mostraron un mejor desempeño de la implementación en EJS. Al variar el tiempo de muestreo se pudo comprobar un menor índice de error en las dos, sin embargo, la aplicación virtual siempre fue superior con esta trayectoria.

- **Validación de la Trayectoria Lineal con Control PID**

Esta trayectoria es más simple de realizar y requiere menos esfuerzo del controlador. La línea tiene una longitud de 14.1cm y el tiempo de simulación es de 1s, ver Figura 30.



a) b)

Figura 30. Trayectoria lineal con un controlador PID, h=0.001, a) en Matlab, b) en EJS.

El índice para estas dos simulaciones se muestra en la Figura 31:

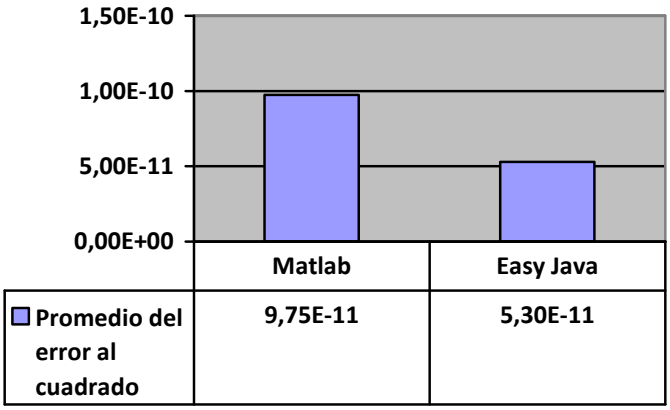
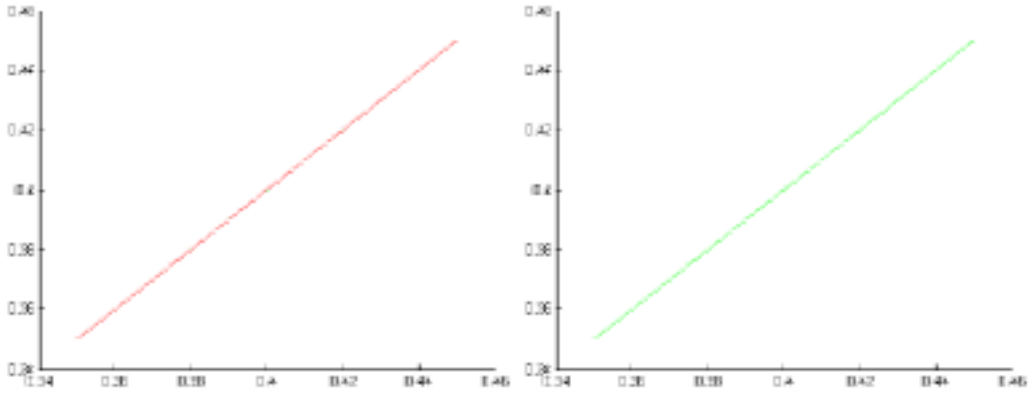


Figura 31. Promedio de error al cuadrado para la trayectoria lineal utilizando un controlador PID con h=0.001.

De nuevo la prueba pero con un tiempo de muestreo 10 veces más pequeño, ver Figura 32.



a) b)

Figura 32. Trayectoria lineal utilizando un PID, h=0.0001, a) en Matlab, b) en EJS

El índice para estas dos simulaciones se muestra en la Figura 33:

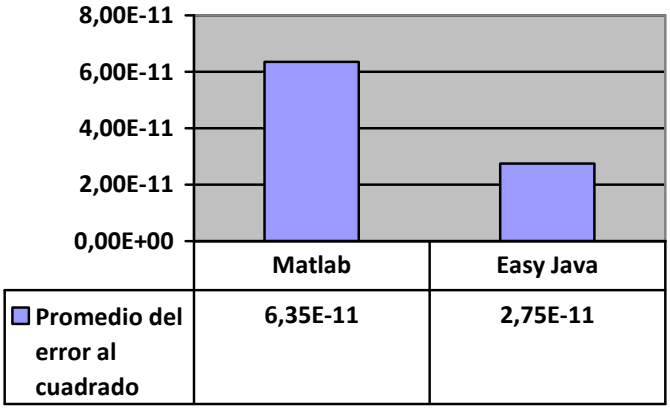


Figura 33. Promedio de error al cuadrado para la trayectoria lineal utilizando un controlador PID con h=0.0001.

La línea es una trayectoria más “fácil” de seguir por lo que los errores son más bajos que los indicadores obtenidos en la anterior trayectoria. En cuanto a desempeño la simulación en EJS sigue siendo mejor.

4.3. VALIDACIÓN DE LA SIMULACION CON CONTROL CTC

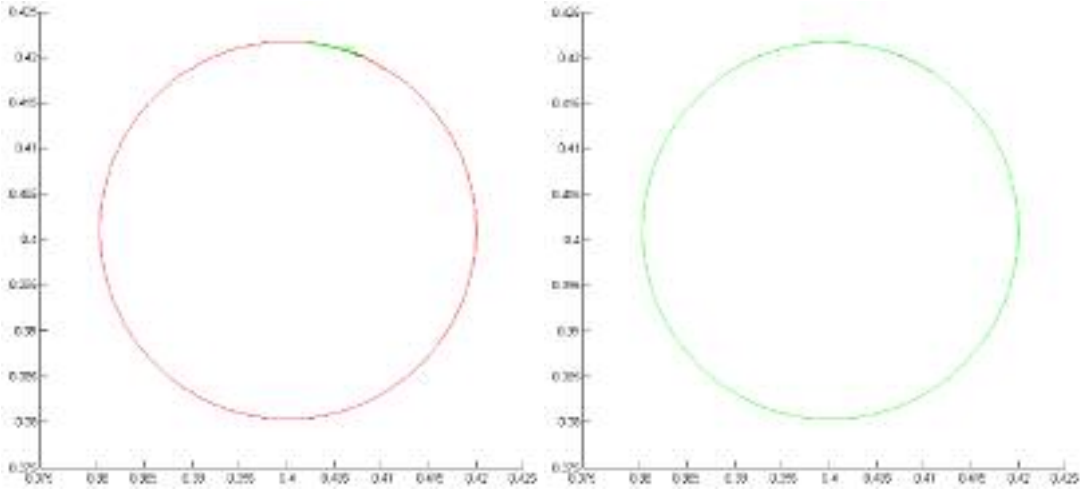
En esta sección se pone a prueba el controlador CTC implementado en EJS. Se realizaron las mismas pruebas que con el anterior controlador obteniéndose los siguientes resultados.

El controlador fue sintonizado con las siguientes constantes:

```
%proporcional

kp1=320000;
kp2=280000;
kp3=200000;
kp4=120000;

%derivativa

kv1=1200;
kv2=850;
kv3=1200;
kv4=1500;
```

- **Validación de la Trayectoria Circular con Control CTC**

Fijando a h=0.001s, se obtuvo los siguientes resultados, ver Figura 34.



a) b)

Figura 34. Trayectoria circular con un controlador CTC, h=0.001, a) en Matlab, b) en EJS.

El índice para estas dos simulaciones se muestra en la Figura 35:

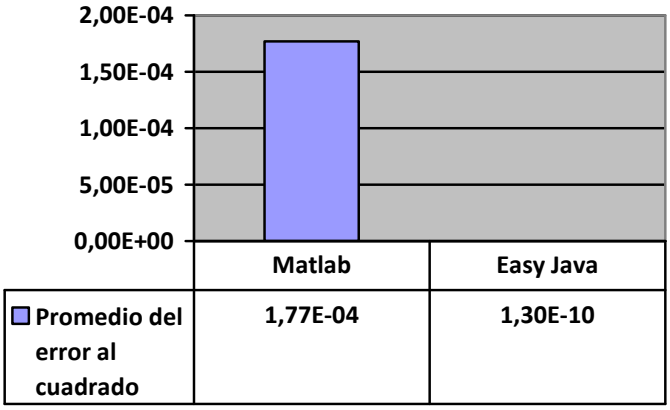
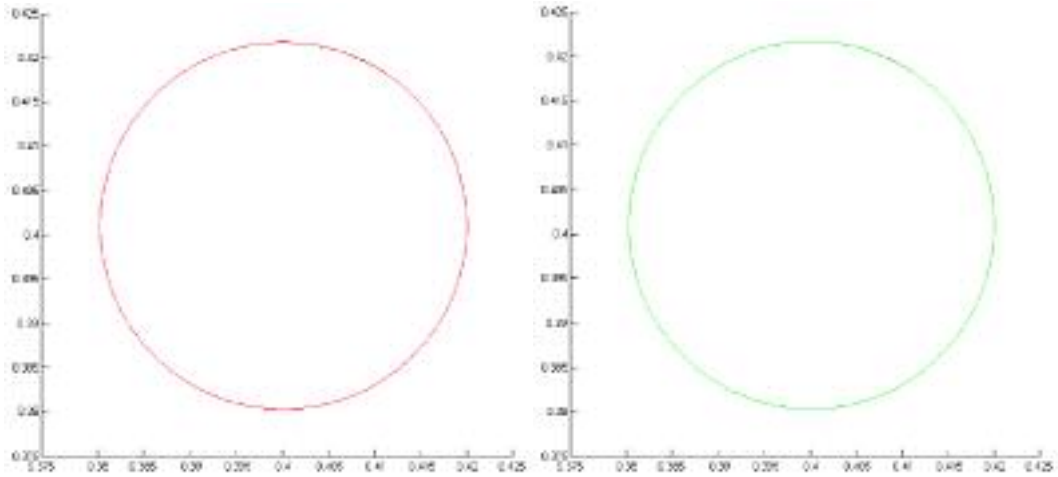


Figura 35. Promedio de error al cuadrado para la trayectoria circular utilizando un controlador CTC con h=0.001.

En esta prueba la diferencia es muy grande, esta vez es notable solo con observar la ventana de trayectoria. Se prosigue con las pruebas pero con un menor tiempo de muestreo, ver Figura 36.



a) b)

Figura 36. Trayectoria circular con un controlador CTC, h=0.0001, a) en Matlab, b) en EJS.

El indicador del promedio del error al cuadrado revela la información de la Figura 37:

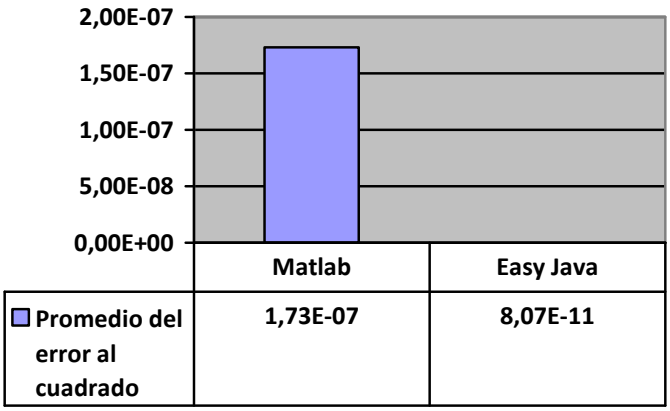
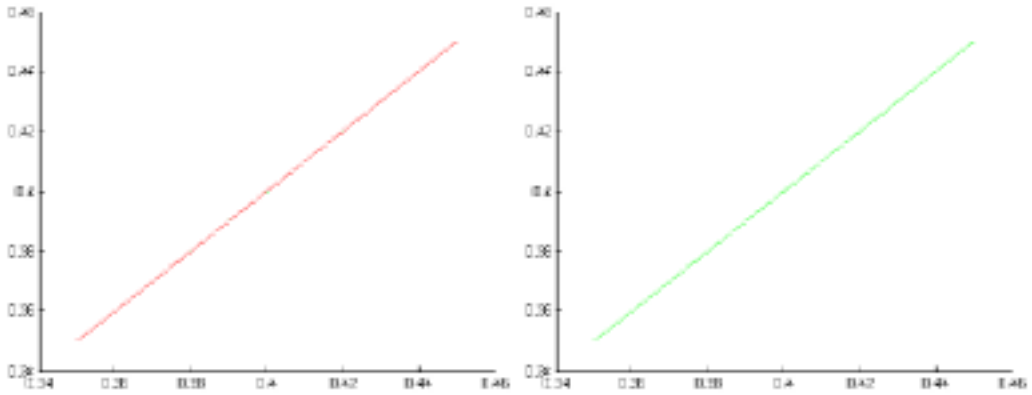


Figura 37. Promedio de error al cuadrado para la trayectoria circular utilizando un controlador CTC con h=0.0001.

Aunque el error disminuyó significativamente en la simulación en Matlab, no fue suficiente para igualar siquiera el rendimiento en la simulación realizada con EJS.

- **Validación de la Trayectoria Lineal con Control CTC**

Realizando las pruebas con el tiempo de muestreo en 0.001s se tienen las siguientes trayectorias, ver Figura 38.



a) b)

Figura 38. Trayectoria lineal con un controlador CTC, h=0.001, a) en Matlab, b) en EJS.

El índice para estas dos simulaciones se muestra en la Figura 39:

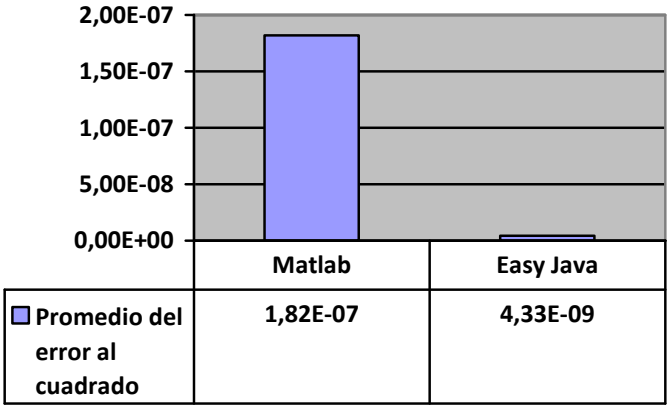
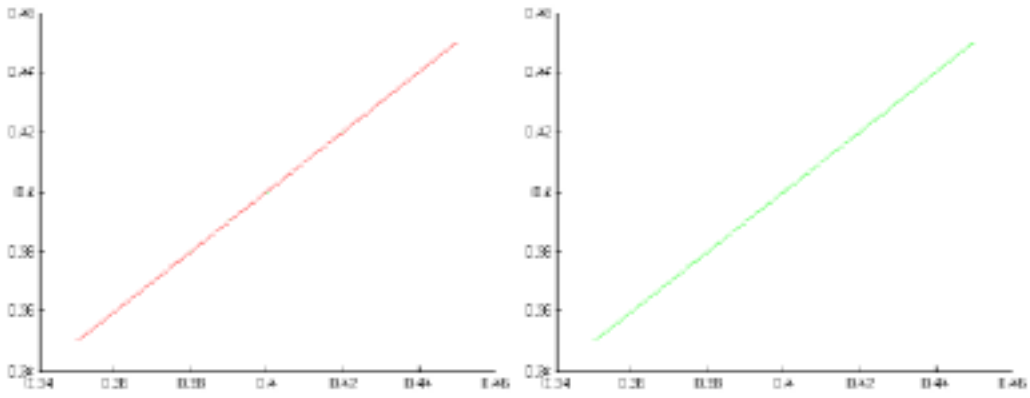


Figura 39. Promedio de error al cuadrado para la trayectoria lineal utilizando un controlador CTC con h=0.001.

Realizando las pruebas con el tiempo de muestreo en 0.0001s se tienen las siguientes trayectorias, ver Figura 40.



a) b)

Figura 40. Trayectoria lineal con un controlador CTC, h=0.0001, a) en Matlab, b) en EJS.

El índice para estas dos simulaciones se muestra en la Figura 41:

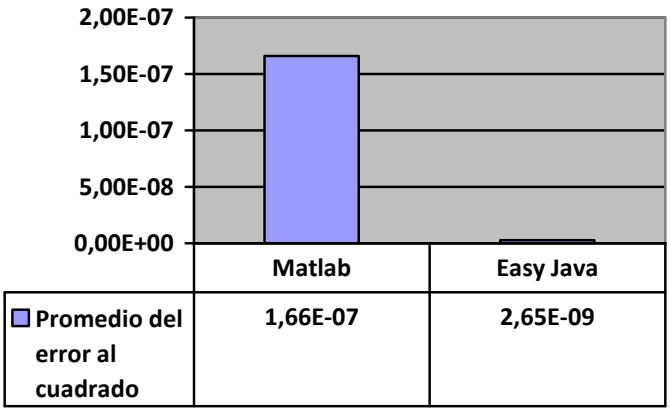


Figura 41. Promedio de error al cuadrado para la trayectoria lineal utilizando un controlador CTC con h=0.0001.

La última prueba confirma el resultado de las anteriores. El desempeño de la implementación de EJS funciona mucho mejor que la de Matlab. Este tema será tratado en la sección “Análisis de resultados”.

4.4. OTRAS TRAYECTORIAS

Aunque no se consideraron para realizar la validación de la herramienta, la aplicación construida en EJS dispone de dos trayectorias más para simular, la trayectoria triangular y la acción Pick-and-Place.

La trayectoria triangular es la unión de tres trayectorias lineales distribuidas espacialmente para que coincidan y formen un triángulo en el espacio cartesiano, ver Figura 42. El tiempo de simulación es de 4 segundos y el triangulo formado tiene dos lados iguales, también llamados isósceles.

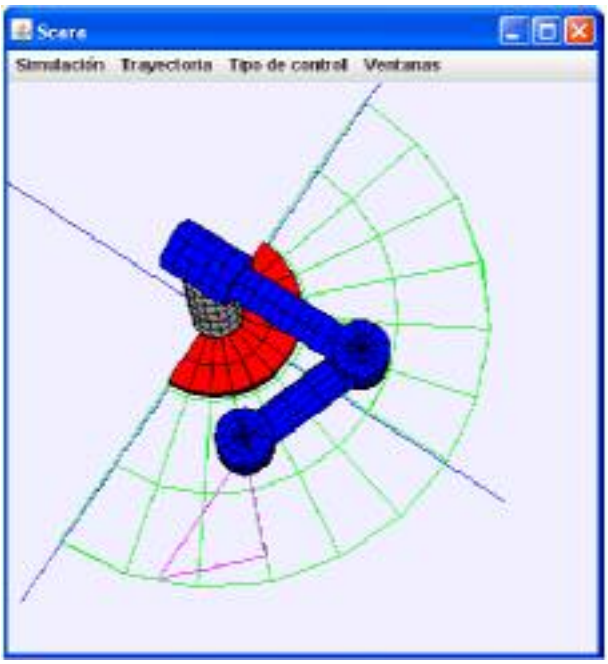


Figura 42. Trayectoria triangular en el espacio virtual.

La acción Pick-and-Place muestra la forma en que un SCARA transportaría un objeto de un lugar a otro, sin embargo como utiliza el espacio tridimensional no es visible claramente en dos dimensiones.

4.5. ANALISIS DE RESULTADOS

Las pruebas arrojaron información importante acerca del diseño y la implementación que se desarrolló durante el presente proyecto. El uso de los indicadores fue bastante útil para poder determinar concretamente el desempeño de cada una de las simulaciones en sus diferentes configuraciones.

Luego de realizadas las 8 pruebas se puede observar claramente que la implementación del modelo virtual (Easy Java) fue mejor en todos los casos, que el rendimiento de la simulación de referencia.

Estos resultados fueron desconcertantes, se supone que la simulación de referencia debería tener un menor error precisamente por ser la de referencia. Para llegar al fondo del asunto se decidió investigar con las siguientes pruebas.

- En primer lugar se dudó sobre la implementación hecha para medir el índice de error, el cual podría mostrar un resultado falso del desempeño de la simulación. Al evaluar nuevamente el código en los dos ambientes de simulación no se encontró ningún error.

- Se intentó ir un poco más atrás, no medir el resultado de la simulación sino medir los valores que se generan durante la simulación, esto con el fin de hacer una identificación más profunda y encontrar los factores que hacen la diferencia. Para ello se compararon los valores diagonales de la matriz dinámica *A(q)* del robot en las dos simulaciones. Esta prueba mostró una diferencia menor pero apreciable. Sin embargo, en EJS como en Matlab la implementación del modelo se realiza de la misma manera.

- Siguiendo con el rastreo del error se decidió verificar las trayectorias cartesianas y la conversión de las mismas al espacio articular. Nuevamente se apreció la diferencia entre los valores de las dos simulaciones.

- Se pudo observar además que los valores en EJS siempre son menores, en términos absolutos, algo que permitía suponer el tipo de error conocido.

Luego de analizar los resultados se excluyó del problema la implementación de los modelos. Además teniendo en cuenta que los valores en comparación tenían una diferencia en crecimiento a medida que la simulación avanzaba, se pudo definir el error como redondeo y/o truncamiento.

El error de redondeo se origina porque, en el ordenador, la recta real se aproxima por los números racionales con un sólo número finito de dígitos (en sistema binario), lo que implica que muchos cálculos se realizan con representaciones aproximadas de los números verdaderos, pues el ordenador sólo utiliza un pequeño subconjunto del conjunto de los números reales. Dicho de otra manera no hay una representación uno a uno del sistema binario al sistema decimal, y viceversa cuando se utilizan decimales, por esto solo se logra una aproximación [53].

La otra causa se denomina el truncamiento, el cual consiste en despreciar los dígitos del valor que exceden el tamaño que el programa puede manejar, haciendo que el valor pierda precisión. Las dos situaciones dependen de las características del ordenador donde se ejecuta la simulación y el software que se use para el mismo fin [53].

Se pudo determinar el valor de redondeo de cada uno de los programas. Para Matlab el valor más pequeño que puede diferenciar a dos valores es 2.2204 e -16, por otro lado el mismo valor para la plataforma Java es 1.4 e -45, en la cual se puede observar una gran diferencia en la precisión. Los anteriores valores fueron obtenidos de la siguiente forma. En Matlab el valor de redondeo se encuentra en la constante EPS. Para Java se deduce de la función Math.nextAfter(0,1) la cual entrega el siguiente valor posible de 0 en la dirección del valor 2, o en el eje positivo. Hay que tener en cuenta que estos valores

pueden variar, como se dijo anteriormente, dependiendo de la máquina en la que se ejecutan las aplicaciones.

Así se pudo encontrar que los factores que inciden en las discrepancias entre los resultados de las diferentes herramientas fueron el redondeo y/o el truncamiento, errores completamente numéricos pero independientes de las aplicaciones creadas y comparadas.

Se debe recordar que el objetivo principal de este proyecto es encontrar y realizar una forma de simulación alternativa a la que normalmente se hace en Matlab, y no reemplazar esta última herramienta. Por este motivo no se modificó la simulación realizada en Matlab para obtener una mayor precisión de palabra y obtener un valor parecido al de EJS.

El objetivo de realizar las pruebas era de verificar que el comportamiento de la nueva herramienta implementada en el presente trabajo era confiable y que tenía un comportamiento similar a la simulación de referencia. Y luego de realizar todas las pruebas, se concluye que la nueva simulación virtual creada en EJS es completamente confiable y VÁLIDA para realizar diseño y sintonización de controladores.

4.6. SIMULACIÓN EN LA RED

Uno de los objetivos principales de este proyecto es buscar una herramienta que permita simular a un robot SCARA en un ambiente virtual a través de una intranet y tener así un laboratorio virtual.

Con EJS esto es posible gracias a la capacidad de crear *Applets* y archivos .jar de la simulación. Estas dos opciones permiten ejecutar la aplicación desde un navegador web con capacidad para Java2 o descargarla como una aplicación cualquiera.

Para tener una idea más acertada se dan las explicaciones de lo que es un Applet y un archivo .jar.

4.6.1. Applet

Un ***applet*** es un componente de una *aplicación* que corre en el contexto de otro programa, por ejemplo un navegador web. El *applet* debe correr en un *contenedor* que lo proporciona un programa anfitrión, mediante un *plugin,* o en aplicaciones como teléfonos móviles que soportan el modelo de programación por *Applets*.

A diferencia de un *programa*, un *applet* no puede correr de manera independiente, ofrece información gráfica y a veces interactúa con el usuario, típicamente carece de sesión y tiene privilegios de seguridad restringidos. Un *applet* normalmente lleva a cabo una función muy específica que carece de uso independiente [54].

Ejemplos comunes de *Applets* son las Java *Applets* y las animaciones Flash. Otro ejemplo es el Windows Media Player utilizado para desplegar archivos de video incrustados en los navegadores como el Internet Explorer.

Un *applet* es un código JAVA que carece de un método *main*, por eso se utiliza principalmente para el trabajo de páginas web, ya que es un pequeño programa que es utilizado en una página HTML y representado por una pequeña pantalla gráfica dentro de ésta [54].

Por otra parte, la diferencia entre una aplicación JAVA y un *applet* radica en cómo se ejecutan. Para cargar una aplicación JAVA se utiliza el intérprete de JAVA (pcGRASP de Auburn University, Visual J++ de Microsoft, Forte de Sun de Visual Café). En cambio, un *applet* se puede cargar y ejecutar desde cualquier explorador que soporte JAVA (Netscape y Explorador de Windows).

4.6.2. Archivo .jar

Los archivos JAR (Java Archive) son la forma sencilla y eficiente de transportar recursos. Con un archivo JAR facilitamos la distribución e instalación de una gran variedad de

archivos, video, sonido, imágenes, texto, etc. Además un archivo JAR puede contener firmas digitales que aseguran la integridad y autenticidad de los datos.

Cuando se genera un ejecutable en Java se genera un archivo .jar. Para las clases (.class) la utilidad es doble: puede empaquetar sus clases y colocar el archivo JAR en el CLASSPATH. Los archivos .class son archivos generados por Java en función del código que ingresó el programador en los archivos .java. El intérprete de Java ejecuta la aplicación de acuerdo a la información que recibe en los archivos .class [55].

Con estas dos herramientas el tener un laboratorio virtual es muy sencillo. Se necesita que se cree la red y luego se puede crear un sitio web donde pueda ser empotrado el *applet* o tener disponible la aplicación comprimida en formato jar para descargarla.

4.6.3. Laboratorio virtual

Una de las grandes ventajas que posee EJS es la posibilidad de crear *Applets* o aplicaciones ejecutables de las simulaciones creadas, ver Figuras 43 y 44. Esta característica facilita la creación de laboratorios virtuales sobre una intranet.

Como ya se dijo anteriormente, los *Applets* pueden ser ejecutados por cualquier navegador web, aunque el explorador debe contar con algunos requisitos, actualmente éstos vienen por defecto con los navegadores.

Cada simulación tiene una página principal que sirve como introducción al contenido de la aplicación. Además posee tres enlaces en la sección de la izquierda.

a. Nuevo SCARA: Enlace a página principal de la simulación.
b. Simulación: Dirige al navegador hacia la página que contiene el Applet de la simulación.
c. Por último un enlace a la página principal de EJS.

Usando la herramienta Apache, el cual es un software libre que permite configurar a un computador como un servidor web, es posible crear un sitio al cual se puede conectar cualquier computador en Internet y realizar una simulación.

Para que la página esté disponible dentro del servidor web, es necesario publicarla dentro de éste. Para lograrlo se copian las páginas web dentro de la carpeta httdocs (para Apache). De esta manera escribiendo la dirección http://localhost/simulations/scarave.app/scarave.html en el navegador web del equipo servidor se abre la ventana de inicio.

Para conectarse desde otro computador distinto al servidor y suponiendo que en la intranet el servidor tiene un dirección IP 172.16.150.40, entonces es necesario escribir en la dirección del navegador web

http:// 172.16.150.40 /simulations/scarave.app/scarave.html

Figura 43. Página de introducción a la simulación.

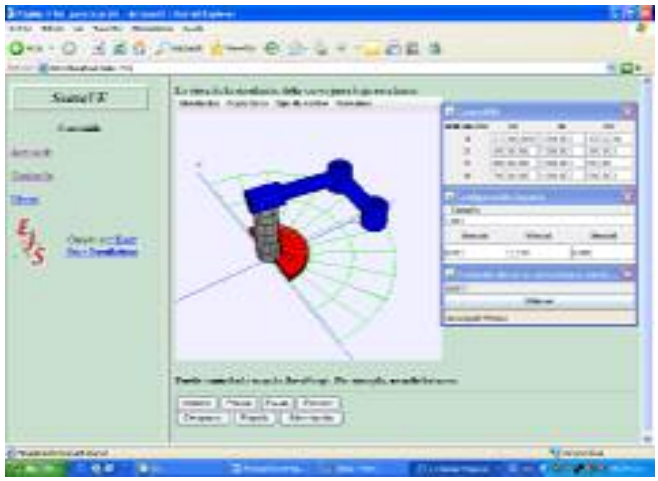


Figura 44. Página que contiene el *Applet* de la simulación.

Es posible también, crear un sitio web que contenga los enlaces a diferentes simulaciones, organizando un laboratorio virtual con todas las simulaciones creadas. Esto acercaría en gran medida a disponer de un laboratorio virtual sobre Internet, el cual permitiría que tanto estudiantes como profesores de cualquier parte del mundo tengan acceso a las nuevas tecnologías de diseño y simulación con unos requerimientos mínimos bastante bajos.

La cantidad de clientes que puede soportar el servidor al mismo momento depende en gran medida de las características del servidor y de la velocidad de transmisión de datos de la red. Sin embargo, la forma como trabajan los *Applets* permite optimizar el rendimiento del servidor, ya que son los clientes quienes ejecutan la aplicación con los propios recursos computacionales y no con los del servidor.

5. CONCLUSIONES

A pesar de las limitaciones que puede tener la realización de un modelo matemático de un objeto real, es posible simular el objeto con las características más importantes o influyentes en cuanto a su comportamiento mediante un modelo simplificado. Además con las nuevas tecnologías es posible tener un modelo virtual que ayude a entender el funcionamiento de un mecanismo.

La realidad virtual es una tecnología que se encuentra en pleno apogeo, la investigación y desarrollo llegan cada vez más lejos y existen proyectos como EJS que permiten que estas tecnologías estén más cerca de institutos educativos con recursos limitados.

Las ventajas que brinda la implementación de simulaciones en *Applets* y archivos *Jar* hace posible crear Laboratorios Virtuales en los cuales no se requieren servidores ni clientes de gran capacidad. Esta característica permite tener como prioridad la calidad de la representación virtual del fenómeno real y no la distribución de la simulación en la red.

EJS es una excelente herramienta para realizar simulaciones científicas en el campo de la ingeniería. El presente trabajo mostró que tiene un alto rendimiento con bajos recursos computacionales, gracias al uso de la plataforma Java.

Se pudo apreciar que existe una diferencia en la forma como los ordenadores y los programas de simulación realizan la representación de los números reales debido a los errores de truncamiento y redondeo. Estos influyen en los resultados de desempeño en pruebas comparativas con las dos herramientas EJS y Matlab. Este elemento es importante al momento de realizar una referencia.

6. PROYECTOS FUTUROS

El presente proyecto constituye el primer paso para el desarrollo de un Laboratorio Virtual de Robótica para la Universidad del Cauca, que permita a los estudiantes analizar las dinámicas de varios modelos de robots frente a las diferentes técnicas de control empleadas en robótica, sin la necesidad de estar físicamente cerca al robot e incluso en las instalaciones de la Universidad.

El siguiente paso a este proyecto es el estudio de las cinemáticas y dinámicas de diferentes robots para después desarrollar el programa en EJS, además de brindar la facilidad al estudiante de variar las características físicas de los mismos y sus trayectorias.

Una característica valiosa en un laboratorio virtual, es el hecho de poder simular ambientes típicos de funcionamiento así como también situaciones extremas para el funcionamiento de los dispositivos.

Un valor agregado para la herramienta está en el desarrollo de librerías tales como motores y sensores comerciales, logrando una simulación que tenga en cuenta con más realismo los límites físicos y eléctricos de los componentes del robot.

Aunque se identificó el error que hace que exista divergencia en los resultados de desempeño de las dos simulaciones, no se establece un método para solucionarlo. Puede ser posible reconfigurar la simulación hecha en Matlab para que tenga una mayor precisión o es posible realizar una implementación que pueda ser estándar, independiente de las herramientas de simulación y del ordenador.

Como se puede ver, aún existe mucho campo de investigación en la Línea de Robótica, más específicamente en el área de simulación con el objeto de desarrollar un Laboratorio Virtual de Robótica para facilitar la comprensión del alumno y brindar mayores herramientas pedagógicas para el profesor.

7. REFERENCIAS BIBLIOGRÁFICAS

[1] CHAVEZ F. BERNARD J. et al. "*Advancing the State of the Art in Flight Simulation via the use of Synthetic Environments*". Iowa State University. Iowa, Estados Unidos. 2001.

[2] COHEN Y., MAVROIDIS C. et al. "*Virtual reality robotic telesurgery simulations using MEMICA haptic system*". Proceedings of SPIE's 8th Annual International Symposium on Smart Structures and Materials. Newport, Estados Unidos. Marzo, 2001.

[3] DOMINGUEZ O. LOPEZ V. et al. "*Resultados Preliminares sobre Interacción Háptica en Laberintos Virtuales, con Propósitos de Diagnóstico en Pacientes con Discapacidades Neuropsicológicas"*. Universidad Autónoma del Estado de Hidalgo, Centro de Investigación en Tecnologías de Información y Sistemas. Pachuca, México. 2000.

[4] VAJTA L., JUHASZ T. "*Cutting Edge Robotics: The Role of 3D Simulation in the Advanced Robotic Design, Test and Control*". Vedran Kordic. Viena, Austria. 2005.

[5] MÜLLER D., ELEFTHERIOU P. et al. "*MARVEL - Mechatronics Training in Real and Virtual Environments: Concepts, Practices, and Recommendations*". Deiter Muller. Bremen, Alemania. 2005.

[6] SANZ W. *"Plataforma virtual Interactiva para la Modelación y Simulación de Robots bajo el ambiente de programación de Matlab"*. Universidad Central de Venezuela. Octubre de 2003.

[7] "*1er Congreso Nacional de Mecatrónica y tecnologías Inteligentes - CONAMTI*" (3 de abril de 2006). "Conferencias" [en línea]. Huichapan, México. Recuperado el 25 de Enero de 2007, de http://www.iteshu.edu.mx/CONAMIT/conferencias.asp.

[8] Tecnoedu (24 de Noviembre de 2006). "*Brazo Robot Antropomórfico simulado con Realidad Virtual RV-M1*" [en línea]. Córdoba, Argentina. Recuperado el 23 de enero de 2007, de http://www.tecnoedu.com/Denford/RobotVR.php.

[9] ZYDA M. "*From Visual Simulation to Virtual Reality to Games*". IEEE Computer Society, Vol 38, No 9, pp 25-32. 2005.

[10] ROY S. "*State of the art of virtual reality therapy (VRT) in phobic disorders*". PsychNology Journal. Vol 1, No 2, pp 176-183. Paris, Francia. 2003.

[11] GOMEZ P. "*Uso de Simuladores y otras ayudas Educativas en Medicina*" Universidad Nacional de Colombia, Revista Facultad de Medicina. Vol. 51, No. 4, pp 227-232. 2003.

[12] PEREZ B. "*Realidad Virtual en Medicina: Revisión de la Endoscopia Virtual*". 2005.

[13] CORREA C., GONZALES M. et al. "*Realidad Virtual Distribuida para soportar la Educación a Distancia en Colombia*". IV Congreso RIBIE (Red Iberoamericana de Informática Educativa). Brasilia, Brasil. 1998.)

[14] Universidad Militar de Nueva Granada (17 de Noviembre De 2006). "*CRV – Centro de Realidad Virtual*" [en línea]. Bogotá, Colombia. Recuperado el 12 de Diciembre de 2006, de http://www.umng.edu.co/www/section-128.jsp.

[15] Universidad del Cauca. "*Línea de Robótica*" [en línea]. Popayán, Colombia. Recuperado el 28 de Enero de 2007, de http://ai.unicauca.edu.co/robotica.html.

[16] MORALES N. Ensayo "*Una Astilla En la Mente*". Universidad de los Andes. Merida, Venezuela. 2002.

[17] LEVIS D. (27 de Noviembre de 2006) "*¿Que es la Realidad Virtual?"* [en línea]. Recuperado el 1 de Febrero de 2007, de http://www.diegolevis.com.ar/secciones/Articulos/Que_es_RV.pdf.

[18] FREUND E., ROSSMANN J. "*Proyetive virtual reality: Bringing the gap between virtual reality and robotic"*. IEEE Transactions on Robotics and Automation. Vol. 15, No 3, pp 411-422. Junio, 1999.

[19] http://oss.sgi.com/projects/inventor

[20] AUKSTAKALNIS S., BLATNER D. "*The Art and Science of Virtual Reality*". Silicon Mirage, Berkeley: Peachpit Press. Pp 238-242. Reino Unido. 1992.

[21] BRUNET P., VINACUA A. et al. (Mayo de 2006) "*Sistemas Gráficos Interactivos*" [en línea]. Universidad Politécnica de Cataluña. Barcelona, España. Recuperado el 1 de Febrero de 2007, de http://www.lsi.upc.edu/~pere/SGI/guions/ArquitecturaRV.pdf.

[22] A. Rowell. "*The design benefit of group VR*" Computer Graphics World. Vol 20. No 2. 1997.

[23] Activ@mente Empresa de Servicios Estratégicos (10 de Noviembre de 2005). "*VRML-Realidad Virtual*" [en línea]. México DF, México. Recuperado el 20 de Octubre de 2006, de http://www.activamente.com.mx/vrml/.

[24] FERTEY G., DELPY T. et al. "*La interfaz de los mundos Real y Virtual:An industrial application of virtual reality: and aid for designing maintenance operations in nuclear plants".* pp 151-162. Francia. 1995.

[25] NAGAO K., REKIMOTO J. "*The world through the computer: Computer augmented interaction with real world objects*". Sony Computer Science Laboratory Inc. Tokio, Japón. 1995.

[26] GALEANO J., GARCIA D. "*La Realidad Virtual*". Colegio Universitario de los Teques. Venezuela. 2000.

[27] ESPINOZA J. Tesis de Grado, "*Creación de ambientes virtuales automáticamente tomando fotografías con un robot móvi*l". Universidad Nacional de México. México. 2002.

[28] CANDELAS F. "*Propuesta de Portal de la Red de Laboratorios Virtuales y Remotos de CEA*". Documento de trabajo elaborado para la Red Temática DocenWeb: Red Temática de Docencia en Control mediante Web. Universidad de Alicante. España. 2003.

[29] Instituto Internacional de Física Teórica y Aplicada (IITAP). "*Informe de la reunión de expertos sobre laboratorios virtuales*". Ames, Estados Unidos. 1999.

[30] Noticiasdot.com (21 de Octubre de 2003)."*770000 Robots Industriales en el Mundo*" [en línea]. Barcelona, España. Recuperado el 25 de Enero de 2007, de http://www.noticiasdot.com/LaActualidad.

[31] DUEÑAS F. "*La Robótica*". Universidad la Salle. Cancún, México. 2006.

[32] Robot Institute of America (RIA) (23 de Agosto de 2005). "*Mathematics concepts for robot design*" [en línea]. Recuperado el 19 de Octubre de 2006, de http://www.roboticsonline.com/articles.

[33] MENESES A., LOPEZ A. Tesis de Grado, "*Control Difuso de un Manipulador de Cuatro Grados de Libertad*". Universidad del Cauca. Popayán, Colombia. 2007.

[34] KHALIL W., DOMBRE E. "*Modeling, Identification and Control of Robots*". Kogan Page Science. Londres. 2002.

[35] VIVAS A., MOSQUERA V. "Predictive functional control of a PUMA robot". Universidad del Cauca, Colombia. The International Congress for Global Science and Technology. Cairo, Egipto. 2005.

[36] SUAREZ R. "*Programación, Planificación y Control en Robótica*". Instituto de Organización y Control de Sistemas Industriales (UPC). Barcelona, España. 2006.

[37] ACOSTA L., SIGUT M. "*Matemáticas y Robótica*". SCTM05: Sociedad, Ciencia, Tecnología y Matematicas. Universidad de la Laguna.. México. 2005.

[38] CRAIG J. "*Introduction to robotics: mechanics & control*". Addison, Wesley. Indiana, USA. 1989.

[39] ÅSTRÖM K., WITTENMARK B. "*Computer – Controlled Systems*". Prentice Hall. New Jersey, Estados Unidos. 1997.

[40] VAJTA L., JUHASZ T. "*Cutting Edge Robotics: The Role of 3D Simulation in the Advanced Robotic Design, Test and Control*". Vedran Kordic. pp 47-60. Viena, Austria. 2005.

[41] WILLIS M. "*Proportional-Integral-Derivative control*". University of Newcastle MNPERE Company. Minneapolis, Estados Unidos. 2003.

[42] MENDES M., KRAUS Jr. W., et al "*Variable Structure Position Control of an Industrial Robotic Manipulator*". Journal of the Brazilian Society of Mechanical Sciences. Vol. 24, No 6, pp 169-176. Brasil. 2002.

[43] VIVAS A., POIGNET P. "*Control Predictivo de un Robot Paralelo*". Revista Iberoamericana de Automática e Informática Industrial. Vol. 3, No 4, pp 46-53. España. 2006.

[44] ASTROM K., HAGGLUND T. "*New tuning methods for PID controllers*", 3era Conferencia Europea de Control. España. 1995.

[45] ZHONG J. *"Tutorial: PID Controller Tuning: A Short Tutorial*". Mechanical Engineering, Purdue University. Estados Unidos. 2006.

[46] DORF R., BISHOP R. "*Modern Control Systems".* Addison-Wesley, Estados Unidos. 1998.

[47] Matlab. Massachussets, Estados Unidos. http://www.mathworks.com.

[48] RoboWorks. Austin, Estados Unidos. http://www.newtonium.com.

[49] Microsoft Robotics Studio. Estados Unidos. http://www.microsoft.com/spanish/msdn/articulos/archivo/050207/voices/learndefault.mspx

[50] 20Sim. Paises Bajos. http://www.20sim.com.

[51] Easy Java simulations. España. http://www.um.es/fem/Ejs/.

[52] www.opensourcephysics.org

[53] FERNANDEZ J. "*Métodos Matemáticos (I.T.I. Química y Física)*" [en línea]. Escuela Universitaria Politécnica. Sevilla. 2004. Recuperado el 22 de Agosto de 2007, de http://www.personal.us.es/julio/metodos/temas/cap1mm04-05.pdf

[54] Wikipedia. (28 de Septiembre de 2007) "*Definición de Applet*" [en línea]. Recuperado el 11 de Octubre de 2007, de http://es.wikipedia.org/wiki/Applet

[55] Gamarod, Codigos Java Scripts. (19 de Abril de 2003) "*Generar un archivo ejecutable con Java*" [en línea]. Recuperado el 11 de Octubre de 2007, de http://www.gamarod.com.ar/articulos/generar_un_archivo_ejecutable_con_java.asp

[56] Zeltzer, D. "*Autonomy,Interaction and Presence. Presence: Teleoperators and Virtual Environments*". 1992.