Herramienta para Programación de Controladores Para un Prototipo de Mano Robótica

Miguel Ángel Cabanillas Pérez
Mauro Fabián Muñoz Medina

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Programa en Ingeniería en Automática Industrial
Popayán, Cauca
2013

Herramienta para Programación de Controladores Para un Prototipo de Mano Robótica

Miguel Ángel Cabanillas Pérez
Mauro Fabián Muñoz Medina

Ing. Víctor Hugo Mosquera Leyton

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Programa en Ingeniería en Automática Industrial
Popayán, Cauca
2013

Tabla de Contenido

Lista de figuras

Lista de tablas

Dedicatoria

A Dios por ser luz y guía en todos los momentos de nuestras vidas.
A nuestros padres y familiares que con su apoyo incondicional nos
dan el impulso para alcanzar todos nuestros sueños y proyectos.

Miguel Ángel Cabanillas Pérez.
Mauro Fabián Muñoz Medina.

Agradecimientos

Los autores expresan sus agradecimientos:

A la Universidad del Cauca por su cariño y conocimiento que nos ha brindado.

A su director el Ingeniero Víctor Mosquera Leyton por su colaboración incondicional, orientaciones intelectuales y personales.

A los ingenieros Carlos Gaviria López y Cesar Augusto Quinayás Burgos que nos brindaron sus conocimientos y su tiempo.

A los miembros del Departamento de Electrónica, Instrumentación y Control por sus enseñanzas.

A los compañeros de Universidad por su alegría y compañía en estos años.

A los evaluadores de este proyecto que con su análisis y recomendaciones mejoran el aporte científico que se desea contribuir.

Capítulo 1

Introducción

En el grupo de I+D en automática industrial de la *Universidad del Cauca* se está trabajando en la construcción de un prototipo de prótesis de mano robótica de nueve grados de libertad. Hasta el momento el resultado es un prototipo de mano robótica de tres dedos y seis grados de libertad, documentado en [1]. En este trabajo fue posible realizar cinco agarres básicos: Pinza de precisión, agarre lateral, agarre de gancho y agarre cilíndrico. Entre los trabajos futuros propuestos en [1], se encuentra el poder realizar movimientos más naturales, haciendo que las articulaciones de los dedos sigan trayectorias a través de algoritmos de control.

1.1. Planteamiento del problema

El diseño del control de sistemas dinámicos, es un campo de ardua investigación, que ha aportado un sinnúmero de avances tecnológicos que han mejorado la calidad de vida de la sociedad. Sus aplicaciones van desde la industria, las comunicaciones, el transporte y por supuesto en la medicina.

Diseñar controladores que gobiernen dinámicas que interactúan con la sociedad, es una labor investigativa que hace algunos años se podía entender básicamente en dos etapas: la primera enfocada en la validación por simulación a través de software matemático como Matlab® y la segunda en la implementación de los algoritmos de control en los dispositivos o encapsulados finales de control (microcontroladores, DSP, FGPAS, entre otros). Actualmente los investigadores cuentan con herramientas, en las cuales es posible realizar validación por simulación y seguidamente validar los resultados en los prototipos físico-mecánicos que son objeto de control.

Ya que se cuenta con modelos virtuales de la prótesis de mano y de la facilidad de prueba de algoritmos de control en el software MatLab® sobre el prototipo virtual, resulta deseable que estos diseños, una vez verificados sobre el prototipo virtual, puedan embeberse y probarse sobre el prototipo real de forma rápida y sin necesidad de cambios significativos sobre la arquitectura hardware del sistema embebido. Esta posibilidad, facilitaría el que puedan mantenerse y mejorarse los diseños obtenidos para posibles nuevas arquitecturas de prótesis de mano a desarrollarse en el futuro, reduciendo el tiempo de diseño de forma significativa.

1.2. Objetivos de la tesis

1.2.1. Objetivo general

Diseñar una herramienta basada en MatLab® para evaluación y programación de controladores para un prototipo mecánico de mano robótica.

1.2.2. Objetivos específicos

1. Diseñar una simulación en 3D para evaluar el desempeño de varios tipos de controladores para un prototipo de mano robótica.

2. Construir un sistema hardware/software basado en dispositivos embebidos que permita el control del prototipo mecánico.

3. Construir las interfaces necesarias para que sea posible programar los dispositivos embebidos directamente desde el entorno de evaluación virtual diseñado.

4. Verificar que el desempeño de los algoritmos de control en el prototipo real, es similar al del modelo virtual mediante experimentos comparativos.

1.3. Organización de la tesis

El presente texto está organizado en seis capítulos, en los cuales se presenta al lector la investigación y el desarrollo hardware y software, para la manipulación y control de un prototipo mecánico de mano robótica. Una contextualización acerca del prototipado rápido con Matlab, instrumentación electrónica para prototipos de mano activa y características mecánicas propias del prototipo con que se desarrolla este trabajo son mostrados en el capítulo 2. El montaje de la instrumentación en el prototipo de

mano robótica: adecuación de señales de sensores, montaje de actuadores, variadores de potencia eléctrica y el estudio para la selección de una tarjeta de prototipado rápido son mostrados en el capítulo 3. En el capítulo 4, se presenta la explicación de la interfaz gráfica de usuario diseñada para el prototipado rápido de controladores. En el capítulo 5, se presentan resultados de simulación con el diseño virtual y de las pruebas realizadas con el prototipo mecánico, haciendo uso de dos técnicas de control y su sintonización. También en este capítulo se presenta el ejercicio experimental para la identificación de una articulación y la generación de consignas articulares. Finalmente se contrastan los resultados obtenidos mediante simulación y los resultados arrojados por el sistema físico y en el capítulo 6 se presenta las conclusiones y trabajos futuros.

Capítulo 2

Tópicos generales y estado de la técnica

En este capítulo se presenta en primera instancia la arquitectura y modelos matemáticos de un prototipo virtual de mano robótica. Seguidamente las características principales de la primera versión del prototipo mecánico. El prototipado rápido de control, desarrollos académicos y herramientas hardware y software disponibles en el mercado que relacionan Matlab®/Simulink® con microcontroladores mediante blockset son temas abordados en este capítulo. Finalmente se realiza el estudio de los transductores de posición, fuerza y corriente que pueden ser usados en el prototipo mecánico sin efectuar cambios significativos en el hardware existente.

2.1. Arquitectura y modelos de una prótesis de mano robótica

La arquitectura general de la prótesis desarrollada en [1] es mostrada en la *figura 2.1*. El número total de grados de libertad es de nueve, tres por cada dedo: índice, medio y pulgar. Los dedos anular y meñique replicarán los movimientos del dedo medio.

2.1.1. Modelo geométrico directo

En este caso de estudio, la estructura es de tipo arborescente. Por medio del *modelo geométrico directo* se describe la posición y orientación del órgano terminal (punta de los dedos) en una región de tres coordenadas *(x,y,z)*, a partir de las posiciones articulares de cada articulación (MCP,PIP y DIP de cada dedo). Los parámetros son

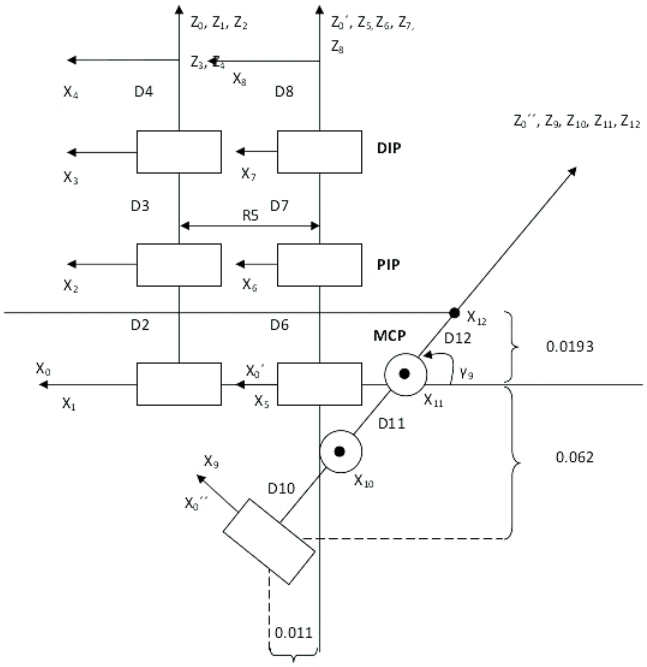


Figura 2.1: Arquitectura de la mano robótica.

mostrados en la *Tabla 2.1.*

j	σ_j	γ_j	b_j	α_j	d_j	θ_j	r_j
1	0	0	0	0	0	$\theta 1$	0
2	0	0	0	0	D2	$\theta 2$	0
3	0	0	0	0	D3	$\theta 3$	0
4	0	0	0	0	D4	0	0
5	0	0	0	0	0	$\theta 5$	R5
6	0	0	0	0	D6	$\theta 6$	0
7	0	0	0	0	D7	$\theta 7$	0
8	0	0	0	0	D8	0	0
9	0	$\gamma 9$	0	0	0	$\theta 9$	0
10	0	0	0	90°	D10	$\theta 10$	0
11	0	0	0	0	D11	$\theta 11$	0
12	0	0	0	0	D12	0	0

Tabla 2.1: Parámetros geométricos de la mano

Con:

D2=0.057; D3=0.039; D4=0.027; D6=0.052; D7=0.036; D8=0.025; D10=0.032;

D11=0.039; D12=0.044; R5=0.021; $\gamma 7 = 45°$; x1=0.062; x2=0.011.

En [1] se presenta cada dedo como una cadena cinemática independiente, de la siguiente manera:

$$ {}^{0}T_E = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & C12C3 - S12S3 & -C12S3 - S12C3 & -C12S3D4 - S12C3D4 - S12D3 - S1D2 \\ 0 & S12C3 + C12S3 & -S12S3 + C12C3 & -S12S3D4 + C12C3D4 + C12D3 + C1D2 \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

$$ {}^{0'}T_E = \begin{pmatrix} 1 & 0 & 0 & R5 \\ 0 & C45C6 - S45S6 & -C45S6 - S45C6 & -C45S6D7 - S45C6D7 - S45D6 - S4D5 \\ 0 & S45C6 + C45S6 & -S45S6 + C45C6 & -S45S6D7 + C45C6D7 + C45D6 + C4D5 \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

$$ {}^{0''}T_E = \frac{\sqrt{2}}{2} \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

$a_{11} = S9$
$a_{12} = (-C10 + C9S10)C11 + (S10 + C9C10)S11$
$a_{13} = (C9S10 - C10)S11 + (C9C10 + S10)C11$
$a_{14} = (-(C9S10 - C10)S11 + (C9C10 + S10)C11)D12 + (C9C10 + S10)D11 + C9D10 - x2$
$a_{21} = \frac{2}{\sqrt{2}}C9$
$a_{22} = -\frac{2}{\sqrt{2}}S9S1011$
$a_{23} = -\frac{2}{\sqrt{2}}S9C1011$
$a_{24} = -(S9C1011)D12 - S9C10D11 - S9D10$
$a_{31} = S9$
$a_{32} = (C10 + C9S10)C11 + (C9C10 - S10)S11$
$a_{33} = -(C9S10 + C10)S11 + (C9C10S10)C11$
$a_{34} = (-(C9S10 + C10)S11 + (C9C10 - S10)C11)D12 + (C9C10 - S10)D11 + C9D10 - x1$

2.1.2. Modelo dinámico

El *modelo dinámico* se define matemáticamente *(Ecuación 2.1)* como la relación entre las fuerzas aplicadas a los actuadores **(Γ)** y las posiciones, velocidades y aceleraciones articulares [2]. Se expresa así:

$$ \boldsymbol{\Gamma} = \boldsymbol{A}(q)\ddot{q} + \boldsymbol{C}(q,\dot{q})\dot{q} + \boldsymbol{Q}(q) + \boldsymbol{F_v}\dot{q} + \boldsymbol{F_s}sign(\dot{q}) \tag{2.1} $$

Donde $\boldsymbol{(\Gamma)}$ son los pares o torques aplicados a cada uno de los motores, $\boldsymbol{A}$ es la matriz de inercia del robot, $\boldsymbol{C}$ la matriz de Coriolis y fuerzas centrífugas, $\boldsymbol{Q}$ el vector de gravedad, $\boldsymbol{F_v}$ la matriz de frotamientos viscosos y $\boldsymbol{F_s}$ la matriz de frotamientos secos. Las posiciones, velocidades y aceleraciones articulares están dadas por q, $\dot{q}$, $\ddot{q}$ respectivamente. El modelo dinámico directo permite realizar la simulación del comportamiento de la mano robot, mientras que el modelo dinámico inverso permitirá la implementación de un controlador basado en el modelo. En [1], se determina el juego mínimo de parámetros que involucra el modelo con el fin de reducir cálculos. Se aplican técnicas de agrupamiento de parámetros y se hacen consideraciones especiales: suponer la matriz del tensor de inercia diagonal y los cuerpos de la mano con una forma simétrica. Finalmente en [1] se presenta la tabla de parámetros dinámicos de la base para cada uno de los dedos, con los cuales se reduce el *modelo dinámico* directo por dedo como sigue:

$$\boldsymbol{\Gamma} = \boldsymbol{A}(q)\ddot{q} + \boldsymbol{Q}(q) \tag{2.2}$$

2.2. Características mecánicas del prototipo de mano robótica

El diseño del prototipo de mano robótica se hizo con base en el modelo geométrico y dinámico *(Capítulo V)* [1], el prototipo se constituye por una palma y tres dedos (pulgar, índice y medio). Los dedos (índice y medio) se ubican encima de la palma y el dedo pulgar se encuentra a un ángulo de 45 grados. La mano presenta una estructura compacta donde el sistema de engranajes, actuadores y sensores están instalados en la mano.

Cada dedo se constituye por cuatro falanges (metacarpiana, proximal, media y distal) y tres articulaciones rotoides (metacarpofalángica (MCP), interfalángica proximal (PIP) e interfalángica distal (DIP)). El mecanismo de transmisión de movimientos se hace con motores DC ubicados en las articulaciones (MCP) y (PIP) y una transmisión de banda con restitución por resorte para la articulación (DIP).

Las junturas son articuladas por un sistema de engranajes, uno de los cuales es recto de 30 dientes módulo 0.5M y el otro se ubica a 180 grados, de 18 dientes módulo 0.5M con manzana y tornillo prisionero M2, se disponen de tal forma que los dientes de un engranaje coincidan perpendicularmente con el centro del otro, evitando que se muerdan los dientes de los engranajes.

Los dedos presentan 2GDL con movimiento independiente en las articulaciones MCP y PIP, la articulación DIP se mueve junto a la articulación PIP de tal forma que si la articulación PIP se mueve hasta un ángulo de 90 grados también lo hará la articulación DIP. A continuación se presentan los datos técnicos del prototipo de mano robótica, *Tabla 2.2.*

Tamaño	**Mano adulta**
Número de dedos	3
Número de GDL	6
Peso	0.130Kg
Número de sensores	6 + 3
Número de actuadores	6

Tabla 2.2: Datos técnicos del prototipo de prótesis de mano.

Toda la información presentada anteriormente corresponde al *(Capítulo IV)* [1].

Dedo	**Articulación**	**Ángulo mínimo**	**Ángulo máximo**
Índice	MCP	0^o	90^o
	PIP	0^o	90^o
Medio	MCP	0^o	90^o
	PIP	0^o	90^o
Pulgar	MCP	0^o	50^o
	PIP	0^o	90^o

Tabla 2.3: Rango de ángulos para articulaciones.

2.3. Prototipado rápido de control

La metodología tradicional de control depende fundamentalmente del modelo de la planta, pero a pesar de que existen diferentes métodos de aproximación de modelos asociados al proceso, en ocasiones los parámetros de dichos modelos son difícilmente medibles, lo que requiere de experimentación para obtener información que describa el modelo dinámico de la planta y lograr una buena estimación de dichos parámetros.

El diseñador debe realizar el análisis y la optimización de un sistema de control antes de implementarlo, haciendo experimentación que permita adquirir datos relacionados con las respuestas del sistema, bajo diferentes condiciones de operación. Los datos obtenidos son usados para mejorar el modelo primario de la planta y realizar un nuevo sistema de control basado en un modelo más preciso [3].

El análisis de controladores basados en modelos requiere, una vez que estos se diseñan, la implementación con la planta real, de tal forma que sea posible verificar su desempeño y permita su validación. El prototipado rápido de control tiene como objetivo la obtención confiable del diseño, logrando esclarecer errores o defectos en el controlador de forma previa [4].

El prototipado rápido de control se define como un mecanismo que permite probar de forma rápida y eficiente diferentes técnicas de control en sus etapas de desarrollo (diseño, implementación, pruebas y verificación), en un equipo con entradas y salidas conectadas a la planta física real, permitiendo interactuar y analizar resultados, para generar rápidamente soluciones en caliente hasta la consecución del controlador optimizado, sin la tediosa necesidad de escribir líneas de código de programación [5].

El proceso del prototipado rápido de control está conformado por las siguientes etapas [6]:

1. Modelado de la planta en un ambiente simulado.
2. Validación del modelo.
3. Diseño de una versión prototipo del sistema de control.
4. Pruebas de la versión prototipo del sistema de control en el modelo de simulación.
5. Evaluación de la versión prototipo del sistema del control en la planta real.

Un sistema típico para realizar prototipado rápido de control se conforma por los siguientes elementos [7]:

- Software de Diseño de Sistemas de Control Asistido por Computador (DSCAC), que permita el modelado matemático mediante funciones y/o bloques básicos.

- Un paquete de bloques simbólicos de entradas y salidas, que permitan importar la estrategia de control, probada en el modelo simulado de la planta.

- Un generador de código que transfiera los diagramas de bloques implementados en el DSCAC, a su correspondiente algoritmo en código C.

- Una interfaz de entradas y salidas para señales analógicas y/o digitales.

- Una aplicación GUI (Graphical User Interface) para el monitoreo del proceso que permita interactuar con el DSCAC para realizar el prototipado rápido de control.

2.4. Herramientas para el prototipado rápido

La esencia del prototipado rápido de control es el desarrollo y validación de técnicas de control en ambientes simulados, cuando se obtiene un diseño con buenos resultados de simulación, se transfiere a la planta real, es decir, se transcribe la técnica de control diseñada al lenguaje de programación de forma automática, evitando que el usuario se preocupe por realizar la programación [6]. A continuación se hará una breve descripción de algunas herramientas para el prototipado rápido que se han desarrollado:

2.4.1. Sistema de prototipado rápido de control para un motor DC con RTAI-LAB

En [5] se diseñó, implementó y validó una plataforma para realizar prototipado rápido de control a una planta basada en un motor DC. La plataforma consta de dos componentes uno software y otro hardware.

El componente software es el que permite la comunicación entre la tarjeta electrónica y el conjunto de herramientas de RTAI-Lab. El diseño software está encaminado al desarrollo de los componentes necesarios para la implementación de un sistema de control en lazo cerrado. Los bloques que se desarrollaron a nivel software son los siguientes:

- Bloque controlador: encargado de generar la señal de esfuerzo de control a partir de una consigna o Set-point.

- Bloque actuador: responsable de adecuar el esfuerzo de control a los valores correspondientes de la variable manipulada que se aplican a la planta.

- Bloque planta (Motor DC): dedicado a la comunicación de las señales de entrada y salida entre la tarjeta electrónica, los sensores, el driver y el motor DC.

- Bloque Transmisor: encargado a la medición y acondicionamiento de las variables de salida de la planta (velocidad, posición y corriente) y entregar una señal normalizada para las funciones de realimentación o despliegue.

- Bloque de Presentación: encargado de la interfaz gráfica de usuario para el monitoreo y supervisión de todas la señales de interés en el sistema de control.

- Bloque parámetros: en este bloque se configurarán todos los parámetros requeridos por los bloques principales para el funcionamiento del lazo de control.

El componente hardware implicó el diseño e implementación de una tarjeta electrónica que actúa como interfaz de comunicación entre el PC y el motor DC. Esta tarjeta se diseño definiendo los siguientes módulos:

- Módulo de alimentación: encargado de prevenir daños por cortocircuito o inversión en la alimentación, como también suministrar la energía necesaria a la tarjeta electrónica y al motor DC.

- Módulo de sensores: en este módulo se diseñaron los circuitos de adecuación de señales de los sensores de posición, corriente y velocidad respectivamente

- Módulo actuador-planta: este módulo lo constituyen el circuito integrado *L298N* (Actuador) y el motor DC (Planta).

- Módulo de comunicación: dedicado al procesamiento de las señales portadoras de información en la tarjeta electrónica, además de la generación de la señal de *PWM* que se le aplica al motor DC.

Cada módulo fue montado inicialmente en protoboard, con el fin de probar su

correcto funcionamiento e integrarlo con los demás para ir construyendo de forma progresiva el prototipo de la tarjeta electrónica.

Este sistema de prototipado rápido de control permite que los estudiantes puedan desarrollar diversas prácticas de control de velocidad y posición, mediante el uso de diferentes de tipos de controladores.

2.4.2. Sistema didáctico para la implementación de controladores digitales

En este proyecto [8] se presenta una herramienta didáctica orientada al apoyo en los procesos de enseñanza-aprendizaje del programa de Ingeniería en Automática Industrial de la Universidad del Cauca, este es un desarrollo con componentes de hardware como software que permiten interactuar y elaborar prácticas relacionadas con áreas como: control digital, instrumentación, modelado e identificación de procesos.

El diseño del sistema se basa en una estructura modular y flexible, de modo que el usuario puede intercambiar módulos para realizar diferentes tipos de prácticas, además se agregan características como portabilidad y bajo costo en relación a sistemas comerciales, por esta razón se diseñaron tres tarjetas tipo hardware:

- Tarjeta principal de control: es la encargada de establecer la comunicación con los diferentes módulos del sistema y permitir la comunicación con un computador desde el cual se puedan realizar diferentes acciones como: adquisición de datos, monitoreo de variables, análisis de datos mediante métodos gráficos. Además la tarjeta principal puede de ser programada mediante código generado en lenguaje C, esta tarjeta permite realizar prácticas de control por PC donde el controlador está en el computador y a través de la tarjeta se envía la señal de control a la planta objeto de control, también permite implementar esquemas de control embebido, en este método el controlador se lleva a cabo en la tarjeta principal y es donde se realizan los diferentes cálculos para generar una señal que permita controlar un sistema específico (planta).

- Planta de temperatura: sistema electrónico con componentes: un elemento calefactor, etapa de potencia (actuador) que permite controlar la carga eléctrica aplicada al dispositivo calefactor, sensor de temperatura para medir la variable a controlar, sistema para generación de disturbios en la planta. Esta planta permite al usuario apreciar

diferentes aspectos en el comportamiento de un sistema de temperatura como tiempos de estabilización, inercia del sistema y disipación de potencia entre otros.

- Planta de Circuitos RC: es también un sistema electrónico muy práctico basado en mallas de resistencias y condensadores, en el cual la variable a controlar es la tensión o voltaje a la salida del circuito aplicado a una carga resistiva, el sistema es altamente flexible ya que permite ser configurado como sistemas de 1er, 2do y 3er orden.

Con respecto al software existen dos partes importantes, el primero es el software que permite la programación de la tarjeta principal de modo que ésta pueda operar con las plantas y realizar acciones sobre las diferentes señales que se generan. Luego está un software desarrollado en Matlab® que permite el monitoreo del sistema desde un computador donde se realiza el análisis del comportamiento de los procesos de forma natural y controlada.

2.4.3. Simulación interactiva de motores de reluctancia autoconmutados

Esta tesis doctoral [9] presenta en el quinto capítulo una plataforma para el desarrollo de accionamientos de un Motor de Reluctancia Autoconmutado (SRM) con procesamiento en tiempo real, constituida según la metodología del prototipado rápido. Esta plataforma es una herramienta para la aplicación de la ingeniería concurrente al diseño de accionamientos SRM digitales y además de ser flexible, modular y robusta, permite al diseñador:

- Configurar convertidores con distintas topologías.

- Implementar distintas estrategias de control en tiempo real.

- Tener acceso a las magnitudes eléctricas más significativas.

- Evitar los problemas habituales de prueba y ajuste, que tanto tiempo hacen perder en el diseño convencional de accionamientos.

- Centrar la atención y los esfuerzos en los conceptos y aspectos clave de los accionamientos en desarrollo.

El Software utilizado en el modelado y en la simulación es Matlab®/Simulink®, haciendo uso del *toolbox*: Real Time Windows Target. Este *toolbox* presenta diferentes aplicaciones como control en tiempo real, simulación en tiempo real para plantas físicas analizadas usando un computador con sistema operativo Windows, permitiendo compartir los datos obtenidos con cualquier otra herramienta que coexista en este sistema operativo.

2.5. Controladores y sistemas de prototipado rápido

El control general del prototipo de mano robótica puede dividirse en dos fases: la primera se denomina *pre-agarre* e involucra el control de posición cuya finalidad es ubicar las articulaciones en los ángulos adecuados para realizar un agarre primitivo (precisión, cilíndrico, gancho y lateral). La segunda fase se denomina *agarre* e involucra el control de la fuerza con la cual los dedos del prototipo de mano robótica sujeta los objetos.

El controlador es el elemento en un *esquema de control*, que a partir de una medición de la variable controlada o variable de proceso, determina que hacer sobre la planta, a través de la modificación de una variable manipulada, que regula la entrada o salida de energía al sistema. Para el control de posición se definen dos variables, las cuales distinguen la posición angular y el voltaje aplicado al motor, dichas variables son la controlada y manipulada respectivamente. A continuación se presentan las tecnologías alternativas para prototipado rápido a través de Simulink®, con el fin de implementar controladores de posición angular.

2.5.1. Tecnología en Arduino

Arduino es una plataforma *OPEN SOURCE* de creación de prototipos electrónicos de código abierto [10], que permite la interacción entre el mundo físico (variable analógica o continua) y el mundo virtual (variable digital o discreta). Se fundamenta en la flexibilidad en recurso software y hardware. Las placas Arduino son construidas con microcontroladores Atmel, los cuales son programados en lenguaje C desde el software Arduino: Programming Language (basado en Wiring1) y el Arduino Development Environment (basado en Processing2). Principalmente la variación del microcontrolador Atmel sobre una placa, determina las diferentes presentaciones de placas Arduino,

destacándose las principales: Arduino Uno, Leonardo, Due, Mega 2560, Mega ADK, Robot, entre otras. En la *figura 2.2* se observa un potenciómetro conectado a una placa Arduino Uno, con el cual se puede variar el voltaje de alimentación a un motor DC.

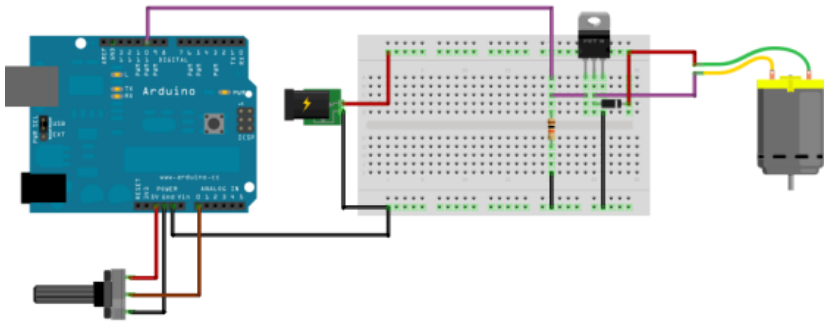


Figura 2.2: Arduino Uno como variador de voltaje a un motor DC [10].

Como plataforma, Arduino simplifica el proceso de trabajo con microcontroladores, en donde el usuario codifica las funciones y operaciones de manejo de recursos del microcontrolador y no sus configuraciones iniciales, que generalmente resultan ser confusas. Además del entorno de desarrollo *(IDE)* propio de Arduino, existen entornos gráficos de programación para ésta tecnología, lo que la hace muy atractiva. Estos pueden ser divididos en: entornos autónomos, entornos esclavos y de ayuda de desarrollo del prototipo. En el primero la interfaz gráfica permite crear firmware que se descarga directamente en el microcontrolador. En el segundo, la placa Arduino es vista como un sistema de entrada y salida de datos, pues un ordenador contiene la ley de control que gobierna el proceso. El tercero, permite hacer el diseño del circuito impreso ó *PCB*.

2.5.1.1. Arduino y MATLAB®/Simulink®

Arduino es multiplataforma, lo que permite hacer desarrollos de forma rápida y flexible, especialmente en software libre. MATLAB®/Simulink® ha agregado entre sus funciones *APM2 Simulink® Blockset* [11], que permite hacer comunicación entre la placa Arduino y la interfaz de Matlab®, algo que resulta ser muy útil, pues permite usar bloques funcionales de Simulink®, evitando tener que implementar estas funciones en C, reduciendo el tiempo de desarrollo de prototipos. Este *toolbox* pertenece al entorno autónomo y esclavo, porque permite compilar el código fuente, generando el lenguaje de máquina ó *firmware* y descargarlo directamente desde Simulink®, así como también utilizar el protocolo serial *RS232* para interactuar con el mundo físico por medio de estas placas.

2.5.2. Tecnología en Microchip

Entre los fabricantes de microcontroladores y microprocesadores se encuentra Microchip, caracterizado por innovar dispositivos para la industria como también para la academia. De esta casa, se distinguen tres clases de chips: microcontroladores *PIC®*, *dsPIC®*, y los controladores de señal digital *(DSC)*. Esta tecnología tiene su propio *IDE* en su última versión *MPLAB® X IDE* que al igual que el software para Arduino es multiplataforma.

2.5.2.1. Sim2lab

Sim2lab es un *blockset* para MATLAB®/Simulink® diseñado especificamente para un amplio rango de microcontroladores de la familia dsPIC33 de la casa Microchip. A través de estos bloques se accede a los recursos físicos del microcontrolador (adc, entradas y salidas digitales, entre otros). En la *figura 2.3* se observa el modelado del desplazamiento de un misil y el diseño del controlador usando Sim2lab.

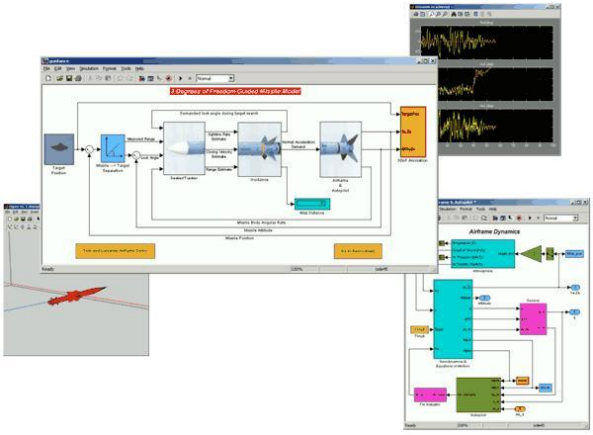


Figura 2.3: Diseño de control de trayectoria de un misil basado en modelo usando Sim2lab como herramienta de prototipado [12].

Adicionalmente este *toolbox* tiene la utilidad *SIM2LAB MyGUI Blockset* que facilita la creación de GUI (Interfaz Gráfica de Usuario) utilizadas, para tareas de monitoreo y registro de datos [12].

2.5.2.2. Embedded Target for dsPIC

Embedded Target es un *blockset* (*figura 2.4*) que al igual que Sim2lab, por medio de bloques de Simulink®, realiza las configuraciones iniciales y utiliza los recursos en un microcontrolador. Genera archivos con extensión .h y .c (lenguaje de alto nivel), para

que posteriormente en *MPLAB IDE* se genere el archivo con extensión .hex (lenguaje de máquina) que puede descargarse mediante la programadora en el dispositivo. Embedded Target tiene una versión de prueba, la cual limita el número de salidas *PWM* y entradas analógicas. La versión comercial permite el uso de todos los recursos físicos del microcontrolador.

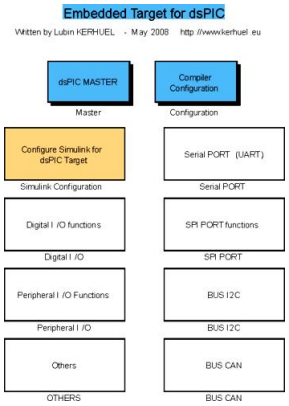


Figura 2.4: Bloques de simulink *blockset* Target for dsPIC [13].

2.5.3. Tecnología en STMicroelectronics

Con mayor capacidad de procesamiento matemático en comparación con un microcontrolador, se encuentran los procesadores digitales de señal *(DSP)*. *RapidSTM32* es un *blockset* para los *DSP* de 32 bits de la casa STMicroelectronics, que permite compilación y programación en caliente en sistemas embebidos. Al igual que en Arduino este *toolbox* pertenece al entorno autónomo y esclavo, porque permite compilar el lenguaje de alto nivel, generar el *firmware* y descargarlo directamente desde Simulink®, así como también utilizar una conexión *USB* para hacer tareas de monitoreo y registro. Se destaca la tarjeta STM32VLDISCOVERY y la familia FiO, comprendida por las boards FiO Lite y FiO Std. Su nombre proviene de: **F**our **i**n **O**ne en alusión a sus cuatro funcionalidades:

1. Adquisición de datos.

2. Generación de señales.

3. Registro de datos.

4. Aplicaciones independientes personalizas.

Actualmente *RapidSTM32* es libre y se distribuye bajo licencia BSD (Berkeley Software Distribution), sin embargo no es *OPEN SOURCE*. La utilidad ofrece dos conjuntos de características:

1. Limitada: permite generar el código fuente para STM32 de 32 bits ARM Cortex-M3, para máximo 2 conversores analógicos digitales, 2 salidas de *PWM* y sin restricciones en salidas o entradas digitales. Generación de archivo .axf (lenguaje de máquina) y programación de board desde el sofware Keil μVision IDE.

2. Completa: habilitada mediante llave hardware por una de las tarjetas de la familia FiO (Lite ó Std) al conectarse a un host del PC. Permite generar el código fuente y *firmware* para los DSPs STM32F103R8 (Lite) y STM32F103RET6 (Std), a demás la programación en caliente directa desde Simulink®. En la siguiente *figura 2.5* se observa la interconexión de dos tarjetas FiO Std, como unidades de envío y recepción para la medición de nivel de agua.

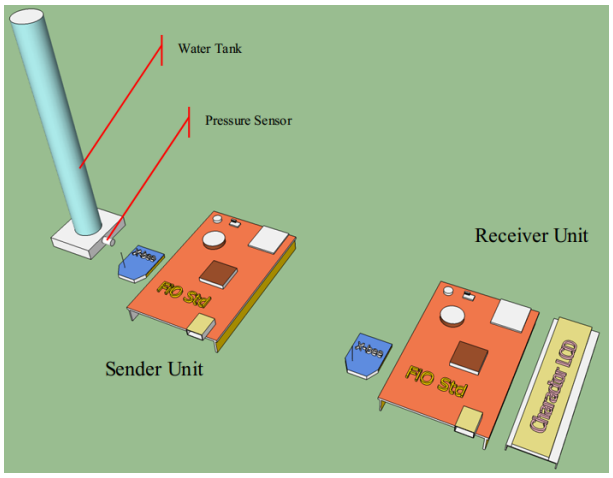


Figura 2.5: Medición de nivel de agua con tarjeta FiO Std [14].

2.6. Sensores en prototipos de mano robótica

La actividad motora y sensorial de la mano humana es muy compleja, sin embargo cuando se hacen prototipos de mano activa, desde la investigación de control de proceso, se puede asimilar como un sistema dinámico que hace funciones de *pre-agarre* y *agarre*. Así como la mano humana está dotada de sensores biológicos, el prototipo es dotado con sensores artificiales. Estos sensores pueden ser clasificados según su comportamiento físico (piezoeléctrico, piezorresistivo) y según la variable medida (tacto, posición, velocidad, fuerza, entre otros).

2.6.1. Sensores de posición angular

Actualmente existe gran variedad de sensores de posición angular en el mercado. Sin embargo todos estos sensores no pueden ser integrados al prototipo de mano robótica. A continuación se presentan sensores alternativos.

2.6.1.1. Sensor fotoeléctrico

Un sensor *fotoeléctrico* (*figura 2.6*), se basa en la percepción (fotorresistencia) de un haz de luz emitido por una fuente luminosa constante (diodo led). Una superficie cilíndrica (adaptada a las articulaciones), es atravesada por este haz de luz. El movimiento de la articulación, produce un efecto de variaciones de longitud, diámetro e intensidad de luz sobre el receptor, el cual los transforma en una variación de voltaje.

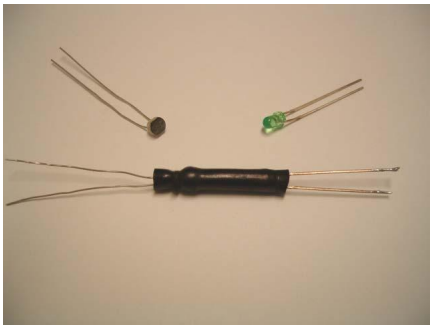


Figura 2.6: Dispositivo fotoeléctrico elástico [15].

Con el fin de evitar que haya fuga de luz hacia el medio externo, así como del medio al dispositivo, la superficie externa es recubierta con pintura de color negro. En la *figura 2.7*, se aprecia como este sensor es usado en un guante, distinguiéndose claramente ángulos de 30°, 55° y 90°, extensión, semiflexión y flexión total respectivamente.

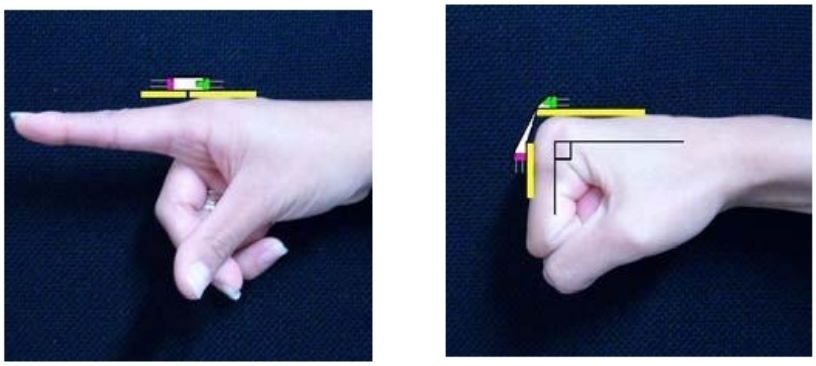


Figura 2.7: Colocación del dispositivo sobre una articulación MCP [15].

2.6.1.2. Sensor fotoóptico

Un sensor *fotoóptico* (*figura 2.8*), funciona de forma similar al anteriormente abordado. Su diferencia radica en el dispositivo emisor, receptor y el medio por el cual viaja el haz de luz emitido. El emisor es un diodo de luz infrarroja, enviada a una determinada frecuencia o señal de reloj. El medio de propagación suele ser fibra óptica o guías de onda planares. Los receptores utilizados son fotodiodos ó fototransistores.

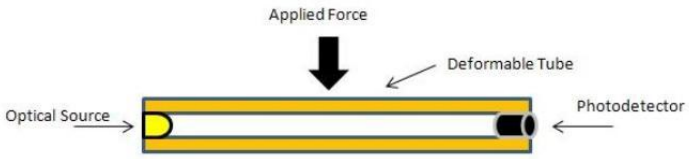
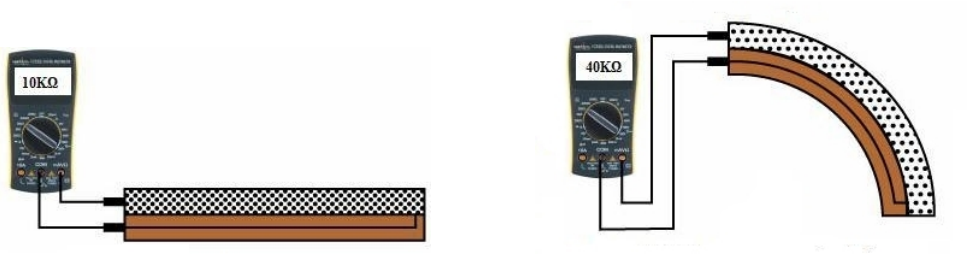


Figura 2.8: Dispositivo fotoóptico [16].

2.6.1.3. Sensor de deflexión

Un sensor de *deflexión* es un tipo de sensor piezoeléctrico. Puede definirse como un resistor variable que cambia su resistencia cuando es doblado. Por ser un transductor pasivo, necesita exitación de una fuente externa. Son elementos resistivos de carbono dentro de un sustrato flexible y delgado. Cuando está en la posición nominal (0°) las partículas de la película de carbono se encuentran más unidas, por lo que la resistencia del dispositivo es menor. Cuando la deflexión es máxima (90°), estas partículas se encuentran más separadas, por lo que la resistencia del dispositivo es mayor, *figura2.9*. La variación de su resistencia es en un solo sentido.



(a) Partículas conductoras juntas. (b) Partículas conductoras más separadas.

Figura 2.9: Comportamiento interno de un sensor de deflexión [17].

En el mercado se encuentran dos presentaciones: de 2,2" y 4,5", con igualdad de características. Tienen un tiempo de vida de más de un millón de usos y su rango de temperatura de trabajo es de $-35°C$ a $80°C$ [18].

2.6.1.4. Sensor de efecto Hall

Un sensor de *efecto Hall*, se basa en el principio de *efecto Hall*, descubierto en 1879 por el físico estadounidense Edwin Herbert Hall. Este principio establece que cuando un conductor es atravesado por un campo magnético, inmediatamente aparece un campo eléctrico en el conductor, que puede ser medido como una diferencia de potencial entre dos puntos del conductor [19]. El prototipo de mano robótica con el cual se desarrolló este proyecto fue inicialmente instrumentado con sensores de *efecto Hall*. En [1] se hace un estudio del sensor *HMC1501* que tiene un rango de medición de ±45°. En la *figura 2.10* se observa el montaje del sensor en las articulaciones del dedo antropomorfo.

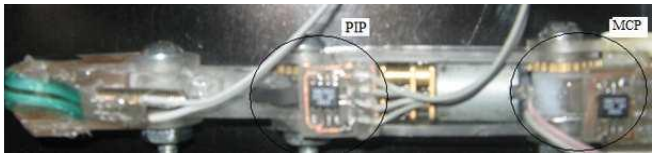


Figura 2.10: Sensor HMC1501 en articulaciones MCP y PIP [1].

En la *figura 2.11*, se muestra la caracterización de un sensor *HMC1501* sobre una articulación. Como puede observarse, la relación es bastante lineal. Sin embargo, sería deseable que el sensor representara todas sus medidas sobre todo el rango del conversor analógico digital.

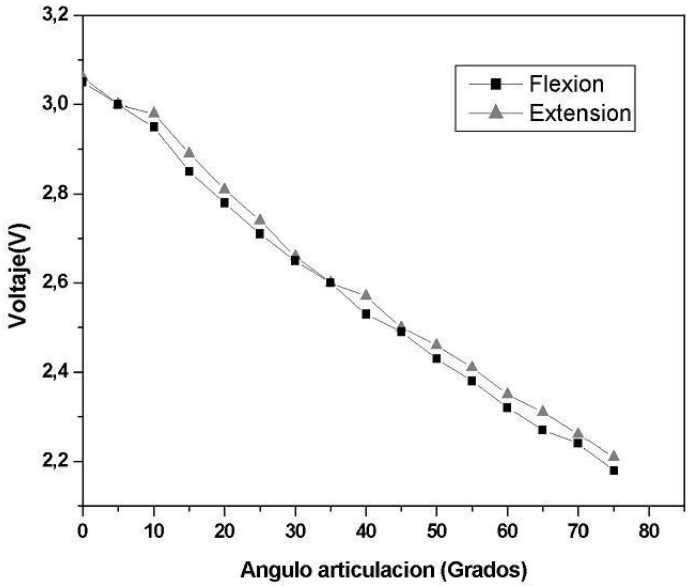


Figura 2.11: Curva de calibración sensor HMC1501 [1].

2.6.2. Sensores de fuerza

Estos sensores permiten transformar una fuerza aplicada sobre una superficie conocida, en una magnitud eléctrica proporcional: voltaje, corriente o resistencia. Seguidamente del control de posición, está el control de fuerza, cuyo objetivo es asegurar los objetos de modo que estos no se caigan o se deslicen evitando también un daño en el actuador (motor o puente H). A continuación se presentan los sensores de fuerza alternativos.

2.6.2.1. Sensor capacitivo

Estos sensores poseen una membrana de material dieléctrico, que es cubierta por dos superficies. Sobre una de éstas se aplica una fuerza, que cambia las propiedades de dicha membrana, produciéndose una variación en la cantidad de carga eléctrica almacenada (ver *figura 2.12*). La *(Ec. 2.3)*, relaciona la capacitancia *(C)* con la deformación de la membrana *(Δd)*, donde A es el área de la superfice y ε es la constante dieléctrica del material.

$$C = \varepsilon \cdot \left(\frac{A}{d + \Delta d}\right) \tag{2.3}$$

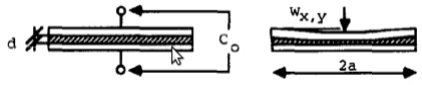


Figura 2.12: Deformación de un sensor de fuerza capacitivo

2.6.2.2. Sensor piezoeléctrico

Estos sensores son de uso común para la medición de fuerza. Generan un voltaje proporcional en amplitud a la presión aplicada. Están hechos de materiales como el cuarzo o cerámicas tratados con materiales con propiedades piezoeléctricas. Internamente se constituyen de moléculas bipolares ordenadas que actúan como pequeños dipolos. Al aplicar cierta fuerza, producen una pequeña oscilación hasta que se estabilizan, es esta oscilación de cargas lo que genera un voltaje en forma de una sinusoidal amortiguada, que posteriormente es procesada por circuitos adicionales (ver *figura 2.13*).

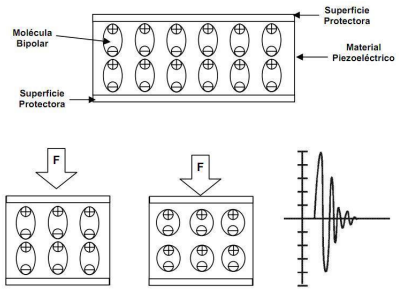


Figura 2.13: Deformación de un sensor de fuerza piezoeléctrico [20].

2.6.2.3. Sensor piezorresistivo

Estos sensores son basados en el principio de la piezorresistividad, el cual consiste en la variación de la resistencia de un material al aplicar una presión en sus superficies. El material está constituido por partículas conductoras y no conductoras, cuando se ejerce una fuerza, éstas partículas se reorganizan, disminuyendo directamente la resistencia del material. Entre sus ventajas esta su durabilidad, debido que no sufre mayor deformación para obtener variaciones de resistencia, lo que implica mayor tiempo de vida. Son de fácil adquisición en el mercado y de fácil adaptación y montaje. El prototipo de mano robótica con el cual se desarrolló este proyecto fue instrumentado con sensores de fuerza *piezorresistivos*.

2.6.3. Sensores de corriente

Entre las variables de medición, para realizar control de fuerza se encuentra la corriente circundante por el actuador. También es una variable de entrada para leyes de control de posición que observan el motor como el conjunto de dos sistemas interactuantes: uno eléctrico más uno físico.

2.6.3.1. Resistencia Shunt

Es una resistencia (ver *figura 2.14*) que se conecta en serie a la carga por lo que en ella cae una diferencia de potencial en sus terminales y resulta proporcional a la corriente que la circula. Se caracteriza por tener un coeficiente muy bajo de temperatura. Son sensores económicos, sin embargo sus medidas no son correctas cuando la frecuencia de señal medida es muy alta, además suelen presentarse pérdidas por calor cuando sus valores son muy elevados.

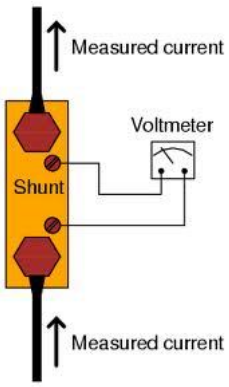


Figura 2.14: Medición de corriente con resistencia Shunt.

2.6.3.2. Sensor de efecto Hall

A diferencia de los sensores de *efecto Hall* para medición de posición en donde hay una conexión magnética, en estos sensores hay una conexión eléctrica en serie con la carga (*figura 2.15*). De fácil acondicionamiento de señal y muy robustos ante el ruido. Entre sus desventajas está la alta variación con respecto a la temperatura de operación. Se encuentra en el mercado en los rangos de medición desde ±5 hasta ±30 amperios.

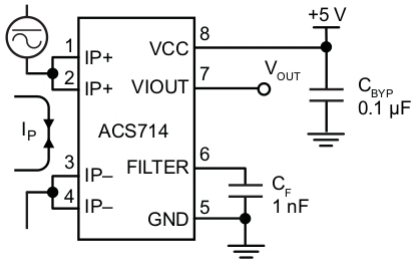
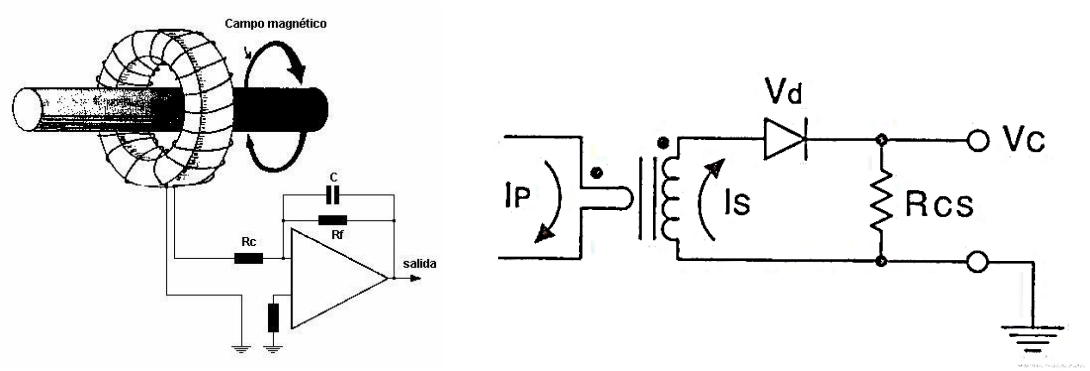


Figura 2.15: Conexión básica de un sensor de efecto Hall: *ACS714ELCTR-05B-T*

2.6.3.3. Sensores inductivos

En este grupo se destacan dos sensores principalmente: transformador de corriente y la *bobina de Rogowsky* (*figura 2.16*). En el primero, se circula una corriente por uno de los devanados del transformador y en el segundo devanado inmediatamente aparece, por medio de inducción electromagnética, un voltaje proporcional a la corriente que circula por la carga. En la *bobina de Rogowsky*, se puede medir la derivada de la corriente, por lo que resulta necesario implementar un integrador a fin de obtener la corriente que circula por la carga [19].



(a) Bobina de Rogowsky. (b) Transformador de corriente.

Figura 2.16: Sensores de corriente inductivos [17].

2.7. Actuadores en prototipos de mano robótica

Los actuadores de mayor uso en el desarrollo de manos robóticas son: micromotores DC con escobillas y cajas reductoras a fin de aumentar el par de carga y reducir la velocidad angular. Actualmente en el mercado es posible encontrarlos en tamaños reducidos, sin embargo su principal desventaja es el mantenimiento que debe realizarse sobre el sistema de escobillas. Estos motores pueden observase en la mano SKKU Hand II, como puede observarse en la *figura 2.17*.

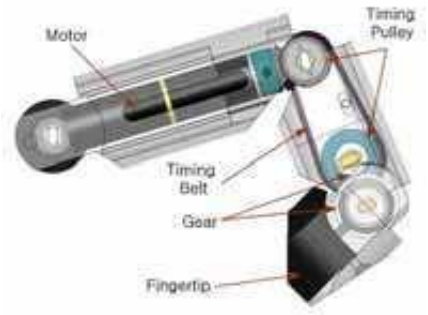


Figura 2.17: Motor DC como actuador en articulación de mano robótica: SKKU Hand II [21].

Los motores *brushless* son otro tipo de actuadores, en los cuales se ha cambiado su sistema de conmutación mecánica a través de escobillas, por una conmutación electrónica, usando el principio de estado sólido. Su costo es más elevado que un micromotor DC, pero las tareas de mantenimiento son menores, que se traduce en más tiempo de vida útil.

Capítulo 3

Instrumentación del prototipo de mano robótica

En este capítulo, se presenta en detalle la instrumentación del prototipo: tarjeta de control y prototipado, sensores para control de posición y de fuerza, adicionalmente sensores de corriente (variable necesaria para la técnica de control por *Modos Deslizantes o Sliding*, que puede ser abordada en posteriores trabajos de investigación). También se muestra el montaje y el acondicionamiento de la señal de estos sensores. Finalmente es presentado el elemento final de control, conformado por motores con caja reductora y puente H, que permiten variar la entrada de voltaje a los motores de las articulaciones haciendo uso de la técnica modulación por ancho de pulso *(PWM)*.

3.1. Tarjeta de control y prototipado rápido

Los requerimientos de instrumentación para el prototipo de mano robótica vienen determinados por el número de variables que deben ser digitalizadas y las salidas de mando de esfuerzo que salen del controlador hacia cada una de las articulaciones. La *Tabla 3.1* relaciona estas especificaciones hardware. Para la selección de la tecnología se elaboró una lista que cumpliera los requerimientos de hardware y que además estuviera asociado a Matlab®/Simulink® mediante un *blockset* de prototipado rápido. La *Tabla 3.2* muestra las alternativas y su disposición de compilación y la programación desde el ambiente de Simulink®.

Módulo	**Cantidad mínima requerida**
ADC	6
PWM	6
I/O Digitales	12

Tabla 3.1: Requerimientos de hardware.

Como resultado del análisis de los requerimientos y alternativas anteriormente mencionadas, se procedió a adquirir una tarjeta *Fio Std* de *STMicroelectronics*. La decisión se apoya fundamentalmente en sus características prestacionales, especialmente si se piensa en posteriores trabajos en la implementación de controladores de fuerza. La familia de tarjetas *FiO* se caracterizan por ser una llave hardware que habilita todas las funcionalidades del *blockset RapidSTM32*.

Alternativa	**Compilación**	**Programación/ restricción**	**Características**
Arduino MEGA 2560	Directa desde Simulink®	Directa desde Simulink®, ninguna	256KB Flash, 4KB EEPROM, 8K SRAM, 16MIPS at 16MHz
DSP Microchip (dsPIC33FJ12GP201)	Directa desde Simulink®	MPLAB IDE + Programadora, 2 adc, 2 pwm	12KB Flash, 1.024KB SRAM, 40 MIPS
STM32 VLDISCOVERY	Directa desde Simulink®	MDK ARM, 2 adc, 2 pwm, total I/O digitales	128KB Flash, 8K SRAM, 1.25 DMIPS/MHz
Fio Lite	Directa desde Simulink®	Directa desde Simulink®, ninguna	128KB Flash, 20KB SRAM, 90 MIPS.
Fio Std	Directa desde Simulink®	Directa desde Simulink®, ninguna	512KB Flash, 64 KB SRAM, 90MIPS.

Tabla 3.2: Análisis de tecnologías de prototipado rápido.

En la *figura 3.1* se muestra la tarjeta de control y de prototipado rápido FiO Std. Las dimensiones de la misma se observan en la *figura 3.2*.

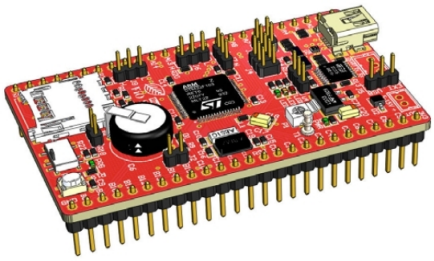


Figura 3.1: Tarjeta FiO Std [22].

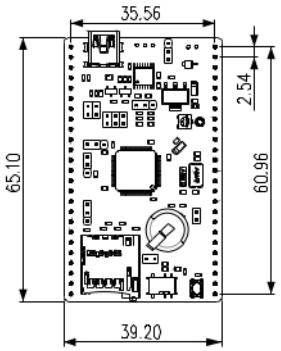


Figura 3.2: Dimensiones tarjeta FiO Std [22].

3.2. Elementos primarios de control

3.2.1. Sensor de posición

Una vez estudiados algunos sensores de posición que podrían adaptarse en las articulaciones del prototipo de mano robótica, se procedió a evaluar criterios técnicos con los cuales se pudo tomar una decisión.

Alternativa	**Etiqueta**
Sensor foto-eléctrico	S1
Sensor foto-óptico	S2
Sensor de deflexión	S3
Sensor de efecto Hall	S4

Tabla 3.3: Alternativas para sensar posición angular.

Los criterios a tener en cuenta se relacionan con una condición o requerimiento, que

son específicos para el prototipo mecánico, entre ellos el rango de medición y tamaño. La prioridad más alta se asocia a que el transductor con el que se piensa instrumentar responda de forma sensible a variaciones de posición angular. Se busca también que el circuito de acondicionamiento de señal pueda ser fácilmente reducido y reproducido, teniendo en cuenta que es el mismo para las seis articulaciones.

Criterio	**Descripción**	**Requerimiento**	**Prioridad**
Rango	Rango de medición del sensor.	Medir ángulos entre 0° a 90°.	5
Montaje	Adaptación sensor sobre articulación	No hacer modificaciones de consideración sobre el prototipo.	4
Tamaño	Dimensión del sensor.	Consecuente a las dimensiones del prototipo.	3
Vida útil	Tiempo conservación de características físicas.	Ejecución de 500 agarres.	2
Costo	Valor de adquisición.	No mayor de 50 dólares por sensor.	1

Tabla 3.4: Criterios y prioridades para elección.

		Alternativas			
Criterio	**Prioridad**	**S1**	**S2**	**S3**	**S4**
Rango	5	**6** 30	**8** 40	**10** 50	**7** 35
Montaje	4	**7** 28	**7** 28	**8** 32	**10** 40
Tamaño	3	**7** 21	**7** 21	**7** 21	**8** 24
Vida útil	2	**7** 14	**7** 14	**10** 20	**10** 20
Costo	1	**10** 10	**10** 10	**8** 8	**8** 8
Total		103	113	**131**	127

Tabla 3.5: Matriz de selección.

Para seleccionar el sensor óptimo para el proyecto se evaluaron las alternativas expuestas mediante la matriz de selección (ver *Tabla 3.5*). A partir de ésta, se apoyó la decisión de seleccionar el sensor de *deflexión*. Cabe aclarar que este método es muy subjetivo y que no puede ser único factor de decisión. Sin embargo existen dos razones por las cuales es preferible cambiar los sensores de posición de *efecto Hall*, con los que primeramente fue instrumentado el prototipo. Siendo el mismo sensor *(HMC1501)*, su comportamiento varía en las diferentes articulaciones. Además como pudo observarse en la *figura 2.11*, hay subutilización del conversor analógico digital. El sensor de *deflexión* es usado en tecnología de asistencia y recuperación física, como punto de aporte de este proyecto se pretende usarlo en un prototipo prueba de mano activa.

3.2.1.1. Montaje y caracterización de los sensores de deflexión

Se ubicaron sensores de deflexión o flexo-sensores, en cada una de las articulaciones. En la *figura 3.3* se muestra la ubicación de los sensores de posición en las articulaciones proximal y medial. El flex sensor es conectado en un puente Wheatstone, que detecta la pequeña variación de resistencia y la transforma en una variación de voltaje, posteriormente amplificada por un amplificador operacional, en configuración diferencial. Por medio de un potenciómetro *RSX* se calibra en cero voltios el puente Wheatstone y con los potenciómetros *RFSX y RFSXA* se calibran las ganancias del operacional para garantizar que la variable de proceso recorre todo el rango del conversor analógico digital.

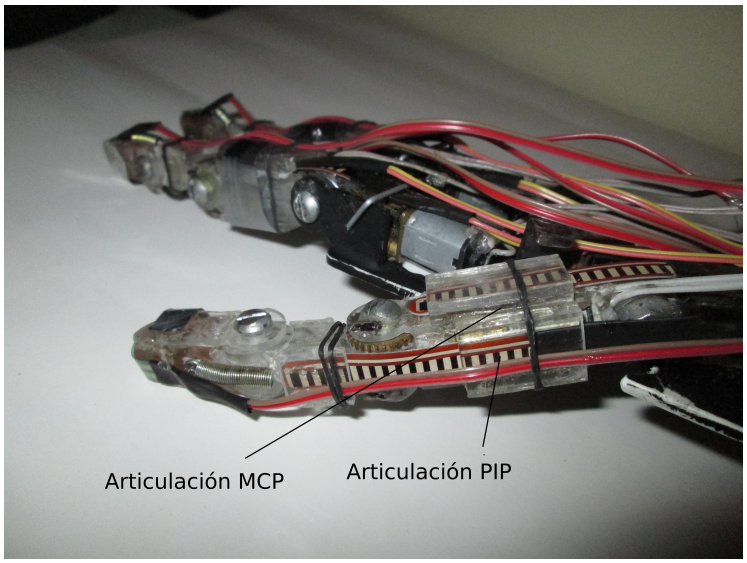


Figura 3.3: Flex sensor en articulaciones MCP y PIP del dedo pulgar.

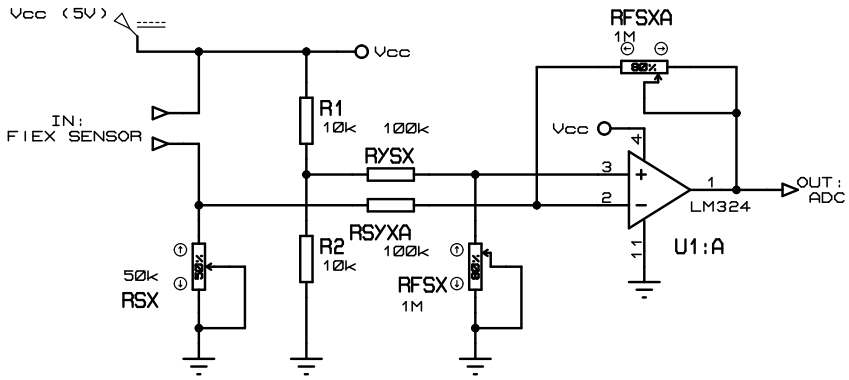


Figura 3.4: Circuito de acondicionamiento para el sensor de posición flex sensor.

La resistencia nominal de fabricación del flexo-sensor varía entre $20k\Omega$ y $35k\Omega$, por lo tanto se escoge $RSX = 50k\Omega$. De esta manera es posible equilibrar el puente *Wheatstone* aun cuando el sensor ha sufrido cambios en sus características debido al uso y la presión sometida en el montaje. Los resistores R1 y R2 se toman de igual valor para hacer el punto de equilibrio en el puente. La configuración del amplificador operacional (U1:A) es en modo diferencial, lo que permite amplificar la diferencia entre dos tensiones v_1 y v_2. La tensión de salida se determina mediante la *(Ec. 3.1)*

$$v_{out} = \frac{RFSX}{RYSX} \cdot (v_2 - v_1) \tag{3.1}$$

Como punto de partida del diseño del circuito de acondicionamiento se fija $RYSX = 100k\Omega$. Para calibrar el sensor de una articulación se procede a ubicar la posición angular en 0^o y posteriormente mediante el potenciómetro *RSX* se equilibra el puente *Wheatstone*. Los potenciómetros *RFSX* se calibran mediante ensayo y error, hasta obtener una tensión de salida $v_{out} \approx 0{,}4$ voltios, (recomendación que se hace para que la tensión aplicada al conversor analógico digital no sea menor o igual a cero). Esto genera una ecuación característica en el sensor que no pasa por el origen. Seguidamente se pone la articulación a 90^o, se ajustan nuevamente los potenciómetros *RFSX* hasta lograr una tensión de salida $v_{out} \approx 3$ voltios, con el fin de no acercarse a la tensión máxima de entrada del ADC (3.3 voltios). Un voltaje superior puede dañar el canal ADC. Cuando se regrese la articulación a 0^o se observará que la tensión de diseño $v_{out} \approx 0{,}4$ voltios ha desaparecido, posiblemente haya una tensión mayor, requiriendo un nuevo ajuste. El proceso de calibración consiste en ejecutar la secuencia anterior variando el estado de los potenciómetros *RFSX* hasta lograr el rango de tensión deseado para la variación de posición angular de la articulación asociada.

El ejercicio práctico de calibración de sensores de posición consiste en variar la consigna de voltaje (en el esquema de control en la *figura 3.5*) , a fin de generar variaciones graduales de posición angular. Con el control PI se recorre la articulación mecánica desde un ángulo cercano a cero, hasta su ángulo máximo. Se realiza a paso promedio de 0.3 voltios. En cada paso mediante una técnica de visión de máquina, se obtiene la posición angular para una articulación del prototipo mecánico. Mediante un *Host* en simulink se registra el voltaje en el ADC, para posteriormente realizar la regresión entre las dos variables.

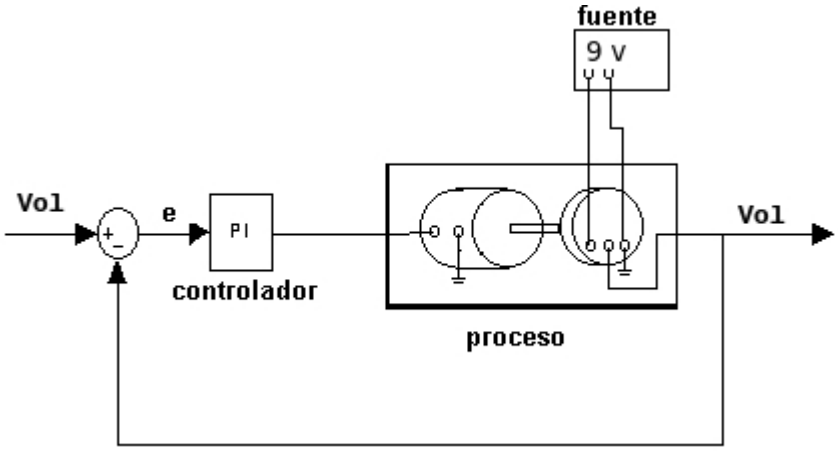


Figura 3.5: Control PI para voltaje, para ejecución de movimientos angulares graduales.

En la *figura 3.6* se muestra el montaje realizado para la adquisición de variación angular mediante una cámara. Se observa en la *figura 3.7* interfaz hecha en *LabVIEW*, para el procesamiento de imágenes.

Figura 3.6: Montaje para la calibración de sensores.

La calibración de los sensores de deflexión se realiza con el ejercicio anteriormente explicado para cada una de las articulaciones restantes. Este experimento fue apoyado por el Ingeniero Cesar Augusto Quinayás (estudiante de doctorado en Ciencias de la Electrónica).

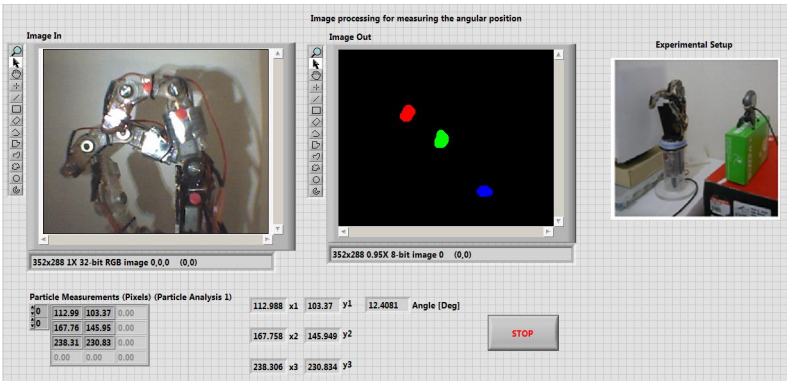


Figura 3.7: Software para medición de ángulos con visión de máquina.

La curva que relaciona la *posición angular ($\theta°$) v.s voltaje (V)* se muestra en la *figura 3.8*.

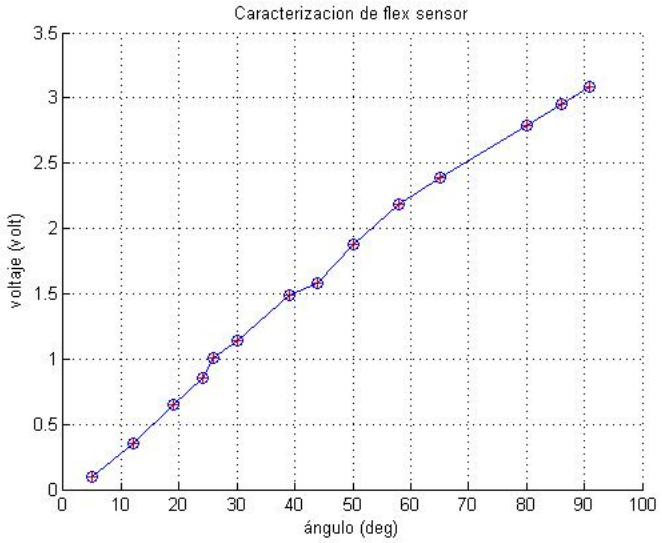


Figura 3.8: Caracterización sensor de deflexión.

3.2.2. Sensor de fuerza

Se decidió mantener los sensores con los que fue instrumentado inicialmente el prototipo. Los FSR presentan ventajas como fácil adaptación, larga vida y su circuito

de acondicionamiento es de fácil implementación. Para agregar independencia entre las articulaciones se agregaron tres sensores de fuerza para las articulaciones *MCP* de los dedos índice, medio y pulgar.

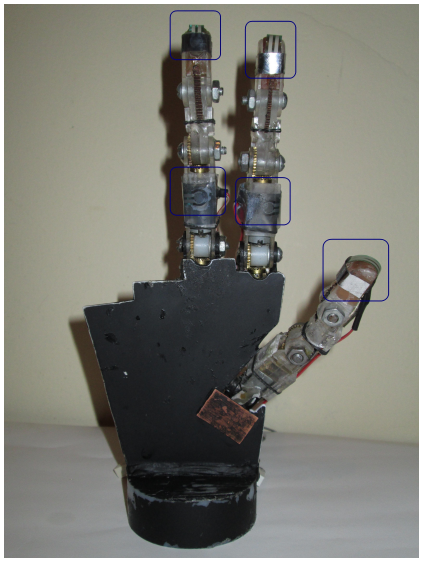


Figura 3.9: Sensores FSR adaptados a la mano.

3.2.2.1. Características técnicas del sensor

La señal del sensor se adecuó a través de un divisor de tensión y un amplificador operacional en configuración seguidor-emisor. Los sensores FSR se comportan como una resistencia variable que cambia desde su valor nominal al aplicarle una carga. En la *figura 3.10* se muestra el circuito de acondicionamiento para el sensor FSR. Este es tomado de [23] siguiendo la recomendación que se sugiere escoger $RM = 3k\Omega$

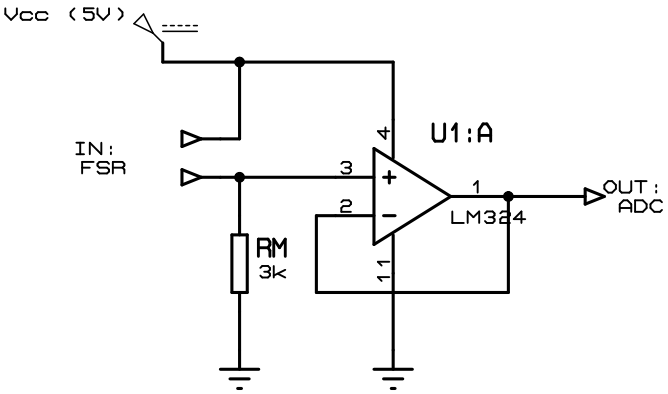


Figura 3.10: Circuito de acondicionamiento para sensor de fuerza FSR.

La curva que relaciona la *carga (N) v.s el voltaje (V)* se muestra en la *figura 3.11*.

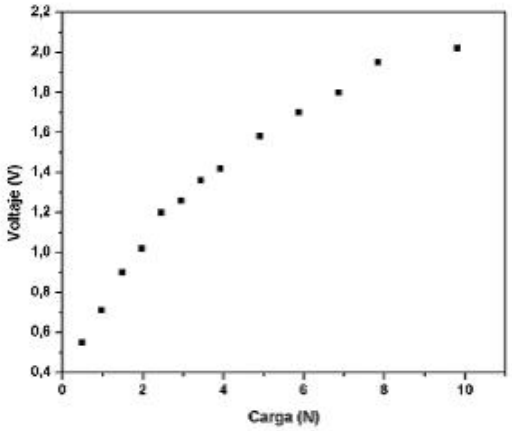


Figura 3.11: Caracterización sensor FSR.

3.2.3. Tarjeta de sensado y acondicionamiento de señales

Se diseñó una tarjeta que acondiciona las tensiones que ingresan al *DSP*, haciendo que la planta quede bien instrumentada: por ejemplo, una variación entre el ángulo mínimo y el ángulo máximo de una articulación, represente una variación de tensión entre 0.4 y 3 voltios respectivamente. En la *figura 3.12* se muestra la tarjeta donde se implementan los circuitos de acondicionamiento de señal para los sensores de *deflexión* y *FSR*.

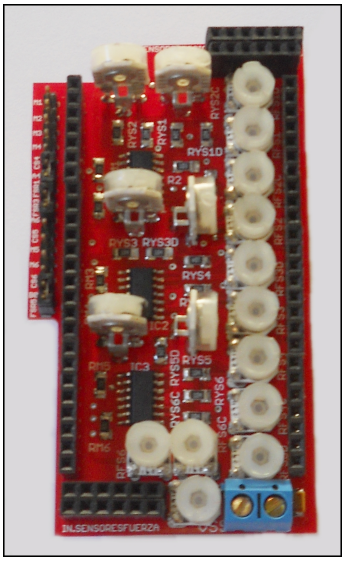


Figura 3.12: Tarjeta de acondicionamiento señales de sensores de posición y fuerza.

Descripción	Valor
Voltaje polarización	5V
Corriente	150mA
Entrada sensores posición	6
Entrada sensores fuerza	6
Entrada sensores corriente	3
Salidas	15
Voltaje salida	0.4 a 3V

Tabla 3.6: Especificaciones de la tarjeta de acondicionamiento de señales de posición y fuerza.

En la *Tabla 3.6* se especifican las características principales de la tarjeta diseñada. El *PCB* fue diseñado en Eagle®, un software con versión de prueba que permite realizar diseños en dos capas con dimensiones no superiores a $100mm \times 80mm$. Posteriormente se enviaron a elaboración a una empresa dedicada a la producción y ensamble de *PCBs*. En el anexo A se encuentra de forma más detallada las conexiones. Las dimensiones de esta tarjeta se ajustaron a distancia entre los 40 pines de la Tarjeta de control FiO Std mostradas en la *figura 3.2*.

3.2.4. Sensor de corriente

Se seleccionó un sensor de *efecto Hall* por sus características de operación: rango de medición de ±5 amperios, error de salida de 1.5 % a $T_A = 25^oC$. Además el ancho de banda del dispositivo puede ser configurado mediante un filtro a través de uno de sus pines [24].

3.2.4.1. Acondicionamiento de señales de corriente

Se diseñó un circuito con un amplificador operacional en configuración diferencial, que ajustará el rango de medición de *-5A a +5A* a uno *-1.6A a +1.6A*, siendo esta la *corriente de Stall* del motor (dato del fabricante). Con la ayuda del software *Proteus* se implementó el circuito de la *figura 3.14*, sin conectar las entradas del amplificador operacional a la salida del sensor de corriente. Cuando la corriente por el motor es *-1.6A* se desea una tensión aproximada a cero en la salida del amplificador operacional. También, por medio de la recta mostrada en la *figura 3.13* es posible encontrar los valores de salida de voltaje del sensor de corriente para *-1.6A y +1.6A*. Se observa entonces que cuando la corriente por el motor es *-1.6A* se tiene un $vs_{out} = 2{,}18v = v_1$ (v_1 es la tensión de referencia). Ahora cuando la corriente por el motor es *1.6A* $vs_{out} = 2{,}82v = v_2$.

Resolviendo de la *(Ec. 3.1)* para Rf y haciendo $Ry = 100k\Omega$ y $v_{out} = 3{,}2$ (valor máximo deseado para la corriente máxima). Se tiene:

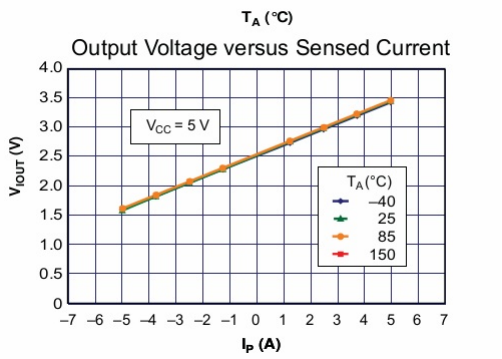


Figura 3.13: Curva característica sensor de corriente. [24]

$$Rf = \frac{v_{out} \cdot Ry}{(v_2 - v_1)} Rf = \frac{3{,}2 \cdot 100k\Omega}{(2{,}82 - 2{,}18)} = 500k\Omega \tag{3.2}$$

Para seleccionar las resistencias R1 y R2 se determina la salida del divisor de tensión como sigue:

$$v_{R1} = \frac{R1}{R1 + R2)} \cdot v_f \tag{3.3}$$

Seleccionando $R1 = 20k\Omega$ y $V_f = 4{,}8v$, $v_{R1} = v_1$ resolviendo en la *(Ec. 3.3)* para $R2$ se tiene:

$$R2 = \frac{R1 \cdot (v_f - v_1}{v_1} = \frac{20k\Omega \cdot (4{,}8 - 2{,}18)}{2{,}18} = 24{,}036k\Omega \approx 24k\Omega \tag{3.4}$$

Cuando la corriente circundante por el motor es nula, se obtiene una tensión fija de $1{,}6v$ aproximadamente. Las corrientes negativas disminuyen la tensión hacia el adc y las corrientes positivas la aumentan.

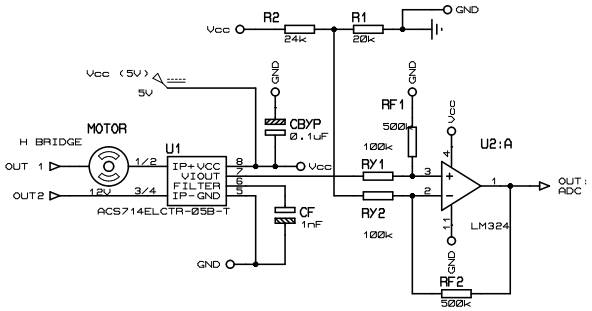


Figura 3.14: Circuito de acondicionamiento para sensor de corriente.

3.2.4.2. Tarjeta de sensado de corriente

Se diseñó una tarjeta para la medición de corriente en cada uno de los motores. Ésta contiene seis sensores de efecto Hall: *ACS714ELCTR-05B-T*.

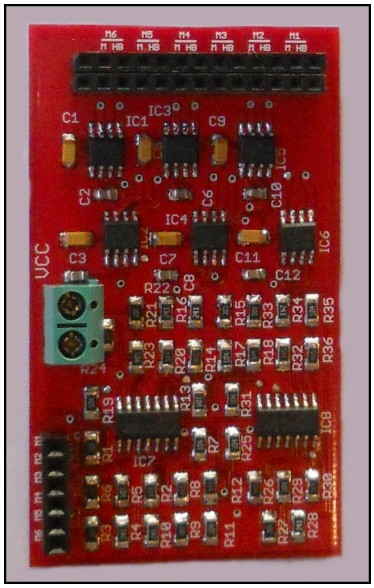


Figura 3.15: Tarjeta de sensado de corriente.

Descripción	Valor
Voltaje polarización	5V
Corriente	100mA
Entrada motores	6
Salidas	6
Voltaje salida	0 a 3V

Tabla 3.7: Especificaciones tarjeta sensado de corriente.

En el Anexo A se encuentra de forma más detallada las conexiones e instrucciones de manejo de la tarjeta de sensado de corriente.

3.3. Elemento final de control

Los actuadores utilizados son el conjunto de los micromotores *DC* con caja reductora y los drivers *L298P*. Mediante la técnica de modulación por ancho de pulso *(PWM)* es posible variar el voltaje aplicado a los motores. La señal de mando ó esfuerzo de control

u(t), determina el porcentaje del periodo que se deben aplicar a los motores y el *signo de u(t)* determina el sentido de giro. Cada driver permite manejar las articulaciones de un dedo. En la *figura 3.16* se muestra el circuito de potencia para el manejo de dos motores.

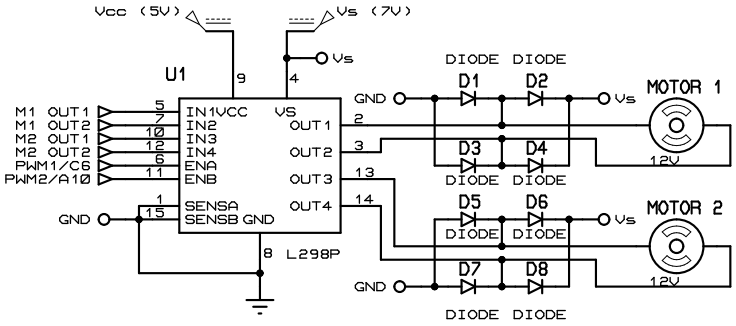


Figura 3.16: Circuito de potencia.

3.3.1. Tarjeta de potencia

Se diseñó una tarjeta que incluye los tres drivers *L298P* que permiten manipular el giro y voltaje aplicado a los motores de las articulaciones, *figura 3.17*.

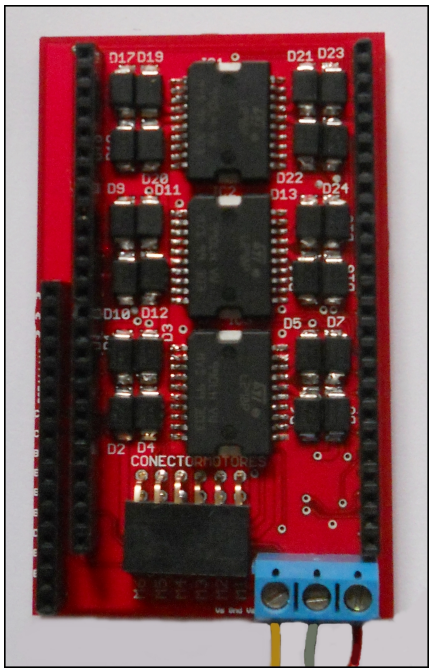


Figura 3.17: Tarjeta de potencia.

La siguiente *Tabla 3.8* muestra las especificaciones de voltajes máximos de

polarización y de salida, así como también las corrientes máximas circundantes en la placa.

Descripción	**Valor**
Voltaje polarización	5V
Voltaje alimentación motores	9V
Corriente *L298P*	210mA
Corriente en los motores	12A
Salidas para motores	6
Voltaje salida	0 a 9V

Tabla 3.8: Especificaciones tarjeta potencia.

3.4. Conexión entre tarjetas

Las tarjetas fueron diseñadas siguiendo el diseño de la tarjeta de FiO Std, para que se interconecten entre sí por medio de regletas, evitando el uso de cables como se ve a continuación en la *figura 3.18*.

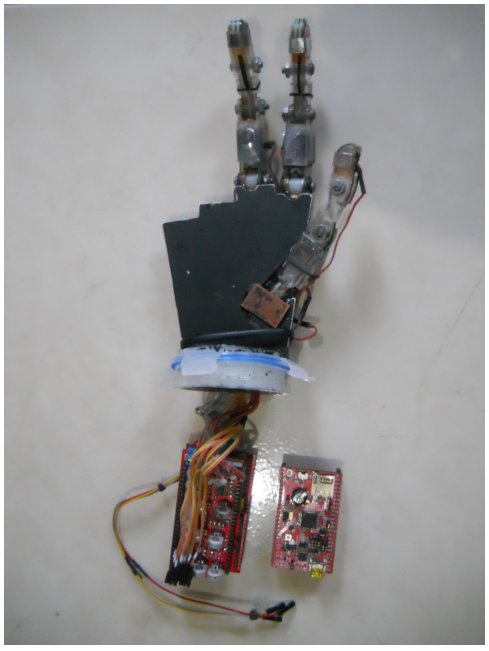


Figura 3.18: Tarjeta Fio Std separada de instrumentación de sensores y potencia.

En la *figura 3.19* se distinguen los tres elementos básicos de un esquema de control *feedback*, controlador (tarjeta FiO Std), elemento primario de control (tarjeta de

acondicionamiento de señales) y elemento final de control (tarjeta de potencia).

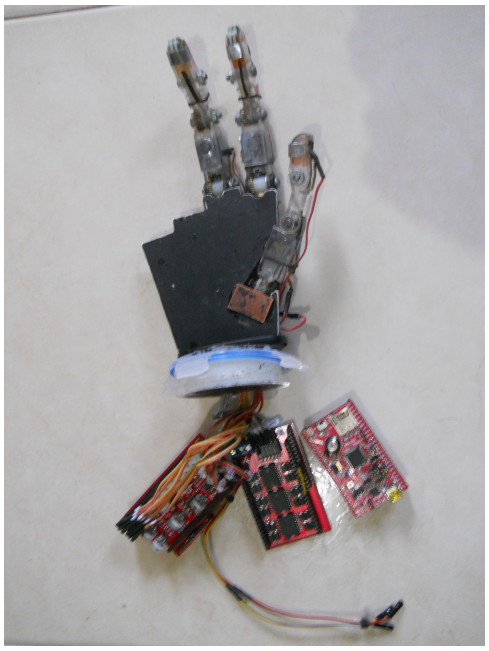


Figura 3.19: Tarjeta FiO Std, acondicionamiento señales de sensores y de potencia.

A continuación se presentan las tablas que relacionan el cableado de los sensores y actuadores con las borneras en las tarjetas diseñadas, así:

En la *Tabla 3.9* se tiene la conexión hecha en la bornera P con los sensores de posición asociada a cada articulación, de igual manera la *Tabla 3.10* muestra la conexión en la bornera F con los sensores de fuerza y la *Tabla 3.11* la conexión en la bornera M con los motores.

Bornera	**Conector de salida**	**Sensor de llegada**	**Articulación asociada**
P	P1	Sensor de posición 1 (SP1)	MCP dedo índice
	P2	Sensor de posición 2 (SP2)	PIP dedo índice
	P3	Sensor de posición 3 (SP3)	MCP dedo medio
	P4	Sensor de posición 4 (SP4)	PIP dedo medio
	P5	Sensor de posición 5 (SP5)	MCP dedo pulgar
	P6	Sensor de posición 6 (SP6)	PIP dedo pulgar

Tabla 3.9: Conexión bornera P con sensores de posición.

Bornera	Conector de salida	Sensor de llegada
F	F1	Sensor de fuerza 1 (FS1)
	F2	Sensor de fuerza 2 (FS2)
	F3	Sensor de fuerza 3 (FS3)
	F4	Sensor de fuerza 4 (FS4)
	F5	Sensor de fuerza 5 (FS5)
	F6	Sensor de fuerza 6 (FS6)

Tabla 3.10: Conexión bornera F con sensores de fuerza.

Bornera	Conector de salida	Actuador de llegada	Articulación asociada
M	M1	Motor 1 (M1)	MCP dedo índice
	M2	Motor 2 (M2)	PIP dedo índice
	M3	Motor 3 (M3)	MCP dedo medio
	M4	Motor 4 (M4)	PIP dedo medio
	M5	Motor 5 (M5)	MCP dedo pulgar
	M6	Motor 6 (M6)	PIP dedo pulgar

Tabla 3.11: Conexión bornera M con motores.

En la *figura 3.20* se observa el etiquetado del cableado y las borneras de conexiones del mismo. El etiquetado se realizó siguiendo las recomendaciones del estándar ISA 5.1 [25].

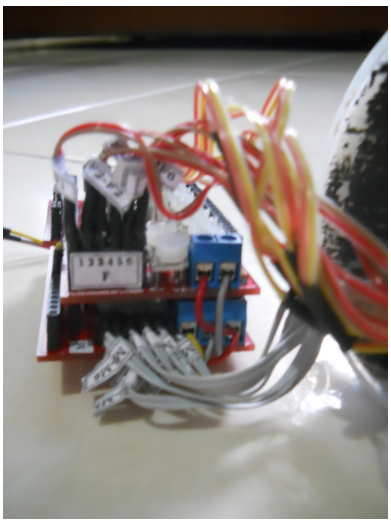


Figura 3.20: Etiquetado para cables de sensores y actuadores.

Los sensores de posición y fuerza fueron conectados a la tarjeta FiO Std como aparece a continuación en la *Tabla 3.12*.

Señal	**función**
SP1	Canal analógico C5
SP2	Canal analógico C4
SP3	Canal analógico C3
SP4	Canal analógico C2
SP5	Canal analógico C1
SP6	Canal analógico C0
FS1	Canal analógico A4
FS2	Canal analógico A5
FS3	Canal analógico A6
FS4	Canal analógico A7
FS5	Canal analógico B0
FS6	Canal analógico B1

Tabla 3.12: Canales analógicos.

La *Tabla 3.13* relaciona los canales de salida de *PWM* y las salidas digitales de configuración de giro para cada una de las articulaciones.

Señal	**función**
PWM1	Señal PWM CH1/C6 MCP índice
PWM2	Señal PWM CH3/A10 PIP índice
PWM3	Señal PWM CH2/B7 MCP medio
PWM4	Señal PWM CH2/C7 PIP medio
PWM5	Señal PWM CH4/C9 MCP pulgar
PWM6	Señal PWM CH1/B6 PIP pulgar
M1O1	Salida digital C13
M1O2	Salida digital C12
M2O1	Salida digital B15
M2O2	Salida digital B14
M3O1	Salida digital B13
M3O2	Salida digital B12
M4O1	Salida digital C10

M4O2	Salida digital C11
M5O1	Salida digital A13
M5O2	Salida digital A15
M6O1	Salida digital B11
M602	Salida digital B10

Tabla 3.13: Señales *PWM* y pines de giro del motor.

En la *figura 3.21* se observa el montaje de las tarjetas diseñadas interconectadas en el soporte del prototipo mecánico.

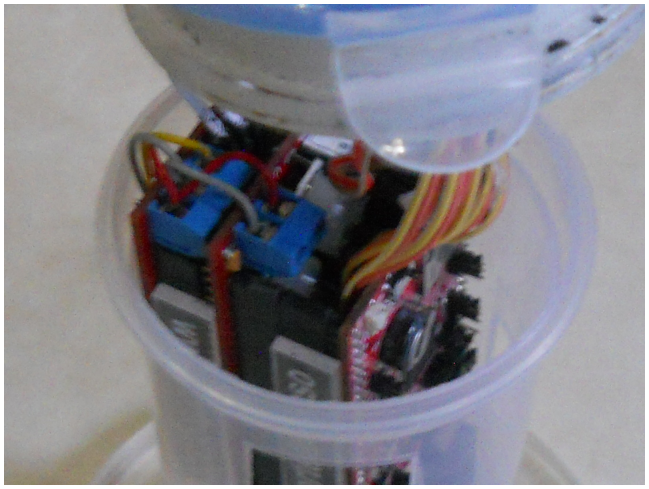


Figura 3.21: Montaje hardware final para el control del prototipo de mano robótica.

Capítulo 4

Interfaz gráfica para diseño y programación de controladores

4.1. Interfaz gráfica de usuario en Matlab®

Una interfaz gráfica es el vínculo que existe entre el usuario y un programa computacional, generalmente se constituye por un conjunto de comandos o menús, instrumentos y métodos por medio de los cuales el usuario se comunica con el programa durante las operaciones que se desean realizar, facilitando la entrada y salida de datos e información. Una interfaz es una de las partes más importantes de cualquier programa, debido a que ésta determina la precisión y desempeño del programa ante los comandos que el usuario pretenda ejecutar. El término en inglés de las interfaces gráficas es Graphical User Interface, denominándolas GUI, en adelante se referirán de la misma manera [26].

Matlab® permite realizar GUIs de forma sencilla usando la herramienta GUIDE (Graphical User Interface Development Environment), la cual está especialmente diseñada para la creación de GUIs, reduciendo la labor al grado de seleccionar, arrastrar y personalizar propiedades.

A diferencia de la ejecución de funciones o scripts de Matlab®, la ejecución de GUIs no predetermina el flujo de ejecución del código. Es el usuario, que a través de la interacción con la GUI, el que determina el orden en que se ejecutan los diferentes comandos o funciones desarrolladas. Otra diferencia es que la ejecución no termina cuando finaliza la ejecución del script o función, sino que la GUI permanece abierta, permitiendo al usuario invocar la ejecución de uno u otro código desarrollado [27].

Una GUI se compone de dos archivos uno *.m y otro *.fig. Cada vez que se adiciona un elemento a la interfaz, de forma automática se genera una función correspondiente en el archivo *.m. El archivo *.m está compuesto por comandos y funciones, que contienen la secuencia de sentencias para la ejecución de la aplicación. En el archivo *.fig se visualizan todos los objetos (botones, gráficas, editores de texto, etcétera) que interactúan con el usuario y hacen posible controlar el proceso para el que fue diseñado [28].

4.2. Descripción general de la interfaz gráfica de usuario

Esta interfaz tiene como objetivo manejar archivos de Simulink®, en los cuales se realiza el diseño y pruebas de diferentes tipos de algoritmos de control sobre un prototipo de mano virtual. Estos diseños una vez verificados y puestos a punto son embebidos de forma rápida y sencilla con el manejo de algunos comandos de Simulink® desde la interfaz, para ser probados posteriormente sobre el prototipo de mano robótica real. Una vez que finalice este proceso se podrá generar un reporte con figuras de los diagramas de bloques del modelo y sus subsistemas, además de un resumen de los archivos de código generados.

4.3. Descripción específica de los paneles

4.3.1. Panel de control general

En la *figura 4.1*, se muestra el panel general de la interfaz de usuario diseñada para la programación de controladores.

Figura 4.1: Panel de control general de la interfaz gráfica de usuario.

En los siguientes apartados se explicará punto por punto cada uno de los elementos que conforman la GUI diseñada.

4.3.2. Barra de herramientas

En primer lugar se encuentra la barra de herramientas en la parte superior de la interfaz, *figura 4.2*.

Figura 4.2: Barra de herramientas.

En ella se puede elegir entre algunas acciones comunes que ofrece Windows y que son de gran utilidad para el manejo de la interfaz gráfica.

Entre las acciones de la barra de tareas, nos encontramos:

- Archivo: El cual muestra "Abrir documentos", con lo que se puede buscar para abrir cualquier documento con extensión *.m, *.mdl, *.docx y *.pdf, en el momento en que el usuario lo requiera direccionándolo sobre los discos y carpetas del computador.

- Simulink®: Con esta acción se despliega la ventana de Simulink®, donde el usuario podrá crear nuevos documentos si los requiere.

- Ayuda: Esta acción es de gran utilidad para el usuario ya que podrá visualizar un documento que le servirá como guía general para el manejo de la interfaz gráfica.

- Reset: Cumple la función de reinicio dejando los archivos en estado inicial.

- Graficar: El usuario podrá desplegar una gráfica comparativa del seguimiento de las posiciones en las articulaciones, verificando el funcionamiento del controlador en el sistema.

4.3.3. Panel de control principal

En la parte izquierda se muestra el panel más importante de la aplicación llamado Nuevo Controlador, *figura 4.3*, en él se encuentran una serie de botones los cuales sirven para interactuar con algunos archivos y comandos de Simulink® en la creación de nuevos modelos, pasando desde la simulación en el modelo virtual hasta la descarga en la tarjeta que gobierna la prótesis de mano real.

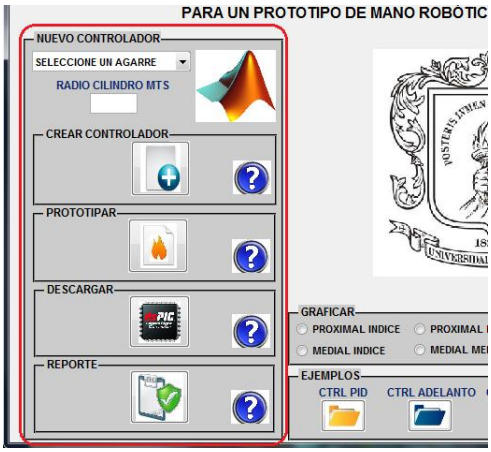


Figura 4.3: Panel principal.

Entre las acciones que se pueden realizar en este panel se tienen:

4.3.3.1. Seleccionar un tipo de agarre

Con esta acción se despliega un submenú (*figura 4.4*), que da la opción de escoger entre distintos tipos de agarres, cargando en los archivos de Simulink® las distintas posiciones correspondientes a las articulaciones o también permite digitar de forma manual en una GUI adicional los ángulos en grados para cada articulación.

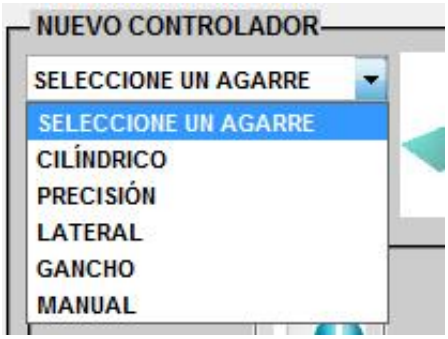


Figura 4.4: Selección de agarres.

Como se puede observar en la anterior figura entre las opciones que el usuario tiene acceso son cilíndrico, precisión, lateral, gancho, manual, estos agarres son definidos en la sección 5.5 del Capítulo 5, por lo que solo se procederá a mostrar en la siguiente *figura 4.5*, un ejemplo en caso de que se seleccione el agarre cilíndrico.

Figura 4.5: Selección del agarre cilíndrico.

Al escoger esta opción se abren por un breve instante de tiempo dos archivos de Simulink® donde se cargan las posiciones para cada articulación y también se muestra una pequeña imagen con la forma del agarre seleccionado. De igual forma sucede cuando se escoge alguno de los otros agarres.

Por otra parte el usuario tiene la opción de digitar las posiciones que desea para cada articulación, esto lo puede hacer al seleccionar la opción manual. En esta ocasión se le despliega una GUI adicional como se muestra a continuación, *figura 4.6*.

Figura 4.6: Interfaz gráfica adicional.

En esta interfaz son digitados los ángulos en grados para cada articulación y luego son cargados a los archivos de Simulink® al pulsar el botón "Cargar Posiciones".

4.3.3.2. Crear controlador

Este subpanel desarrolla los pasos para que el usuario pueda construir un nuevo controlador, aquí están dispuestos dos botones como se ve a continuación, *figura 4.7*.

Figura 4.7: Subpanel crear controlador.

Al pulsar el primer botón se despliega un archivo de Simulink® (*figura 4.8*), en el cual el usuario podrá construir los controladores para cada articulación y posteriormente probarlos en simulación.

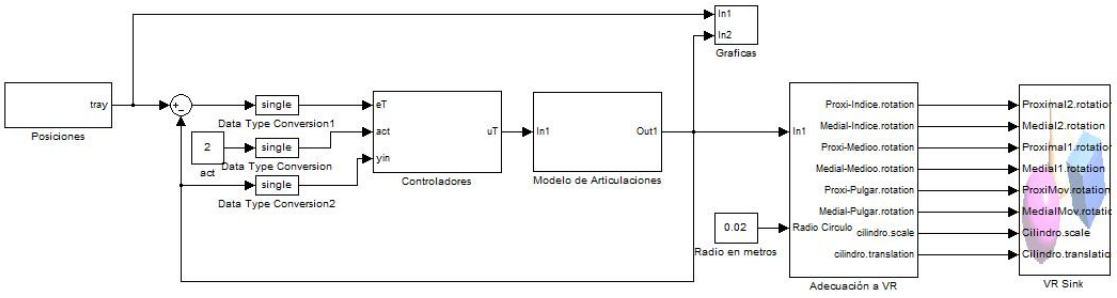


Figura 4.8: Modelado.mdl.

Como se puede observar en la anterior figura el archivo está compuesto por ocho bloques.

- Posiciones: Aquí se encuentran las posiciones deseadas para cada articulación que se quiere que el modelo virtual siga, y como se observará a continuación se hace la conversión por medio de ganancias a radianes, ya que todo el modelo trabaja en esta unidad de medida, *figura 4.9*.

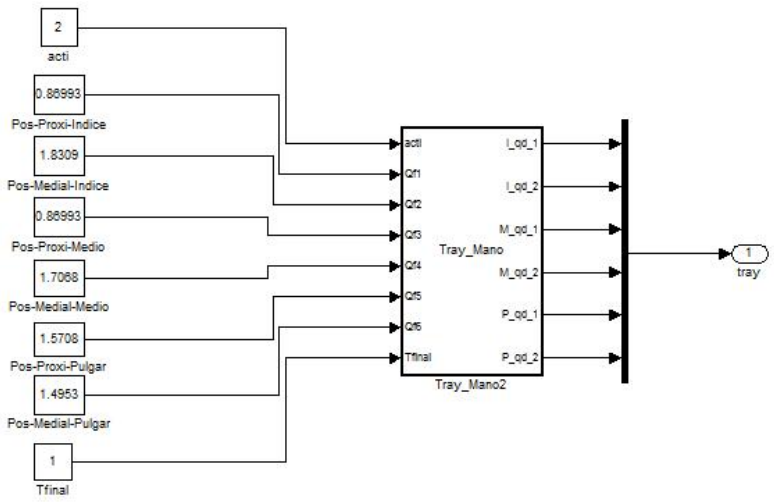


Figura 4.9: Bloque Posiciones.

- Controladores: En este bloque recae el mayor esfuerzo para el usuario debido a que aquí se construirán los controladores para cada articulación, se debe tener en cuenta que se usaron datos tipo *single*, esto para minimizar el consumo de memoria en la tarjeta de la prótesis cuando se embebe el controlador, por esto a la entrada del controlador se hace una previa conversión del tipo de dato mediante el bloque *Data Type Conversion*. En el bloque *Embedded Matlab Fcn* es donde se escribirá el código para los controladores, en la *figura 4.8.* se observa que a este bloque le ingresa un valor a través de una constante, que corresponde al estado de la prótesis de mano, esto se explicará de forma detallada en la sección D.3.4. del Anexo D. A continuación se muestra el resultado del bloque Controladores para el caso del Controlador PID, *figura 4.10.*

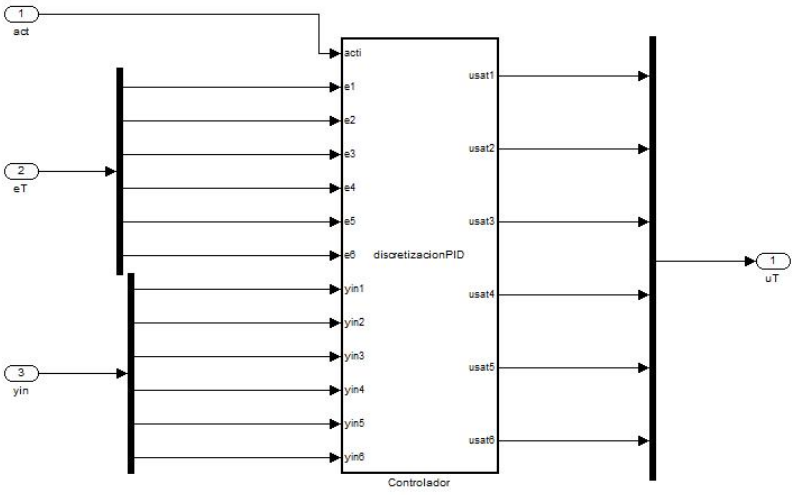


Figura 4.10: Bloque Controladores.

- Modelo de Articulaciones: En este bloque se encuentra el modelo identificado para cada articulación, *figura 4.11*.

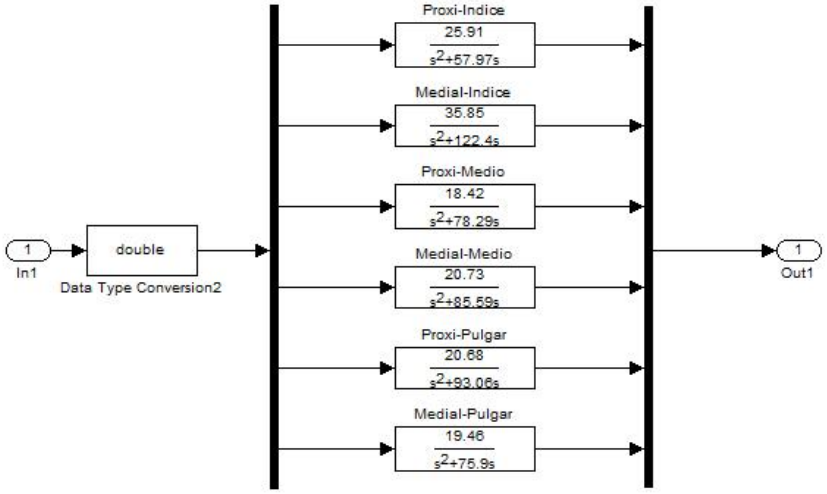


Figura 4.11: Modelo identificado de las articulaciones.

- Gráficas: En este bloque se grafican las señales para ir observando el desempeño de los controladores, estas gráficas podrán ser visualizadas dentro del panel general de la aplicación, *figura 4.12*.

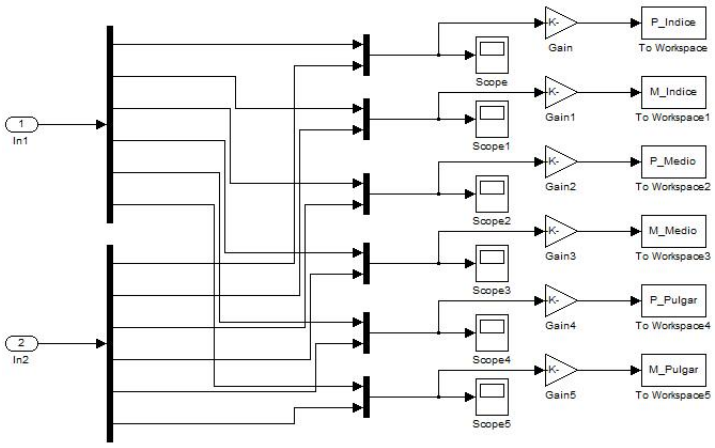


Figura 4.12: Bloque Gráficas.

- Adecuación a VR: Este bloque tiene como función manejar el eje de rotación de las articulaciones en el prototipo virtual, *figura 4.13*.

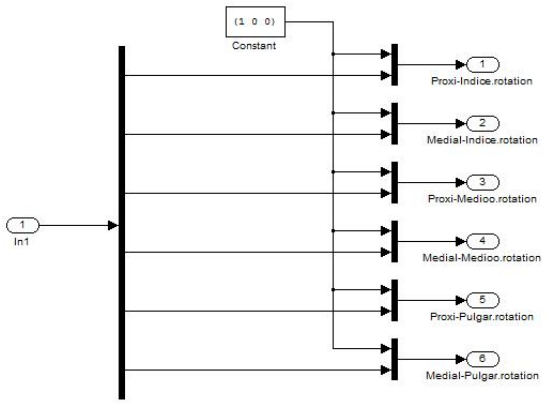


Figura 4.13: Bloque Adecuación a VR.

- VR Sink: Al desplegar este bloque se abre una ventana con el modelo del prototipo virtual de la mano, para observar en simulación el desempeño de los controladores diseñados, *figura 4.14*.

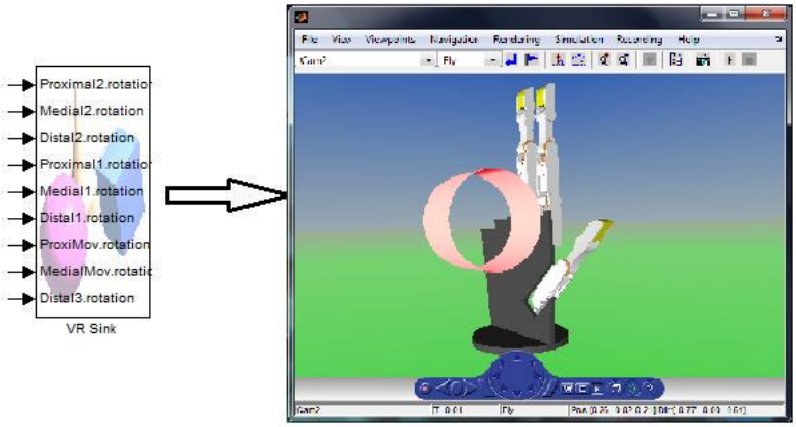


Figura 4.14: Bloque VR Sink.

El segundo botón que se encuentra en el subpanel Crear controlador es la ayuda para este paso, al ser pulsado este botón se despliega una ventana que contiene una breve explicación de lo que el usuario podrá y deberá realizar aquí.

4.3.3.3. Prototipar

Como se puede observar a continuación en la *figura 4.15*, este subpanel también está compuesto por dos botones.

Figura 4.15: Subpanel prototipar.

El primero se encarga de ejecutar una función la cual realiza una copia de los controladores creados y puestos a punto en el subpanel anterior, pegándolos en un nuevo archivo que es desplegado en el módulo controladores, *figura 4.16*.

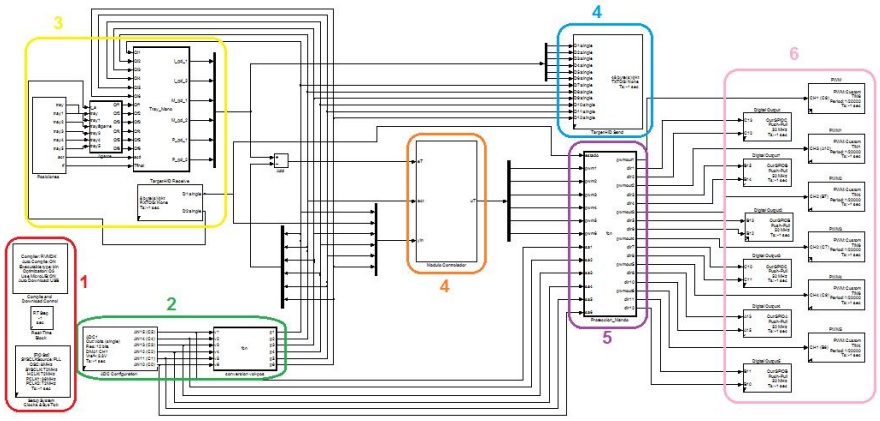


Figura 4.16: Descarga.mdl.

Como se puede ver en la anterior figura el archivo se compone de varios bloques, los cuales pertenecen a siete módulos que se definen así:

1. Módulo de configuraciones del *DSP*: Este módulo se compone de los bloques *Compile and Download control, Setup System Clockls y SysTick* los cuales realizan las configuraciones básicas del *DSP* y configuran la compilación y programación directa desde Simulink®.

2. Módulo sensado: Este módulo contiene la configuración de los canales ADCs asociados a las seis articulaciones. También lo constituye una *Embedded Matlab Fcn* que contiene la ecuación característica de los seis sensores de posición.

3. Módulo de consignas: Comprendido por una *Embedded Matlab Fcn* que calcula trayectorias o escalones para los diferentes tipos de agarre, asegurándose que al salir del reposo al estado de control, la referencia tenga el mismo valor de la salida, a fin de iniciar con error pequeño.

4. Módulo de control de posición: En este módulo se implementan los algoritmos o leyes de control.

5. Módulo comunicación: Se compone de bloques que permiten enviar y recibir datos de hasta 64 bytes por paquete. Estos son usados para enviar el estado de la mano (reposo, control o libre) y el tipo de agarre a ejecutar. Por su parte la tarjeta de control envía los datos provenientes de los sensores y la referencia asociada a cada articulación.

6. Módulo de mando y protección: Este bloque escribe directamente sobre los registros que gobiernan la entrada de potencia y giro al motor de las articulaciones. Con este módulo se controla el estado de la mano en general.

7. Módulo actuadores: Lo componen los *PWM* y Digital Output uno por cada articulación. Este módulo interactúa directamente con la prótesis de mano real, a través del puente H, a quien se le asigna un sentido de giro de los motores y la potencia aplicada a estos mismos.

4.3.3.4. Descargar

En este subpanel al igual que en los anteriores se disponen dos botones como se ve a continuación en la *figura 4.17*.

Figura 4.17: Subpanel descargar.

Al ser pulsado el botón central de este módulo se ejecuta una función que se encarga de la creación y descarga de los controladores, embebiéndolos sobre la tarjeta que gobierna la prótesis de mano real, con lo que se podrá ver su comportamiento. También se encuentra el botón de ayuda. Al terminar se despliega otra interfaz *figura 4.18*, por medio de la cual se controla la ejecución de la ley de control pulsando los botones que ahí se presentan. Los detalles de ésta GUI se explicarán más adelante en el Anexo D correspondiente al manual de usuario.

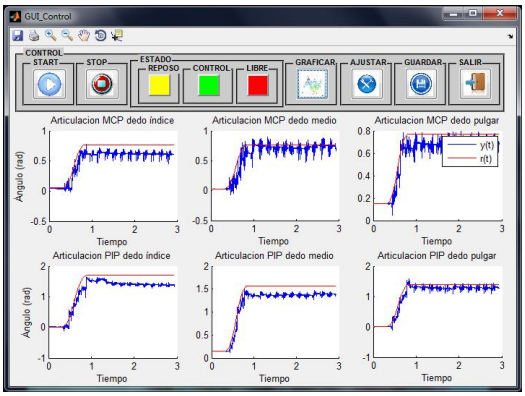


Figura 4.18: GUI ejecución y control.

4.3.3.5. Reporte

Este subpanel es el último del panel de control principal, como en todos se compone de dos botones, *figura 4.19*.

Figura 4.19: Subpanel Reporte.

Al pulsar el botón se ejecuta una función que generará un archivo *.html, que documenta el código generado, los diagramas de bloques y sus subsistemas, ajustes de configuración y un informe de trazabilidad. A continuación se puede observar en la *figura 4.20* un bosquejo del informe generado.

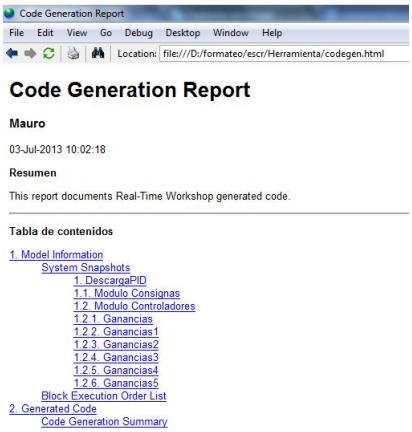


Figura 4.20: Contenido del reporte generado.

4.3.3.6. Graficar

En este panel se extraen las gráficas de posición de las articulaciones, con lo que se puede realizar la comparación del seguimiento que tiene la señal con la consigna o posición deseada y realizar ajustes, *figura 4.21*.

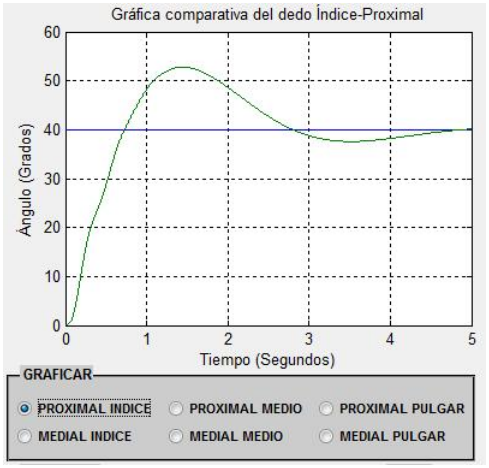


Figura 4.21: Gráfica comparativa del dedo Índice Proximal.

Al pulsar cualquiera de los radio button se generará la simulación del archivo inicial de Simulink® donde se están construyendo los controladores, y luego de algunos segundos se mostrará la gráfica correspondiente, cambiando la imagen del escudo de la Universidad del Cauca que se tenía. Al ser pulsado nuevamente volverá a visualizarse la imagen como estaba inicialmente.

4.3.4. Ejemplos

En este panel el usuario podrá acceder a los archivos de los controladores diseñados y probados en este trabajo de grado, con el fin de que le sirvan de guía cuando se disponga a construir el propio, *figura 4.22*.

Figura 4.22: Panel de ejemplos de los controladores diseñados.

Para el caso del controlador por adelanto de desarrolló una GUI auxiliar (*figura 4.23*)

con la cual el usuario podrá diseñar de una forma sencilla los controladores, digitando los parámetros de la planta y los parámetros de diseño.

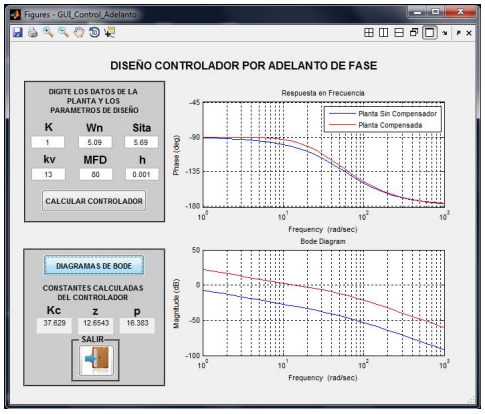


Figura 4.23: GUI para controlador por adelanto.

4.3.5. Salir

Este panel cumple una función sencilla pero muy importante, que es la de terminar la ejecución de la aplicación finalizando la construcción de los modelos de controladores, *figura 4.24*.

Figura 4.24: Panel para salir de la aplicación.

Con este panel se termina la explicación del archivo *.fig de la GUI, luego se procede a guardarla para generar el archivo *.m. Toda la programación y código del archivo *.m que se hace en las funciones que corresponden a cada elemento se mostrará y explicará en detalle en el Anexo C que se adjunta a este documento.

Capítulo 5

Simulación y pruebas experimentales

5.1. Identificación de una articulación

En el Anexo F, se obtuvo un modelo matemático para un motor DC, para el cual la entrada es el voltaje $V_a(s)$ y la salida es la posición angular del rotor $\theta(s)$. Para el caso particular de una articulación del prototipo de mano robótica, $\theta(s)$ es limitado por el ángulo mínimo y el ángulo máximo que pueda recorrer. Estos ángulos fueron mostrados en la *Tabla 2.3*. Se observa también que en la *(Ec. F.19)* existe la presencia de un integrador. Estas dos condiciones hacen que no pueda realizarse una toma de datos de entrada y de salida en lazo abierto y con una señal rica en componentes de frecuencia, por ejemplo un escalón que sería ideal para un proceso de identificación. Esta es una situación práctica donde es imposible remover la realimentación. Según [29] la principal diferencia entre identificación en lazo cerrado e identificación en lazo abierto es el origen de los datos y no las técnicas de estimación empleadas.

5.1.1. Experimento para adquisición de datos

Para la adquisición de datos de una articulación fue cerrado el lazo de retroalimentación con un controlador PID, el cual se calibró mediante un ejercicio experimental que permitió obtener valores de las constantes proporcional K_p, integral K_i y derivativa K_d, de forma tal que la respuesta del sistema sea acotada, situación que no es posible en lazo abierto debido a la acción integral inmersa en la dinámica de la planta.

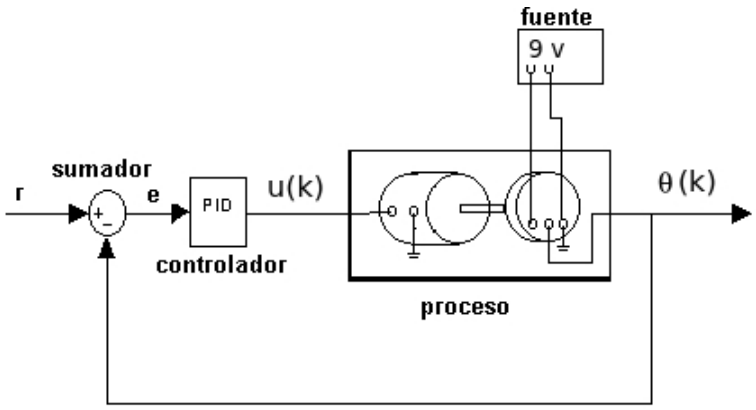


Figura 5.1: Adquisición de datos en lazo cerrado para identificación.

El esquema mostrado en la *figura 5.1*, fue implementado para la adquisición de datos de entrada y de salida. $u(t)$ es el esfuerzo de control que sale del controlador PID, el cual representa un porcentaje de periodo de PWM y cuyo signo determina el sentido de giro del rotor. Si la ley de control envía un esfuerzo de control del 100 % se aplica la tensión total de la fuente de alimentación del motor, así por ejemplo para el experimento realizado con una fuente de $9v$ un porcentaje de 60 % PWM representa una tensión de entrada al motor de $5{,}4v$. El ángulo de salida es leído en términos de voltaje mediante un conversor analógico digital. En la *figura 5.2*, se pueden observar los datos obtenidos para una articulación MCP del dedo índice.

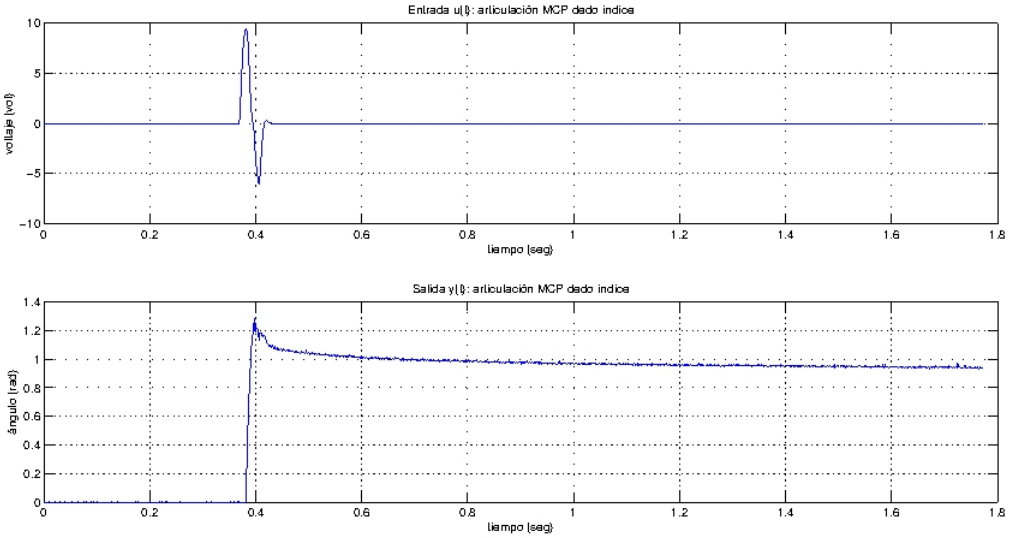


Figura 5.2: Datos de entrada y salida.

5.1.2. Estimación de parámetros

Con los datos obtenidos mediante el experimento mencionado anteriormente, se procedió a utilizar el *toolbox de Matlab®/Simulink® Parameter Estimation*. Con esta herramienta se estiman los parámetros J_m y f_m de la estructura relacionada en la *(Ec. F.19)*. De la *(Ec. F.4)* en régimen permanente se tiene:

$$v_a = R_a \cdot i_a + K_b \cdot w_m \tag{5.1}$$

Como $k_b = k_m$

$$v_a = R_a \cdot i_a + K_m \cdot w_m \tag{5.2}$$

$$k_m = \frac{v_a - R_a \cdot i_a}{w_m} \tag{5.3}$$

Si se aplica un freno al motor mediante un par de carga, hasta lograr $w_m = 0$ se puede definir de la *(Ec. 5.4)*:

$$R_a = \frac{v_a}{i_a} \tag{5.4}$$

De los datos del fabricante sabemos que el voltaje nominal $v_a = 6v$ y la corriente de stall $i_a = 1{,}6A$, se puede calcular R_a como sigue:

$$R_a = \frac{6v}{1{,}6A} = 3{,}75\Omega \tag{5.5}$$

De los datos del fabricante conocemos que la velocidad angular nominal $w_m = 100rpm = 10{,}47rad/seg$ y la corriente sin carga $i_a = 70mA$ se puede calcular k_m usando la *(Ec. 5.3)*:

$$k_m = \frac{6v - 3{,}75\Omega \cdot 70 \cdot e^{-3}A}{10{,}47rad/seg} = 0{,}548\frac{v}{rad/seg} \tag{5.6}$$

Para la articulación MCP se obtuvo:

Parámetro	**Valor**	**Unidades**
J_m	0,004037	$K_g \cdot m^2$
f_m	0,17672	$K_g \cdot m^2/seg$

Tabla 5.1: Parámetros estimados para articulación MCP dedo índice.

Con estos valores se calculan w_o mediante la *(Ec. F.21)* y ξ con la *(Ec. F.22)* para

obtener un modelo descrito por la *(Ec. F.20)*, en donde existe un polo dominante en $2 \cdot \xi \cdot w_o$. La forma reducida de un motor que posee dos polos reales, uno de ellos más cercano al integrador conocido como polo dominante del motor DC. Finalmente:

$$w_o^2 = 25{,}91 \tag{5.7}$$

$$2 \cdot \xi \cdot w_o = 5{,}09 \tag{5.8}$$

Un modelo matemático para la articulación MCP del dedo índice, puede ser descrito por la *(Ec. 5.9)*

$$G_1(s) = \frac{25{,}91}{s \cdot (s + 57{,}97)} \tag{5.9}$$

5.1.3. Validación del modelo

La validación del modelo, se puede observar en la *figura 5.3*, de la cual se puede afirmar que existe una fuerte relación entre los datos de la planta y la salida del modelo. Sin embargo existe una dinámica mostrada en los datos de salida que no corresponden al verdadero comportamiento de la planta, esto es que el ángulo final en el tiempo de estado estable tiende a disminuir continuamente, algo que no es posible debido que el esfuerzo de control ha disminuido a cero, además esto representaría que la articulación está haciendo un movimiento de extensión, algo que no sucede, incluso el par de carga ejercido por la estructura del dedo generaría un movimiento de recogimiento del dedo, por lo tanto este comportamiento obedecería a alguna caída de tensión en el circuito que alimenta el flexsenor.

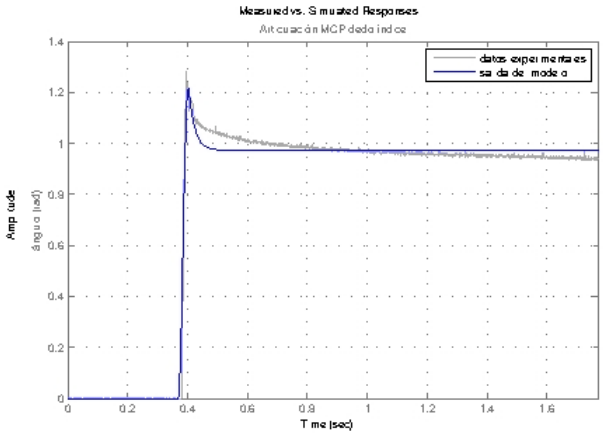


Figura 5.3: Salida del modelo.

Este método experimental fue usado para identificar las cinco articulaciones restantes. Estos modelos son contenido en la *Tabla 5.2*. Cabe resaltar que en las articulaciones existen dinámicas complejas de modelar, entre ellas la histéresis en el actuador mecánico, un pivote mecánico que genera un retardo en la salida del sistema. Un modelo matemático que involucre las irregularidades mecánicas debido al desgaste y desacople de los piñones, representaría de mejor forma una articulación del prototipo.

Articulación	Modelo
MCP índice	$G_1(s) = \frac{25{,}91}{s \cdot (s+57{,}97)}$
PIP índice	$G_2(s) = \frac{35{,}85}{s \cdot (s+122{,}4)}$
MCP medio	$G_3(s) = \frac{18{,}42}{s \cdot (s+78{,}29)}$
MCP medio	$G_4(s) = \frac{20{,}73}{s \cdot (s+85{,}59)}$
MCP pulgar	$G_5(s) = \frac{20{,}68}{s \cdot (s+93{,}06)}$
PIP pulgar	$G_6(s) = \frac{19{,}46}{s \cdot (s+75{,}9)}$

Tabla 5.2: Modelos para seis articulaciones.

Idealmente es deseado tener el *modelo dinámico directo* de la mano robótica, con el cual se simula el comportamiento dinámico del prototipo mecánico, como una unidad completamente acoplada. El desarrollo de este trabajo, se hace con los modelos presentados en la *Tabla 5.2*, con los cuales se presentan los resultados de simulación.

5.2. Generación de trayectorias

La capacidad de manipular y usar elementos por el prototipo mecánico es una tarea de gran importancia y objeto de estudio. La interacción del prototipo con el objeto implica un conocimiento a priori, que permitan realizar un agarre planificado, que se traduce básicamente en la estabilidad y naturalidad de ejecución de agarre. En [33] se hace el estudio detallado para la estimación de los ángulos finales (Q_f) para cada una de las articulaciones del prototipo mecánico, cuando el elemento de agarre es un cuerpo cilíndrico de radio r, observar *figura 5.4*. Según [2], a diferencia de los sistemas de control clásico, cuya consigna es un escalón, en robótica no puede utilizarte este tipo de referencia, debido que esto implicaría ir de una posición final (en radianes o grados) en un tiempo infinitesimal. El robot podría sufrir daños debido a las inercias de las masas de los eslabones que lo conforman. En tanto, lo correcto es seguir trayectorias que van

de una posición inicial Q_i hasta una posición final Q_f de forma suave como lo muestra la *figura 5.5*.

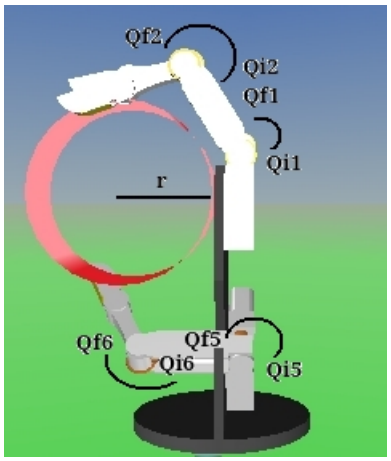


Figura 5.4: Q_f y Q_i para las articulaciones del prototipo virtual

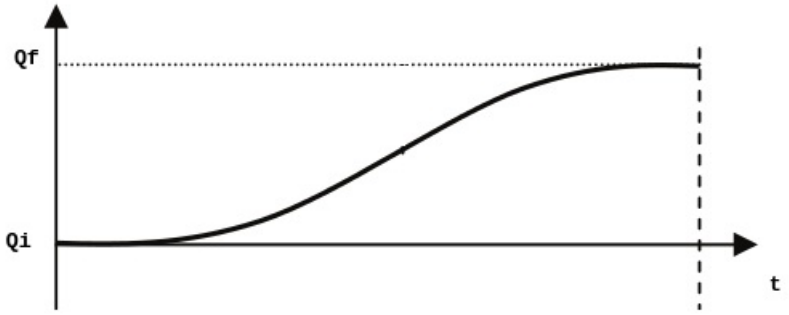


Figura 5.5: Consigna de quinto grado, modificada de [2].

El controlador en robótica construye la consigna de control en línea o en tiempo de ejecución. En [2] se explica la implementación de una consigna grado cinco, usando la *(Ec. 5.10)*:

$$q(t) = 10\left(\frac{t}{tf}\right)^3 - 15\left(\frac{t}{tf}\right)^4 + 6\left(\frac{t}{tf}\right)^5 \tag{5.10}$$

En la *figura 5.6*, el bloque resaltado con color azul corresponde a los ángulos finales (Qf_j con $j = 1, 2, ..., 6$), o valor final de la consigna de control en estado estacionario. El bloque resaltado con color rojo, es una *Embedded Matlab Function*, en donde se encuentra implementado el algoritmo matemático que construye las consignas articulares de grado cinco para las seis articulaciones, punto a punto en cada periodo de muestreo.

En la *figura 5.7* se observa una trayectoria para una articulación MCP, calculada en un *scrip* en Matlab® cuya posición final es de $\pi/6$ radianes, resaltando que idealmente la posición de reposo de una articulación es de 0^o. Para experimentación práctica las (Qi_j con $j = 1, 2, ..., 6$) serán pequeños ángulos existentes debido a los pequeños voltajes existentes en los ADC del microcontrolador cuando la articulación esta en 0^o.

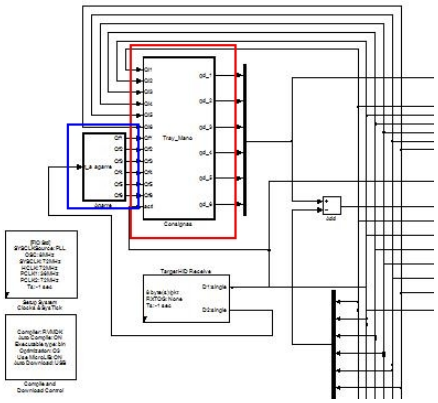


Figura 5.6: Bloque de construcción de trayectoria.

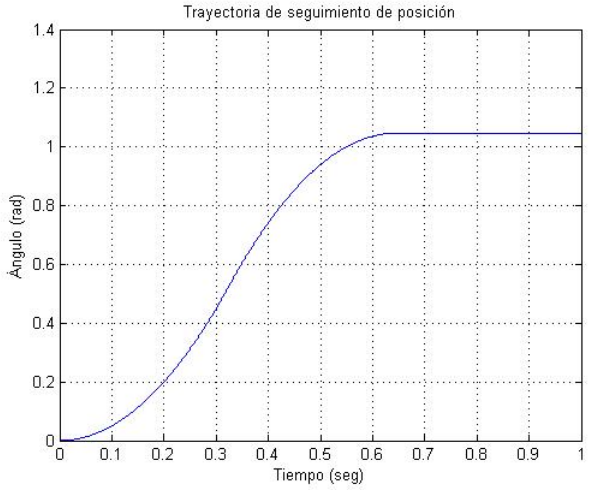


Figura 5.7: Trayectoria para una articulación con $Q_i = 0$

Para evitar errores iniciales de seguimiento excesivos se hace $Qi_j = Qi_e$ con $j = 1, 2, ..., 6$, donde Qi_e es la posición articular existente en cada una de las articulaciones del prototipo mecánico. En la *figura 5.8* se observa el conjunto de consignas articulares para un agarre cilíndrico cuyo cuerpo tiene radio 2,25cm creadas por el microcontrolador inmerso en la tarjeta *FiO Std*, enviadas por medio una comunicación USB HID.

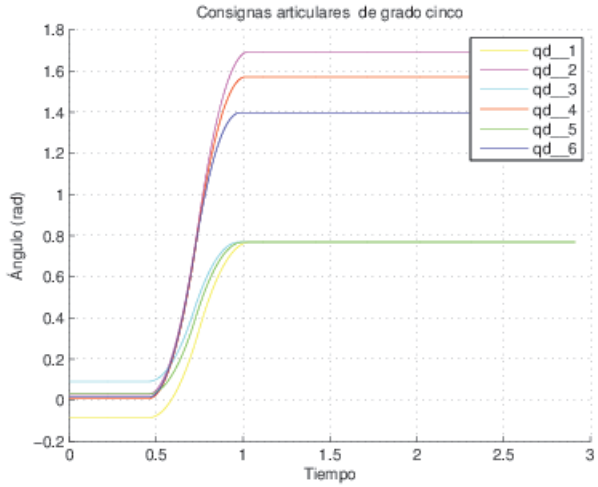


Figura 5.8: Consignas articulares construidas por el microcontrolador para cada articulación del prototipo mecánico.

5.3. Modelo virtual del prototipo de mano robótica

5.3.1. V-Realm Builder 2.0.

V-Realm Builder 2.0. [34],(*figura 5.9*), es un paquete de herramientas de Matlab® que permite visualizar simulaciones de sistemas dinámicos en una escena 3D de realidad virtual. Mediante una sencilla interfaz de funciones orientadas a objetos y bloques, este *toolbox* enlaza, respectivamente, Matlab® y Simulink® con las gráficas de realidad virtual. Utilizando señales de Simulink® es posible controlar parámetros como la posición, orientación y dimensión de los objetos definidos en el entorno 3D, generando así una animación de la simulación.

Una de las características más importantes de este programa es que al estar integrado con Matlab®, se puede trabajar paralelamente para la simulación, en función de una programación establecida.

Este *toolbox* se escogió debido a su gran soporte bibliográfico, tutoriales e información y principalmente porque permite incorporar de forma sencilla archivos creados en el software CAD SolidEdge® [35] al ser guardados en VRML (Virtual Reality Modeling Language), que es un formato de archivo para describir objetos y mundos virtuales interactuando en tres dimensiones.

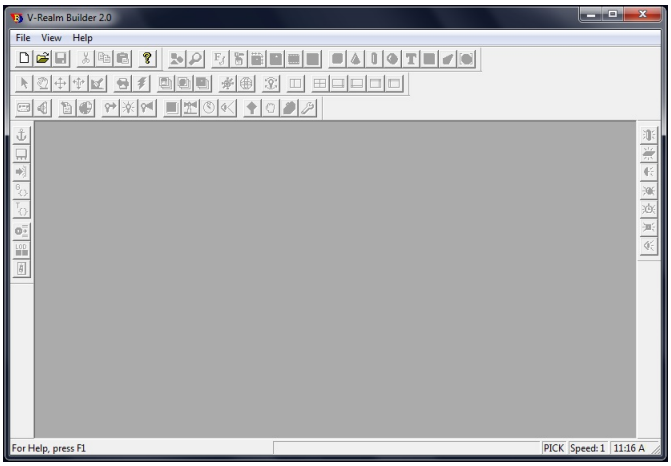
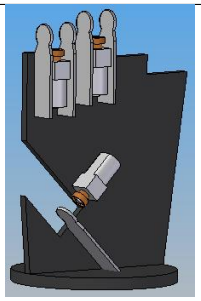
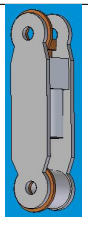


Figura 5.9: V-Realm Builder 2.0.

5.3.2. Desarrollo del modelo virtual

Para realizar la construcción del modelo virtual en V-Realm Builder 2.0, es necesario exportar los cuerpos construidos en SolidEdge®. Estas piezas inicialmente se encuentran en formato *.par, lo que requiere que sean guardadas como imágenes en formato *.wrl, para luego importar las en la herramienta de 3D. En la *Tabla 5.3*, se indican las piezas construidas en SolidEdge® que hacen parte del prototipo de mano robótica, definiendo las tres falanges correspondientes a cada uno de los dedos.

Pieza	**Descripción**	**Diseño**
Palma de la mano	Es la pieza base de construcción del entorno virtual y no posee movimiento. La comprenden las falanges metacarpianas de cada uno de los dedos y los motores para el movimiento rotacional de las falanges proximales.	
Falange Proximal	La conforma un motor que junto con el sistema de engranajes generan el movimiento de rotación de la falange medial.	

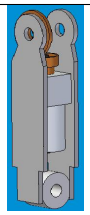
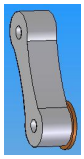
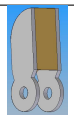
Falange Proximal Dedo Pulgar	Se conforma por un motor con el sistema de engranajes para el movimiento rotacional de la falange medial que se une al extremo superior con un rote de 45°.	
Falange Medial	Esta pieza en el extremo inferior tiene el sistema de engranajes para el movimiento de rotación generado junto con el motor situado en la falange proximal.	
Falange Distal	La conforman dos piezas laterales y una en el medio dándole el grosor del dedo.	

Tabla 5.3: Piezas del prototipo de mano robótica en SolidEdge®.

V-Realm Builder 2.0 trabaja con la estructura de árbol jerárquico "padre e hijos", lo que requiere un análisis para definir cuáles van a ser los padres y cuáles los hijos, con la limitante de que un padre puede tener más de un hijo, pero un hijo no puede tener más de un padre. La jerarquía del prototipo de mano robótica se estableció como muestra la *figura 5.10*.

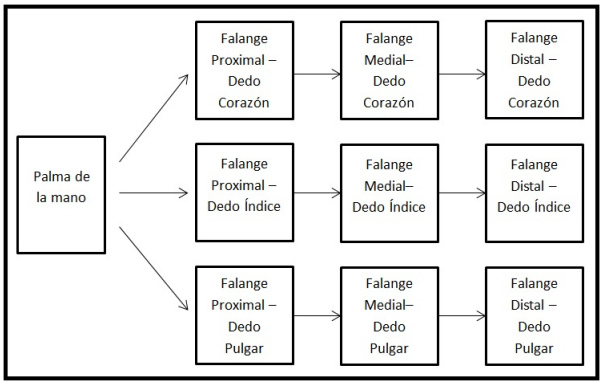


Figura 5.10: Estructura jerárquica padre e hijos

Como padre principal se definió la palma de la mano, a su vez a ésta se le asocian tres hijos, que son, las falanges proximales de cada uno de los dedos que conforman el prototipo de mano robótica. Luego a la falange proximal se le asocia como hijo la falange medial y finalmente a ésta se le asocia como hijo la falange distal; de igual forma para cada uno de los dedos.

Posteriormente se empiezan a enlazar cada una de las piezas que forman el prototipo de mano robótica siguiendo la estructura jerárquica definida anteriormente. Inicialmente se hizo el acoplamiento de la palma con las falanges proximales, que a su vez se le acoplan las falanges mediales y finalmente a éstas las falanges distales. El prototipo de mano robótica se puede observar en la *figura 5.11*.

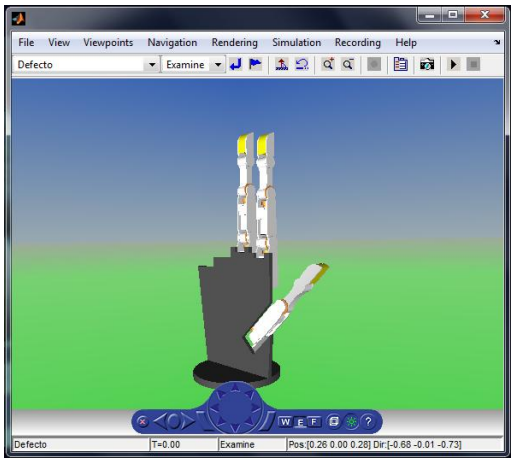


Figura 5.11: Prototipo de mano robótica en V-Realm Builder.

Los detalles de la construcción del modelo virtual se observan en el Anexo B adjunto a este documento.

5.3.3. Simulación de agarres en el modelo virtual

Para medir el desempeño del modelo virtual en Simulink® se implementó en un archivo *.mdl un esquema de control por retroalimentación (Feedback) *figura 5.12*, donde se establecen los ángulos de rotación para realizar los agarres de acuerdo a la *Tabla 5.4*, se construyen los controladores y el modelo de planta para las articulaciones, se adecúa el eje de giro y con el bloque VR Sink se despliega una ventana para observar los movimientos del prototipo de mano robótica en simulación.

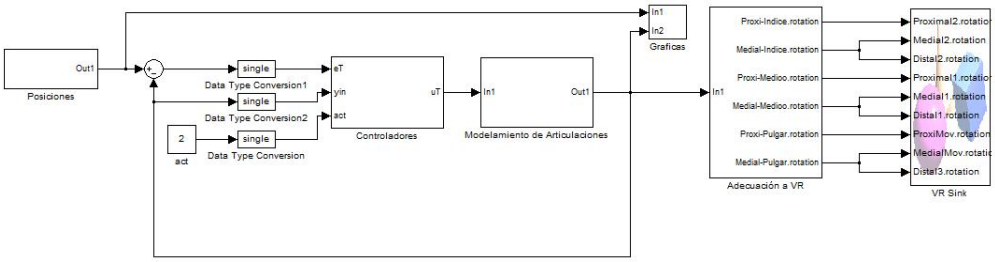


Figura 5.12: Bloques de control y simulación en Simulink®.

La siguiente *Tabla 5.4* consigna los ángulos (determinados en [1] Pág. 78.) para la ejecución de cada agarre, con excepción del agarre cilíndrico donde los ángulos finales surgen de acuerdo al radio del cilindro (ver *figura 5.28*) y el agarre de gancho donde a partir de la lectura tomada de los sensores en las articulaciones se determinaron sus ángulos. Cabe aclarar que los ángulos para ejecutar cualquier agarre no son únicos y los presentados a continuación fueron con fines experimentales.

Postura de agarre	Medio		Índice		Pulgar	
	Proximal	Medial	Proximal	Medial	Proximal	Medial
Reposo	0°	0°	0°	0°	0°	0°
Cilíndrico	60°	75°	60°	75°	50°	30°
Lateral	70°	35°	80°	35°	35°	30°
Precisión	45°	10°	70°	20°	45°	20°
Gancho	10°	80°	10°	80°	50°	30°

Tabla 5.4: Ángulos para diferentes posturas de agarre.

Con la anterior información se pudieron realizar los siguientes agarres:

- Agarre cilíndrico (*figura 5.13*): Se consigue rotando el pulgar totalmente en la posición de aducción o acercamiento palmar. Se utiliza para sujetar latas y objetos que requieran un agarre totalmente envolvente.

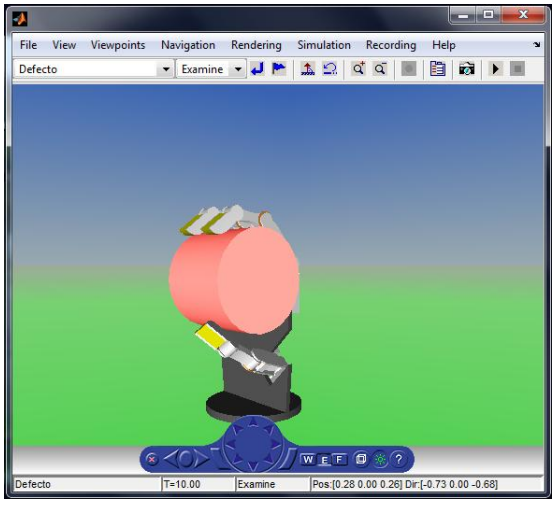


Figura 5.13: Agarre cilíndrico.

- Agarre lateral (*figura 5.14*): en el cual el pulgar cierra sobre el índice. Esta postura es utilizada para mantener objetos delgados como una tarjeta o un CD.

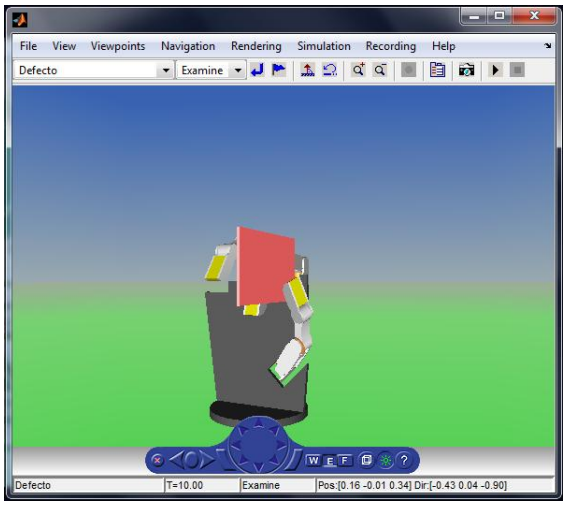


Figura 5.14: Agarre lateral.

- Pinza de precisión (*figura 5.15*): postura que se logra cuando el dedo índice y el dedo pulgar se juntan (o el dedo índice, el dedo medio y el pulgar), para agarrar objetos y mantenerlos cuando se necesita precisión.

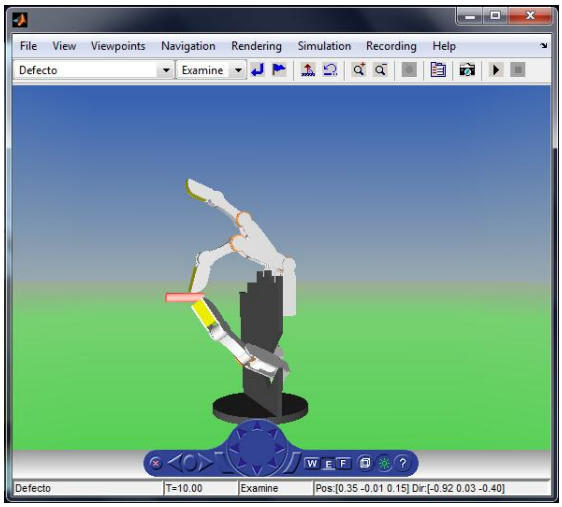


Figura 5.15: Pinza de precisión.

- Agarre de gancho (*figura 5.16*): donde todos los dedos y el pulgar cierran de manera conjunta para crear un puño. Este agarre es utilizado para sujetar una bolsa o sostener un maletín.

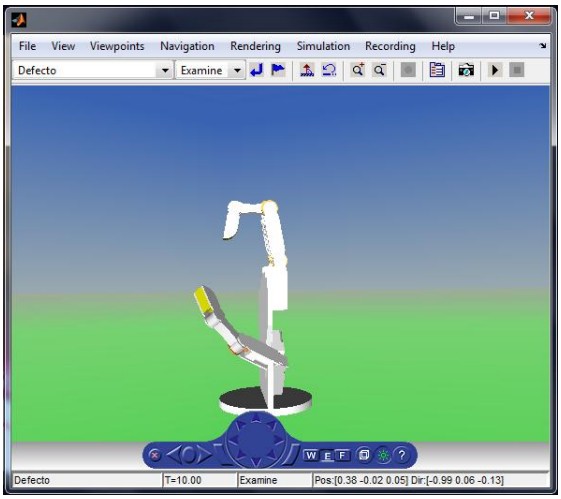


Figura 5.16: Agarre de gancho.

De esta manera se consiguió validar el diseño virtual del prototipo de mano robótica, ya que ésta ejecuta de forma correcta cada agarre programado.

5.4. Implementación de un controlador PID de dos grados de libertad

Una de las técnicas de control clásicas es el controlador proporcional, integral derivativo *PID*, caracterizado esencialmente por ser robusto y de fácil implementación en sistemas embebidos. Su forma académica se representa mediante la *(Ec. 5.11)*.

$$u(t) = K \cdot \left(e(t) + \frac{1}{T_i} \cdot \int_0^t e(\tau)\, dt + T_d \cdot \frac{de(t)}{dt} \right) \tag{5.11}$$

Donde $u(t)$ es el esfuerzo de control y $e(t) = r(t) - y(t)$ el error de seguimiento de la salida con respecto a la referencia o consigna de control. El controlador descrito por la *(Ec. 5.11)*, está limitado a resultados de simulación, dado que no existe un derivador puro, por lo que es necesario limitarlo, tal como lo son los sistemas dinámicos reales. La acción derivativa del controlador también tiene un efecto no deseado y es el amplificar señales de ruido. Un esquema que soluciona las dos desventajas anteriormente mencionadas del controlador descrito por la *(Ec. 5.11)*, es un controlador que tiene acción derivativa limitada y que se anticipa a la señal de salida y no del error existente. Este controlador se conoce como *Controlador PID de dos grados de libertad* y es descrito mediante la *(Ec. 5.12)*.

$$u(s) = k_p \cdot \left(e(s) + \frac{k_i}{s} \cdot e(s) - \frac{k_d \cdot s}{T_f \cdot s + 1} \cdot y(s) \right) \tag{5.12}$$

donde $K = k_p, k_i = \frac{K}{T_i}, k_d = K \cdot T_d$. Aunque este controlador es implementable, persiste un efecto no deseado, generado por la acción integral denominada *windup* [30]. La acumulación de error en la acción integral, hace que el esfuerzo de control aumente de forma excesiva. En la *figura 5.17*, se observa que para disminuir $u(t)$ es necesario esperar que se genere un esfuerzo de control de igual magnitud pero de signo inverso, a fin que se presente una cancelación en la salida del controlador $u(t)$ relacionada con la acción integral. Si tomamos como ejemplo una válvula, observamos que se encuentra limitada por topes físicos, que le permite interrumpir completamente el flujo de energía (caudal de agua, gas, entre otros) o dejar pasar el 100 % del máximo valor de la variable manipulada. Un controlador PID, sin acción *anti-windup* puede disminuir el tiempo de vida del actuador. En algunos esquemas de control, se verifica el estado del actuador, para el ejemplo mencionado la posición angular de la válvula. Sin embargo, en algunos procesos, el actuador es sometido a condiciones de altas temperaturas o presiones, limitando el uso de sensores. También es posible encontrar modelos del actuador y mediante estos, estimar el estado del elemento final.

Un puente H, puede ser representado mediante una acción de saturación. El esquema del controlador PID de dos grados de libertad con acción *anti-windup* es mostrado, en donde la salida del actuador es estimada mediante un modelo matemático. *figura 5.18*.

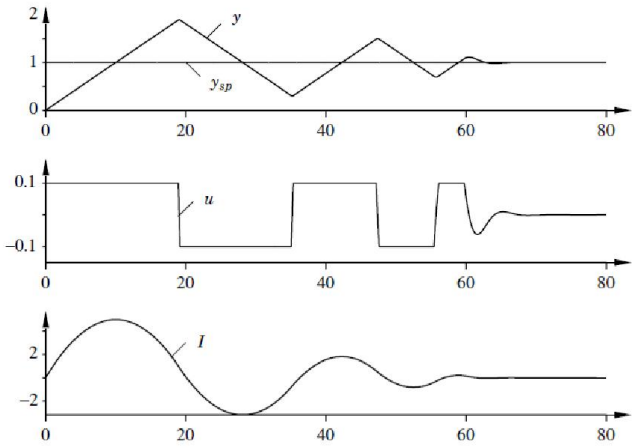


Figura 5.17: Efecto *windup* generado por la acción integral [30].

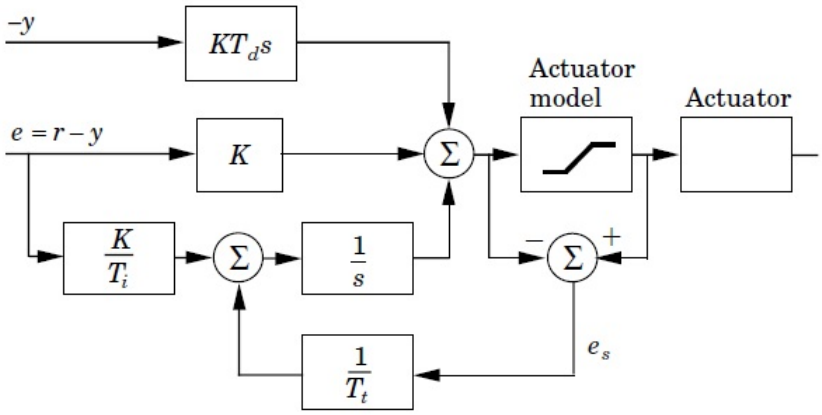


Figura 5.18: Esquema de control PID con *anti-windup* de dos grados de libertad [30].

5.4.1. Implementación computacional

Los controladores se embeben en sistemas electrónicos que entienden instrucciones, que toman datos del exterior y toman decisiones por medio de sus periféricos. El controlador expresado mediante *(Ec. 5.12)* debe ser interpretado para ser ejecutado en un sistema electrónico, como un microcontrolador, DSP, entre otros. La secuencia ideal de operación de un controlador en un sistema computacional es la siguiente:

1. Esperar un ciclo de interrupción.
2. Leer un canal analógico.
3. Calcular la señal de control.
4. Actuar sobre el exterior por medio de uno de sus periféricos.

La discretización aproximada del controlador se presenta a continuación.

5.4.1.1. Acción proporcional

La acción proporcional puede estimarse mediante la *(Ec.5.13)*

$$P = K \cdot \big(b \cdot y_s p - y\big) \tag{5.13}$$

Cambiando las variables de tiempo continuo a muestras, se tiene:

$$P(t_k) = K \cdot \big(b \cdot y_s p(t_k) - y(t_k)\big) \tag{5.14}$$

Donde t_k representa la muestra actual.

5.4.1.2. Acción integral

La acción integral puede estimarse mediante la *(Ec. 5.15)*:

$$I(t) = \frac{K}{T_i} \cdot \int_0^t e(t)\, dt \tag{5.15}$$

Derivando en ambos lados de la *(Ec. 5.17)*:

$$\frac{dI}{dt} = \frac{K}{T_i} \cdot e(t) \tag{5.16}$$

La cual puede aproximarse :

$$\frac{I(t_{k+1} - I(t_k))}{h} = \frac{K}{T_i} \cdot e(t_k) \tag{5.17}$$

resolviendo para $I(t_k + 1)$, se tiene:

$$I(t_{k+1}) = I(t_k) + \frac{K \cdot h}{T_i} \cdot e(t_k) \tag{5.18}$$

5.4.1.3. Acción derivativa

La acción derivativa puede estimarse mediante la *(Ec. 5.19)*:

$$D(s) = -\frac{s \cdot K \cdot T_d}{1 + s \cdot T_d/N} \cdot Y(s) \tag{5.19}$$

La acción derivativa es limitada por un filtro de primer orden con constante de tiempo T_d/N. Este filtro elimina las señales de alta frecuencia. Generalmente N toma valores entre 8 a 20.

$$\frac{T_d}{N} \cdot \frac{dD}{dt} + D = -K \cdot T_d \frac{dy}{dt} \tag{5.20}$$

De igual manera similar como en la acción integral, aproximando los valores continuos en valores de muestras.

$$\frac{T_d}{N} \cdot \frac{D(t_k) - D(t_{k-1})}{h} + D(t_k) = -K \cdot T_d \frac{y(t_k) - y(t_{k-1})}{h} \tag{5.21}$$

Resolviendo para $D_(t_k)$:

$$D(t_k) = \frac{T_d}{T_d + N \cdot h} \cdot D(t_{k-1}) - KT_d \frac{y(t_k) - y(t_{k-1})}{h} \tag{5.22}$$

Finalmente:

$$p(t_k) = k \cdot (b \cdot r(t_k) - y(t_k)) \tag{5.23}$$

$$e(t_k) = r(t_k) - y(t_k) \tag{5.24}$$

$$d(t_k) = \frac{T_d}{T_d + N \cdot h} \cdot \left(d(t_{k-1}) - K \cdot N \cdot (y(t_k) - y(t_{k-1})) \right) \tag{5.25}$$

$$u(t_k) = p(t_k) + i(t_k) + d(t_k); \tag{5.26}$$

$$i(t_{k+1}) = i(t_k) + \frac{k \cdot h}{T_i} \cdot e(t_k); \tag{5.27}$$

5.4.2. Desempeño de la ley de control PID en las articulaciones

La ley de control tratada anteriormente, fue probada en los agarres primitivos: cilíndrico, lateral, precisión y gancho.

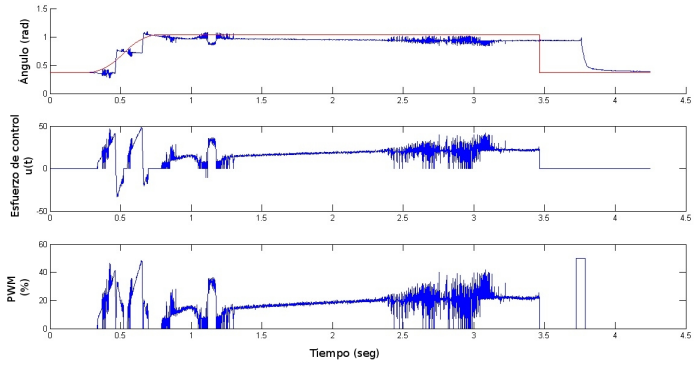


Figura 5.19: Seguimiento de una trayectoria articular en una articulación MCP dedo índice.

En la *figura 5.19* se observa el seguimiento de una trayectoria cuyo ángulo final es de $\pi/3$ radianes ó 60^o, los datos rojos corresponden a la trayectoria a seguir y los datos azules corresponden a la posición angular de la articulación MCP. Para $t \lessapprox 0{,}4$ se observa que la referencia se ajusta a los datos provenientes del sensor, a fin de no ingresar errores iniciales grandes a la ley de control. La gráfica intermedia corresponde al esfuerzo de control asociado a la articulación, la gráfica inferior corresponde al *PWM* aplicado al puente H. Para $0{,}4 \lessapprox t \lessapprox 1$ la trayectoria se construye hasta lograr establecerse en el ángulo final deseado. Se observa que la salida no sigue fielmente la referencia. En

esencia son dos los objetivos de control en cualquier sistema de control automático: seguir la consigna y hacerlo con el menor error posible. Evidentemente el seguimiento no es el deseado, pero se debe esencialmente a dos factores: los actuadores del prototipo mecánico no están dotados de mecanismos de mediana precisión, además de la existencia de un juego mecánico presente en las articulaciones del prototipo, en algunas con un efecto mayor que otras. Observe que para el rango objeto de estudio, cuando la salida es mayor que la referencia, el esfuerzo de control es negativo, sin embargo el PWM aplicado siempre será positivo, dado que el signo del esfuerzo determina la inversión de giro al motor DC. Para $1{,}3 \lessapprox t \lessapprox 2{,}5$ se observa que la posición angular sigue la trayectoria en su ángulo final y lo hace con un $3^o \lessapprox error \lessapprox 8^o$ aproximadamente. El esfuerzo de control aumenta debido al error presente, sin embargo aplicando aproximadamente el 30 % del PWM, no hay un efecto inmediato en la señal de salida del sistema. Esto puede entenderse como una zona muerta en el actuador. Para $2{,}5 \lessapprox t \lessapprox 3{,}1$ se presenta una señal de ruido en el sensor, sin embargo el mayor efecto de este ruido en la señal de esfuerzo de control cae sobre la zona muerta del actuador, por lo que el sistema no se desestabiliza. Finalmente en $t \approx 3{,}5$ se desactiva la ley de control, $u(t) = 0$ la señal del sensor se sostiene debido a la presencia del integrador en la dinámica de la articulación, para $t \approx 3{,}8$ la fuente que alimenta las tarjeta de acondicionamiento de señales y de potencia es apagada y para $t \approx 4{,}25$ aproximadamente es terminada la adquisición de datos.

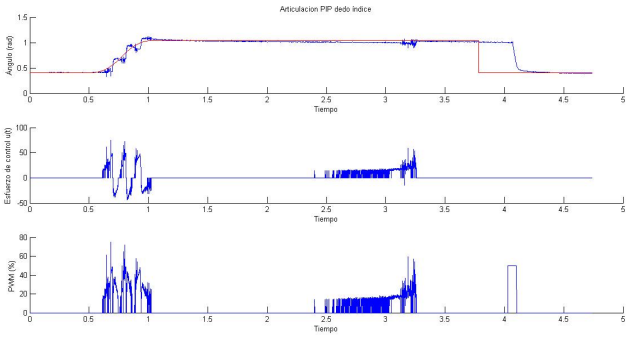


Figura 5.20: Seguimiento de una trayectoria articular con aceptación de error de 2^o.

En la *figura 5.20* se observa la respuesta de la posición articular para una articulación MCP dedo índice, para un agarre cilíndrico, agregando una tolerancia de error de 2^o por medio de software al controlador PID propuesto. Como puede observarse, hay un mejor seguimiento de la trayectoria, sin embargo no es lo suficientemente fiel a lo deseado. Finalmente el error de estado estable es bastante aceptable.

Se agregaron estos dos efectos evidenciados en el prototipo mecánico, a los modelos matemáticos obtenidos: el juego mecánico presente en la articulación y la zona muerta en el puente H, tal como se muestra en la *figura 5.21*.

Figura 5.21: Adición efectos de zona muerta en el actuador y juego mecánico en el modelo matemático.

Como se muestra en la *figura 5.22* la señal de salida del modelo presenta una respuesta con amplitud sostenida, que no obedece al comportamiento de la dinámica del sistema, sino a los dos efectos anteriormente mencionados.

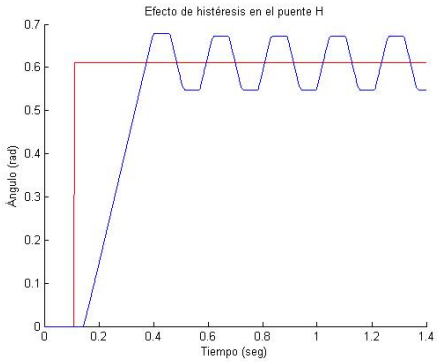


Figura 5.22: Generación de respuesta oscilatoria para una entrada tipo escalón.

En la *figura 5.23* se observa la oscilación de la articulación MCP. Estos datos fueron obtenidos cuando se realizaron pruebas de sintonización experimental. Sin embargo no es adecuado llevar todas las articulaciones a la respuesta con amplitud sostenida, dado que como anteriormente se mencionó se pueden generar daños en el actuador.

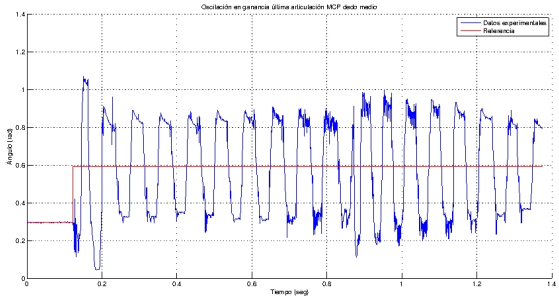


Figura 5.23: Salida oscilatoria de una articulación MCP dedo medio.

5.4.3. Sintonización experimental del controlador PID

La sintonización del controlador PID, se realizó en cada articulación por separado. Primeramente se aumenta la ganancia proporcional, logrando hacer movimientos articulares para errores relativamente grandes. Una vez la señal logra seguir la consigna, se empieza a calibrar la ganancia integral a fin de disminuir el error. Finalmente se aumenta la ganancia derivativa hasta obtener mejores respuestas.

Las constantes encontradas para las diferentes articulaciones para este agarre son mostradas en la *Tabla 5.5*.

Articulación	K_p	K_i	k_d
MCP índice	180	80	2
PIP índice	180	30	1.7
MCP medio	180	80	0.9
PIP medio	180	30	2.6
MCP pulgar	180	80	2.1
PIP pulgar	240	30	1

Tabla 5.5: Constantes del controlador PID.

Para obtener los datos de simulación, se usaron las trayectorias generadas por el microcontrolador en línea y enviadas por comunicación USB HID. El esquema de bloques usado es el mostrado en la *figura 5.24*

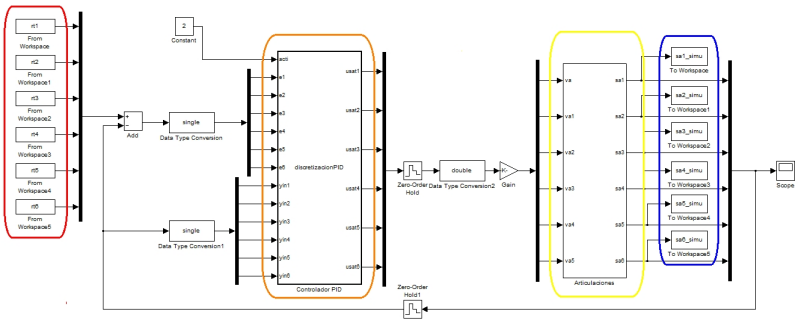


Figura 5.24: Esquema para adquisición de datos simulados control PID.

En el rectángulo rojo se observan los *from workspace* que contiene las estructuras con tiempo, donde están almacenadas las trayectorias aplicadas en línea por el

microcontrolador al prototipo mecánico (ver la *figura 5.8*). En el rectángulo naranja se resalta el controlador PID implementado en una *Embedded Matlab Function*. En el rectángulo amarillo se encuentra un *subsystem* que contiene los modelos descritos en la *Tabla 5.2* para cada una de las articulaciones del prototipo mecánico. En el rectángulo azul se observa los *to workspace* que contiene las estructuras con tiempo donde se almacena la salidas individuales de cada articulación. Para un cuerpo cilíndrico de 2,25cm se determinó los (Qf_j con $j = 1, 2, ..., 6$) y se construyeron las trayectorias mostradas en la *figura 5.8*.

5.4.3.1. Agarre cilíndrico

Los resultados para el agarre cilíndrico son mostrados en las *figuras 5.25, 5.26 y 5.27* para los dedos índice, medio y pulgar respectivamente. Los datos de color rojo (REF), corresponden a la trayectoria de seguimiento o referencia, los datos de color azul (DE), corresponden a los datos experimentales, adquiridos mediante una comunicación USB HID. Los datos de color negro (SM), corresponden a la salida los modelos de cada articulación.

En la *figura 5.25* se observa un seguimiento de la articulación MCP del dedo índice bastante buena, sin embargo con un error considerable. Existe una fuerte relación entre los datos experimentales y los datos de simulación provenientes de los modelos para esta articulación en el estado transitorio.

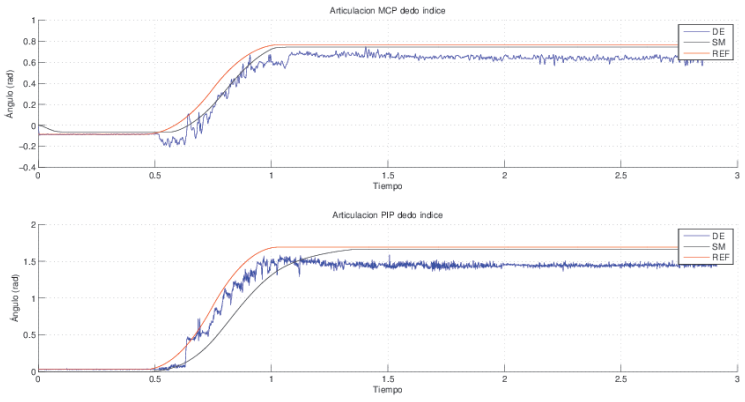


Figura 5.25: Comparación de datos del modelo y datos experimentales dedo índice control PID.

En la *figura 5.26* se observa un seguimiento de la articulación MCP del dedo medio bastante buena, sin embargo al igual que la articulación MCP del dedo índice, existe un

error considerable. También para esta articulación existe una fuerte relación entre los datos experimentales y los datos de simulación provenientes de los modelos en el estado transitorio.

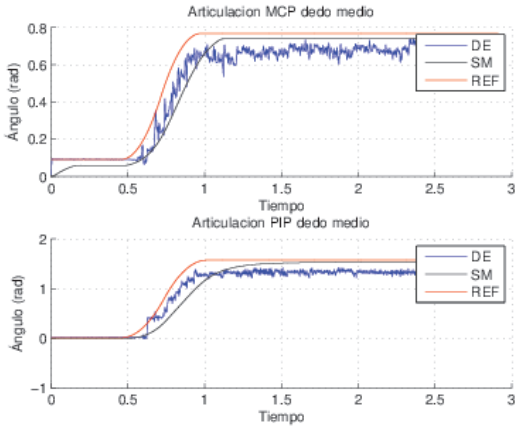


Figura 5.26: Comparación de datos del modelo y datos experimentales dedo medio control PID.

En la *figura 5.27* se observa mejor error de estado estacionario para las articulaciones del dedo pulgar comparados con los dos dedos restantes. Sin embargo el transitorio es mejor para los dedos índice y medio.

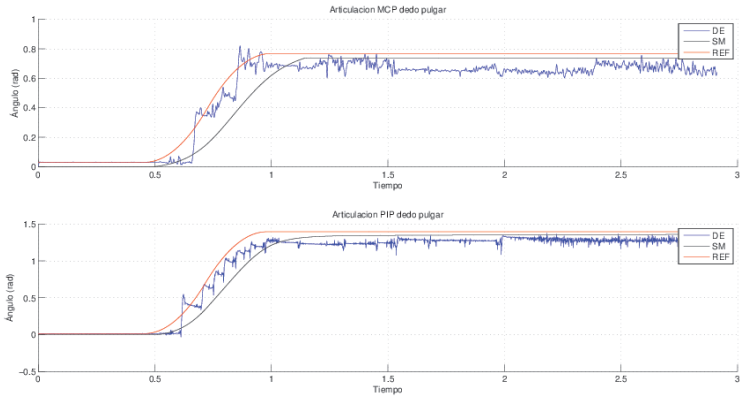


Figura 5.27: Comparación de datos del modelo y datos experimentales dedo pulgar control PID.

En la *Tabla 5.6* se presenta el error en estado estacionario para cada articulación en los datos experimentales y datos del modelo.

Articulación	Error en los DE *(rad)*	Error en los datos SM *(rad)*
MCP índice	0.1238	0.0191
PIP índice	0.2479	0.0311
MCP medio	0.0870	0.0279
PIP medio	0.2399	0.0348
MCP pulgar	0.1065	0.0289
PIP pulgar	0.1167	0.0353

Tabla 5.6: Errores articulares de estado estacionario: datos experimentales y salida del modelo con ley de control PID.

En la *figura 5.28* se puede observar la secuencia en la ejecución de un agarre cilíndrico en el prototipo mecánico.

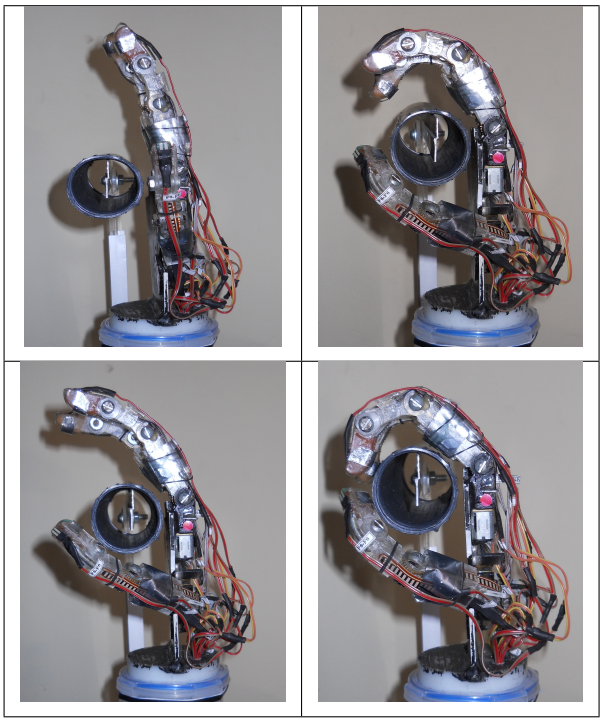


Figura 5.28: Ejecución de un agarre cilíndrico.

A partir de las *figuras 5.25, 5.26 y 5.27* y la *Tabla 5.6*, se concluye que un controlador PID difícilmente puede cumplir de manera fiel los dos objetivos de control (seguir trayectoria y error mínimo), en presencia de comportamientos no lineales en la dinámica real del prototipo mecánico. La recomendación es adecuar el prototipo mecánico,

eliminando en lo mayor posible las dinámicas no deseadas, entre estos: juegos mecánicos, histéresis en el actuador, fricciones secas severas, entre otras. Finalmente se pueden implementar leyes de control, en las cuales los conocimientos previos y prácticos del prototipo mecánico puedan ser inmersos en la lógica de control, como en un controlador fuzzy estudiado en el Anexo E.

5.4.3.2. Agarre de gancho

En las *figuras 5.29, 5.30 y 5.31*, se observan los datos simulados y experimentales para un agarre primitivo tipo gancho.

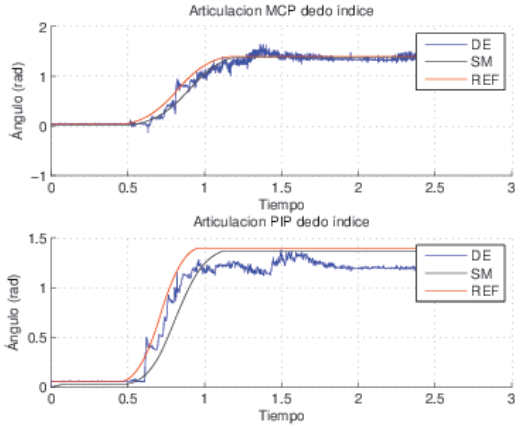


Figura 5.29: Control PID, agarre de gancho: dedo índice.

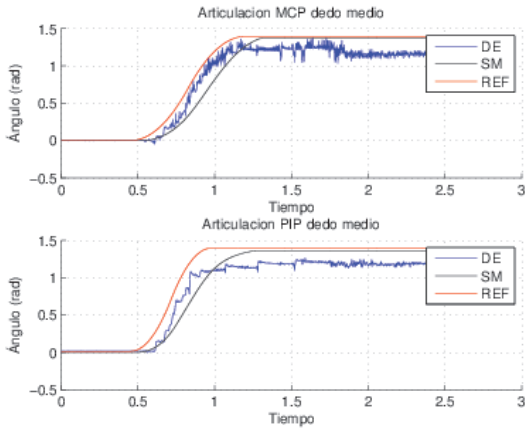


Figura 5.30: Control PID, agarre de gancho: dedo medio.

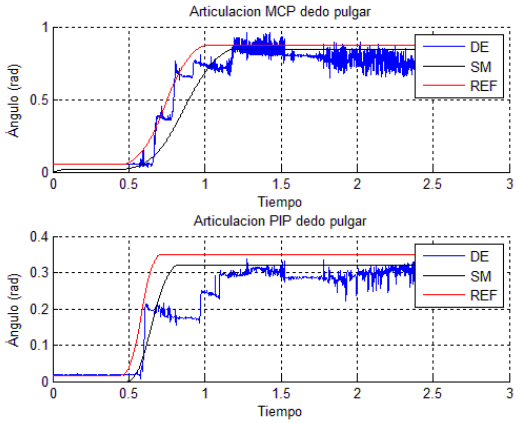


Figura 5.31: Control PID, agarre de gancho: dedo pulgar.

Para este agarre se hace un mejor seguimiento en las articulaciones MCP del prototipo mecánico. La salida de los modelos tiene un error de estado estacionario bastante aceptable, a diferencia de la articulación PIP del dedo pulgar, en donde los datos experimentales no logran hacer un seguimiento fiel a la referencia. Esto se debe principalmente a los juego mecánicos existente y la diferencia de enganche entre los engranajes en las diferentes tomas de datos.

5.4.3.3. Agarre lateral

En las *figuras 5.32, 5.33 y 5.34* se observan los datos simulados y experimentales para un agarre primitivo tipo lateral.

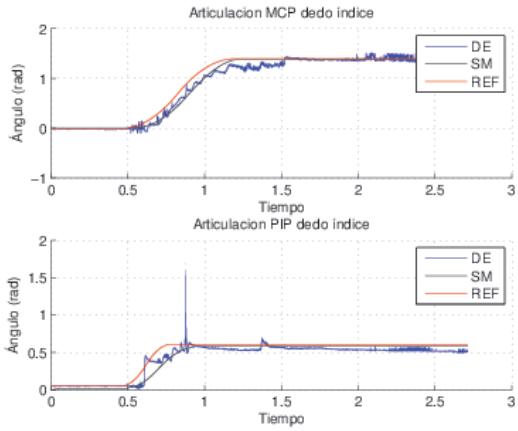


Figura 5.32: Control PID, agarre lateral: dedo índice.

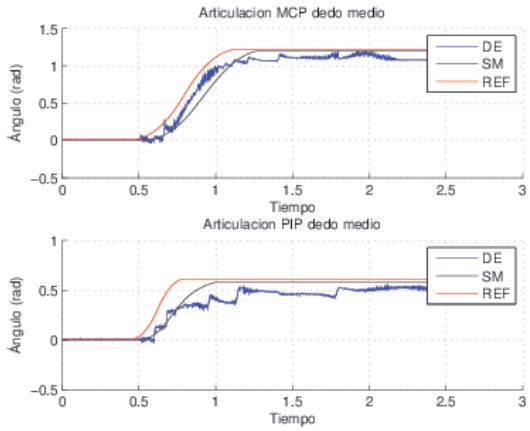


Figura 5.33: Control PID, agarre lateral: dedo medio.

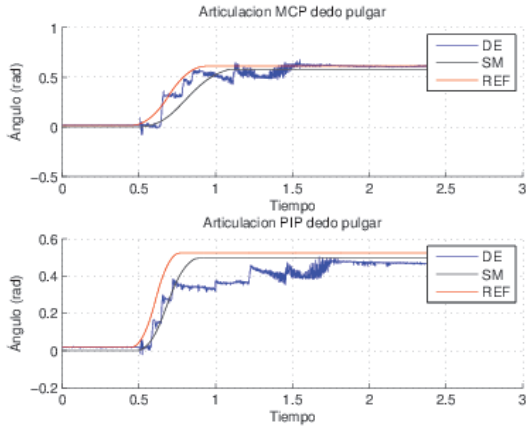


Figura 5.34: Control PID, agarre lateral: dedo pulgar.

En este agarre se puede observar que para los tres dedos, el error final o de estado estacionario es relativamente aceptable, sin embargo solo para la articulación MCP del dedo índice hay un seguimiento relativamente fiel. Un control PID, logra hacer un seguimiento a consignas articulares básicamente por la acción del sujetador. Sin embargo los datos experimentales reflejan que para sistemas con no linealidades, esta ley de control queda limitada y no logra los dos objetivos de control de manera eficiente: Seguir una consigna y hacerlo con un error mínimo.

5.4.3.4. Agarre de precisión

En las *figuras 5.35, 5.36 y 5.37* se observan los datos simulados y experimentales para un agarre primitivo tipo lateral.

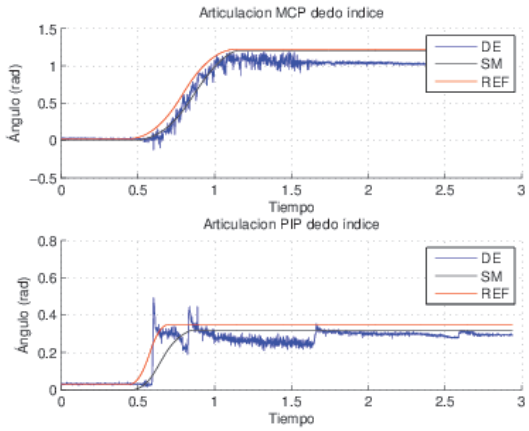


Figura 5.35: Control PID, agarre de precisión: dedo índice.

Al igual que en el agarre de gancho y lateral, las articulaciones MCP de los dedos índice y medio ejecutan un mejor seguimiento de la trayectoria. Las articulaciones PIP de los tres no logran seguir la consigna, sin embargo el movimiento ejecutado es escalonado.

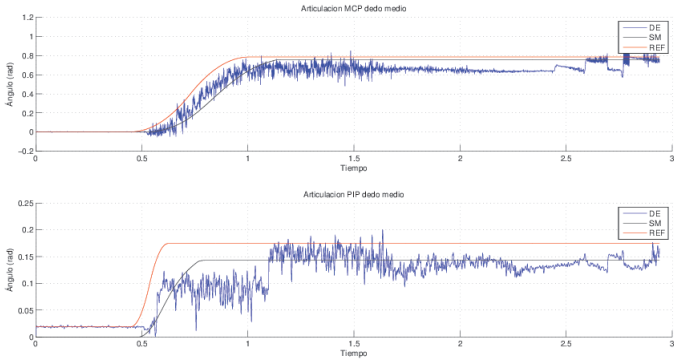


Figura 5.36: Control PID, agarre de precisión: dedo medio.

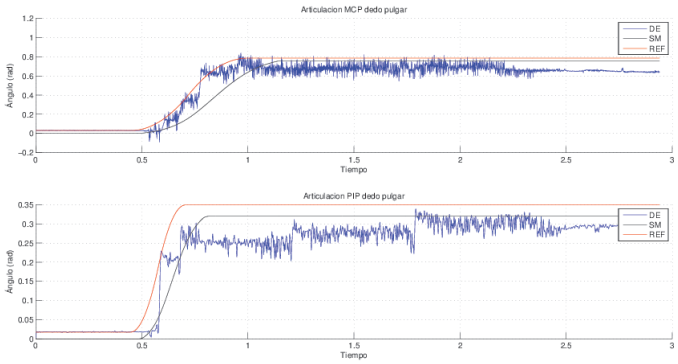


Figura 5.37: Control PID, agarre de precisión: dedo pulgar.

El controlador PID para la articulación PIP del dedo pulgar presenta mayor dificultad de calibración para todos los agarres primitivos. Esto debe principalmente la fricción seca existente, dado que es necesario aplicar un par considerable para lograr mover esta articulación.

5.5. Implementación de un compensador por adelanto de fase.

Para el diseño del controlador se deben tener en cuenta las especificaciones que se desean satisfacer, lo cual implica en muchos casos la repetición continua del método de ensayo y error.

Hacer el ajuste de la ganancia del sistema es el primer paso, sin embargo no en todos los casos esto es suficiente para alcanzar las especificaciones dadas. Aumentar la ganancia mejora el funcionamiento en estado estacionario, pero no así la estabilidad del sistema. En este caso es necesario rediseñar el sistema para alterar su funcionamiento de forma global, de manera que el sistema se comporte en la forma deseada. Este rediseño se denomina compensación y al dispositivo que se inserta se le denomina compensador. El compensador modifica el desempeño con déficit del sistema original [37].

5.5.1. Procedimiento de diseño para compensar en adelanto por el método de respuesta en frecuencia.

Requiere modificar la forma de la curva de respuesta en frecuencia dando suficiente adelanto de fase como para contrarrestar el atraso de fase excesivo [38].

- Suponga el siguiente compensador en adelanto:

$$G_c = K_c \frac{s+z}{s+p}, con \mid z \mid > \mid p \mid \tag{5.28}$$

- Remplazando $s = jw$,

$$\begin{aligned} G_c(jw) = K_c \frac{jw+z}{jw+p} = K_c \frac{z}{p} \frac{\frac{jw}{z}+1}{\frac{jw}{p}+1} = K \frac{1+jw\alpha\tau}{1+jw\tau} \\ con, \tau = \frac{1}{p}, \alpha = \frac{p}{z}, K = \frac{K_c}{\alpha} \end{aligned} \tag{5.29}$$

- El valor máximo de adelanto de fase se presenta en W_m.

- Donde la W_m media geométrica de p y z.

$$W_m = \sqrt{zp} = \frac{1}{\tau\sqrt{\alpha}} \tag{5.30}$$

- Ángulo de adelanto de fase.

$$\varnothing(w) = \tan^{-1}\alpha w\tau - \tan^{-1} w\tau \tag{5.31}$$

$$\tan\varnothing(w) = \frac{\alpha w\tau - w\tau}{1 + (w\tau)^2\alpha}, W_m = \frac{1}{\tau\sqrt{\alpha}} \tag{5.32}$$

$$\tan\varnothing(w) = \frac{\frac{\alpha}{\sqrt{\alpha}} - \frac{1}{\sqrt{\alpha}}}{1+1} = \frac{\alpha - 1}{2\sqrt{\alpha}} \tag{5.33}$$

$$\sin\varnothing_m = \frac{\alpha - 1}{\alpha + 1} \tag{5.34}$$

5.5.2. Diseño del compensador para la articulación MCP del dedo índice.

De acuerdo al modelo de la articulación hallada en la *Tabla 5.2*, se requiere diseñar un compensador con $K_v = 32$ y MF $= 80$ para:

$$G = \frac{25{,}91}{s(s + 57{,}97)} \tag{5.35}$$

- Determinar K para satisfacer las condiciones de error de estado estacionario:

$$K_v = \lim_{s\to 0} s\frac{K_c(s+z)}{(s+p)}\frac{25{,}91}{s(s+57{,}97)} = 32 \tag{5.36}$$

$$K_v = K_c\frac{z}{p}\frac{25{,}91}{57{,}97} = 32 \tag{5.37}$$

$$K = \frac{K_v}{K_c} = 71{,}59 \tag{5.38}$$

- Con $K = 71.59$ se traza el diagrama de Bode del sistema.

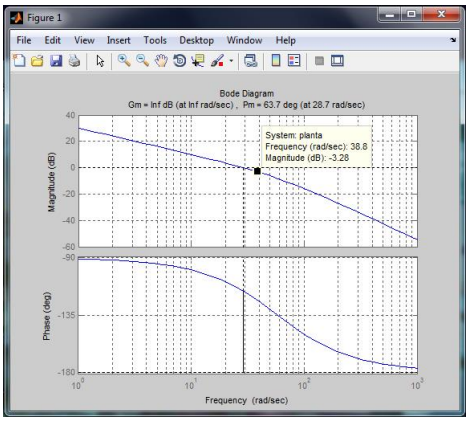


Figura 5.38: Diagrama de Bode con K = 71.59.

- Cálculo del ángulo de adelanto de fase.

$$\varnothing_m = MF_{deseado} - MF_{actual} + MF_{seguridad} = 80° - 63{,}7° + 5° = 21{,}3° \tag{5.39}$$

- Cálculo de alfa.

$$\sin \varnothing_m = \frac{\alpha - 1}{\alpha + 1}, \alpha = \frac{1 + \sin \varnothing_m}{1 - \sin \varnothing_m} = 2{,}14 \tag{5.40}$$

- Para W_m el compensador introduce una magnitud de:

$$10 \log \alpha = 3{,}3dB \tag{5.41}$$

En la gráfica de Bode se calcula la frecuencia a la cual la magnitud es - 3.3 dB. Esta es la nueva frecuencia de cruce por cero del sistema compensado. En este caso $W_m = 38{,}8rad/sg$.

- Ubicación de polo y del cero.

$$p = W_n \sqrt{\alpha} = 56{,}759 \tag{5.42}$$

$$z = \frac{p}{\alpha} = 26{,}523 \tag{5.43}$$

- Cálculo K_c.

$$K_c \frac{z}{p} = 71{,}59 \tag{5.44}$$

$$K_c = \frac{29{,}08p}{z} = 153{,}202 \tag{5.45}$$

- Compensador por adelanto de fase diseñado.

$$G_c = 153{,}202\frac{s + 26{,}523}{s + 56{,}759} \tag{5.46}$$

- Diagrama de Bode del sistema compensado.

$$G_1 = 153{,}202\frac{s + 26{,}523}{s + 56{,}759}\frac{25{,}91}{s(s + 57{,}97)} \tag{5.47}$$

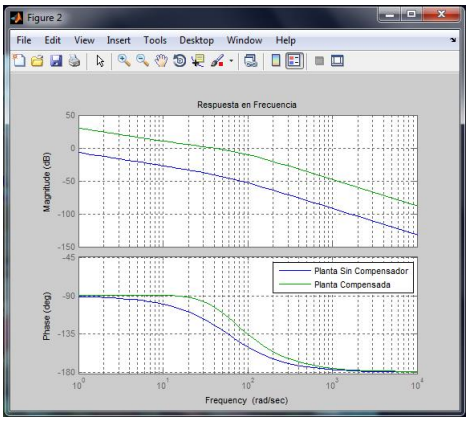


Figura 5.39: Diagrama de Bode del sistema compensado.

- Respuesta en el tiempo sistema compensado.

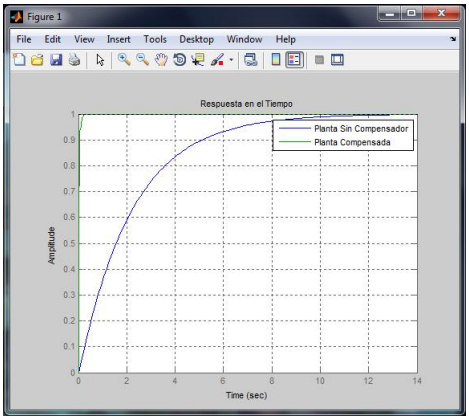


Figura 5.40: Respuesta en el tiempo del sistema compensado.

5.5.3. Diseño del compensador para todas las articulaciones.

De acuerdo con el modelo de planta que fue identificado para cada articulación (*Tabla 5.2*) y siguiendo el mismo procedimiento descrito en el apartado anterior, los compensadores obtenidos se resumen en la siguiente *Tabla 5.7*.

Compensador por adelanto de fase		Índice		Medio		Pulgar	
		MCP	PIP	MCP	PIP	MCP	PIP
Parámetros de diseño	Kv	32	32	25	36	25	30
	MDF	80	80	80	80	80	80
Compensador	Kc	153.202	151.506	162.546	268.334	157.971	202.361
	Zc	26.523	30.91	23.5943	32.4549	23.9464	27.301
	Pc	56.759	42.8692	36.0622	58.5162	33.6179	47.2108

Tabla 5.7: Compensadores hallados para las articulaciones.

5.5.4. Desempeño del compensador por adelanto de fase.

Diseñados los compensadores por adelanto para cada articulación, se procede a la adquisición de datos de simulación, al igual que lo mostrado en el controlador PID se usó las trayectorias generadas por el microcontrolador. El esquema de bloques usado es en esencia el mismo mostrado en la *figura 5.24*, en este caso solo se ha variado el controlador como se ve a continuación en la *figura 5.41*.

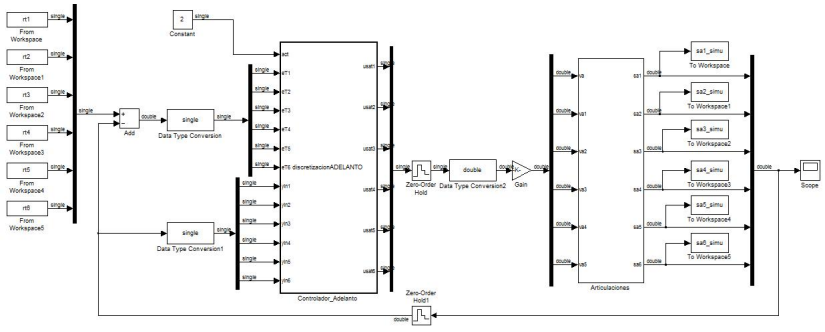


Figura 5.41: Esquema para adquisición de datos simulados para el compensador por adelanto.

5.5.4.1. Agarre cilíndrico

Los resultados obtenidos para el agarre cilíndrico se muestran a continuación en las *figuras 5.42, 5.43 y 5.44* para cada dedo respectivamente. Al igual que en el controlador PID se tiene que los datos de color rojo (REF) corresponden a la trayectoria, los datos en

color azul (DE) son los datos experimentales y los datos de color negro (SM)corresponden a la salida simulada de los modelos.

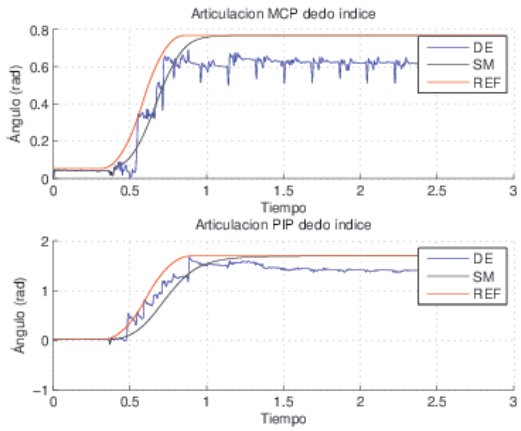


Figura 5.42: Comparación de datos del modelo y datos experimentales: dedo índice compensador por adelanto.

En la anterior *figura 5.42* se aprecia un buen seguimiento de la trayectoria en el transitorio, sin embargo ambas articulaciones no alcanzan la consigna y mantienen un error, el cual se podría disminuir realizando un ajuste a los compensadores.

La siguiente *figura 5.43* muestra un buen seguimiento de la trayectoria, aunque es mucho mejor en la articulación PIP que en la MCP, donde existe un error mayor. De lo anterior se puede decir que hay una buena correlación entre los datos experimentales y los datos de simulación para las dos articulaciones del dedo índice.

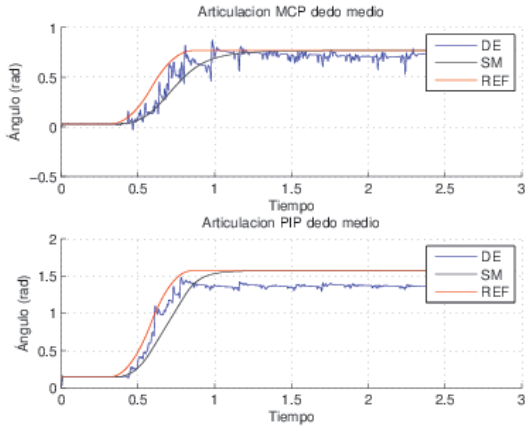


Figura 5.43: Comparación de datos del modelo y datos experimentales: dedo medio compensador por adelanto.

Finalmente en la *figura 5.44* se observa que la articulación PIP realiza muy buen seguimiento de la trayectoria y su error es considerablemente bajo, sin embargo la articulación MCP en el transitorio sigue la consigna de forma escalonada, se desearía que se realizara con mayor suavidad. En términos generales para el agarre cilíndrico el seguimiento hecho por las seis articulaciones es buena y como se mencionó anteriormente, se buscaría mejorar el seguimiento de la trayectoria mediante ajustes en los compensadores o realizando una mejor identificación de cada articulación considerando los fenómenos no deseados que existen en el prototipo mecánico.

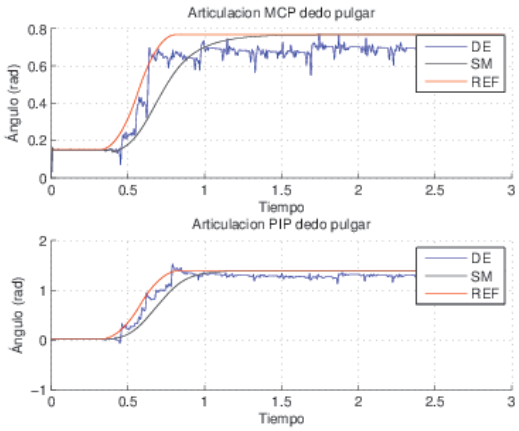


Figura 5.44: Comparación de datos del modelo y datos experimentales: dedo pulgar compensador por adelanto.

La siguiente *Tabla 5.8* muestra el error en estado estacionario para cada articulación en los datos experimentales y datos del modelo.

Articulación	Error en los DE *(rad)*	Error en los SM *(rad)*
MCP índice	0.1526	0.3717 e^{-3}
PIP índice	0.2927	0.2827 e^{-3}
MCP medio	0.0477	0.2213 e^{-3}
PIP medio	0.1954	0.2180 e^{-3}
MCP pulgar	0.0829	0.2114 e^{-3}
PIP pulgar	0.1134	0.2333 e^{-3}

Tabla 5.8: Errores articulares de estado estacionario: datos experimentales y salida modelo compensador por adelanto.

La anterior tabla muestra que los datos en simulación, como era de esperarse, presentan un error insignificante, mientras que en los datos experimentales en todas las articulaciones mantienen error de estado estacionario, siendo el mayor en la articulación PIP del dedo índice donde es de 16.7 grados aproximadamente y el menor error en la articulación MCP del dedo medio que es 2.7 grados aproximadamente, se puede concluir que el compensador por adelanto en general muestra un aceptable funcionamiento, el cual se podría mejorar, como ya se ha mencionado, realizando una mejor calibración o encontrando un modelo más aproximado para las articulaciones que involucre los fenómenos indeseados presentes en la mecánica del prototipo.

5.5.4.2. Agarre de precisión.

La siguiente *figura 5.45* muestra un buen seguimiento de la trayectoria en la articulación MCP, aunque mantiene un error de estado estacionario alto, sin embargo es mucho mejor que las respuesta obtenida en la articulación PIP, en la cual se presenta una oscilación durante el transitorio, finalmente se estabiliza manteniendo un error alto.

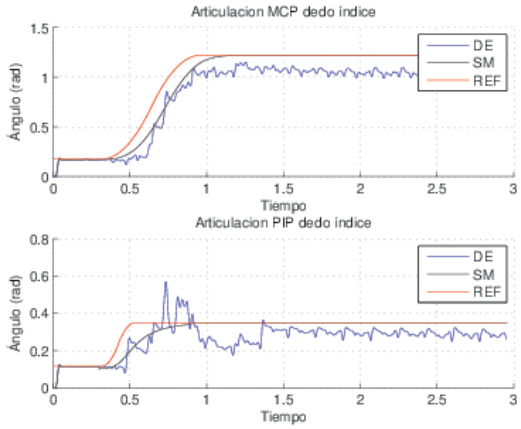


Figura 5.45: Compensador por adelanto, agarre de precisión: dedo índice.

En la *figura 5.46* se observa un seguimiento escalonado de la trayectoria en la articulación MCP, aunque en estado estacionario presenta un error aceptable, por el contrario en la articulación PIP, se aprecia un pobre seguimiento a la consigna llegando solo hasta la mitad, este fenómeno se presentó con regularidad cuando el ángulo en la consigna son menores a 20 grados.

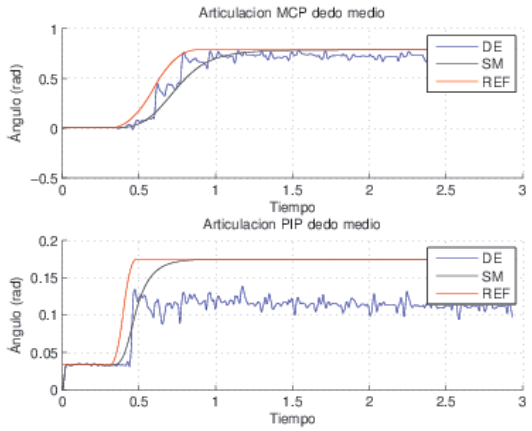


Figura 5.46: Compensador por adelanto, agarre de precisión: dedo medio.

En la *figura 5.47* se observa una respuesta parecida en ambas articulaciones, donde durante el transitorio realizan un seguimiento escalonado de la trayectoria, hasta llegar al estado estacionario con error alto.

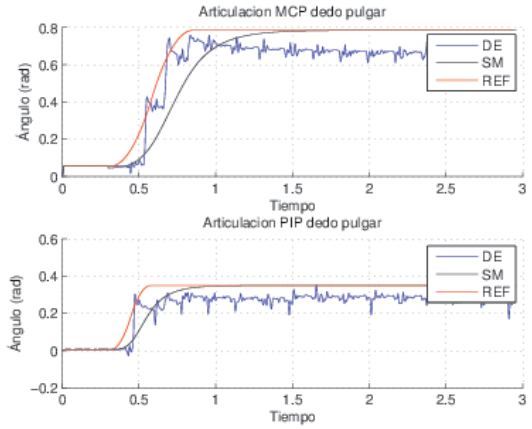


Figura 5.47: Compensador por adelanto, agarre de precisión: dedo pulgar.

5.5.4.3. Agarre de lateral.

La siguiente *figura 5.48* se puede apreciar un muy buen seguimiento de la trayectoria en la articulación MCP, aunque al final mantiene error de estado estacionario, mientras que en la articulación PIP el transitorio presenta un poco de oscilación y al igual que la articulación MCP se estabiliza manteniendo error de estado estacionario.

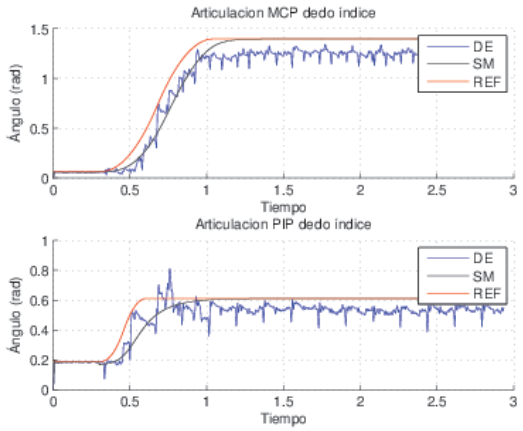


Figura 5.48: Compensador por adelanto, agarre de lateral: dedo índice.

En la *figura 5.49* se observa una respuesta parecida en ambas articulaciones en el transitorio, ambas tienen buen seguimiento de la trayectoria, aunque en estado estacionario la articulación MCP presenta un error bastante menor al de la articulación PIP.

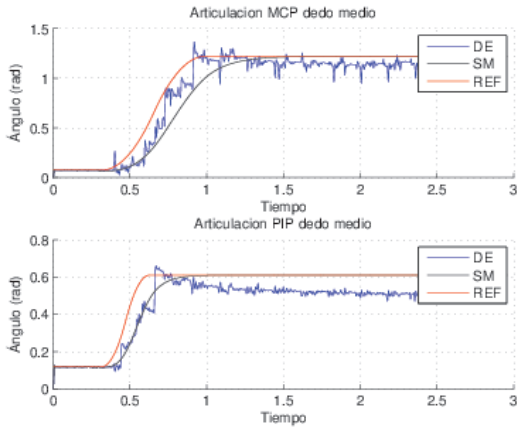


Figura 5.49: Compensador por adelanto, agarre de lateral: dedo medio.

La *figura 5.50* muestra un muy buen seguimiento de la trayectoria en ambas articulaciones en el transitorio, aunque la articulación PIP la realiza más suavemente, sin embargo se nota que la articulación MCP mantiene el buen seguimiento en estado estacionario mientras que la PIP el error es un poco alto.

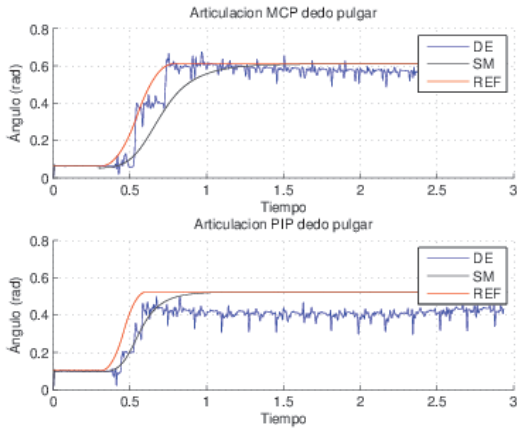


Figura 5.50: Compensador por adelanto, agarre de lateral: dedo pulgar.

5.5.4.4. Agarre de gancho.

La *figura 5.51* muestra buen seguimiento de la trayectoria en ambas articulaciones durante el transitorio, aunque en estado estacionario las dos mantienen error, el cual es mayor en la articulación PIP.

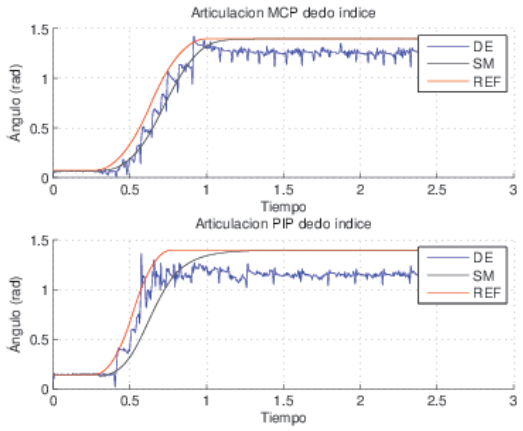


Figura 5.51: Compensador por adelanto, agarre de gancho: dedo índice.

En la siguiente *figura 5.52* se aprecia muy buena respuesta y seguimiento de la trayectoria. En la articulación MCP el seguimiento se realiza de forma continua hasta en el estado estacionario, de forma similar la articulación PIP mantiene el seguimiento pero presentando un pequeño error.

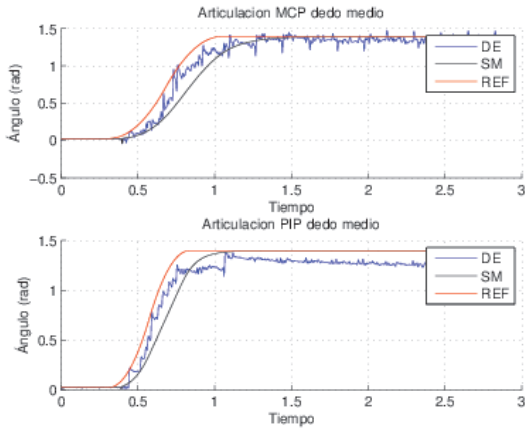


Figura 5.52: Compensador por adelanto, agarre de gancho: dedo medio.

Finalmente la *figura 5.53* muestra un seguimiento escalonado en ambas articulaciones durante el transitorio. La articulación MCP presenta un excelente estado estacionario, en donde el error presentado es muy bajo, por el contrario la articulación PIP se estabiliza manteniendo un cierto error.

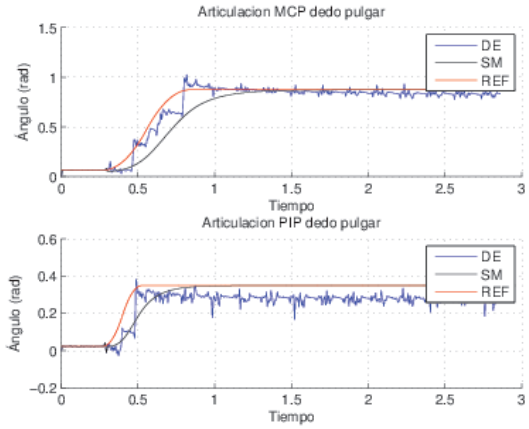


Figura 5.53: Compensador por adelanto, agarre de gancho: dedo pulgar.

Para concluir en anterior análisis se puede decir que el compensador por adelanto de face tuvo un buen funcionamiento en la ejecución de todos los agarres, en algunos mejor que en otros, notándose mejor respuesta cuando los ángulos finales en las consignas de las articulaciones eran superiores a 20 grados respectivamente.

Capítulo 6

Conclusiones

1. Se construyó un modelo virtual del prototipo de mano robótica con el cual es posible observar los diferentes agarres primitivos: de pinza, cilíndrico, lateral y gancho. Su construcción fue hecha con el *toollbox V-Realm Builder 2.0* lo que facilita la interacción con cualquier utilidad de Matlab®/Simulink® en donde es posible diseñar un sin número de controladores que pueden embeberse o empotrarse rápidamente.

2. Se hizo una clara distinción entre los instrumentos de un lazo de control para una articulación, que sirvió de base para diseñar y reproducir los circuitos en una tarjeta de potencia, que permite manipular la tensión aplicada a cada uno de los seis motores asociados a los grados de libertad del prototipo mecánico de mano robótica. Se diseñó y construyó una tarjeta de acondicionamiento de señales que permite la calibración de sensores de *deflexión* ajustando el rango de la variable medida al rango del conversor analógico digital.

3. Se desarrolló una herramienta que permite gestionar la utilidad de una tarjeta FiO Std de prototipado rápido para la implementación, programación en caliente y puesta en marcha de algoritmos de control, permitiéndose mantener resultados previos, para los agarres primitivos anteriormente mencionados.

4. Se contaron con elementos no deseados en las diferentes articulaciones lo que complica obtener una representación cercana entre un modelo matemático y la dinámica real de las articulaciones. Sin embargo fue posible implementar tres controladores: PID, fuzzy y adelanto de fase y sintonizarlos para hacer diferentes agarres primitivos.

Trabajos futuros

1. Dado que es posible usar la tarjeta FiO Std para la adquisición de datos, se desearía profundizar en la identificación de sistemas en lazo cerrado, a fin de evitar que la realización del experimento de adquisición de datos no disminuya la correlación de los datos de entrada con los datos de salida de un sistema.

2. Dado que quedan disponibles tres conversores analógico digitales en la tarjeta FiO Std y se deja una tarjeta para el sensado de corriente en seis los motores, se pueden hacer trabajos futuros que involucren controladores en donde sea necesario medir la corriente en el inducido.

3. Por las cualidades prestacionales de la tarjeta FiO Std y la disponibilidad de la instrumentación de los sensores FSR, se pueden diseñar y embeber controladores de fuerza de agarre.

Bibliografía

[1] Cesar Augusto Quinayás Burgos, *Diseño y construcción de una prótesis robótica de mano funcional adaptada a varios agarres*. Tesis de Maestría Universidad del Cauca, 2010.

[2] Oscar Andrés Vivas Albán, *Diseño y control de robots industriales: teoría y práctica*. Págs. 104-106, 2010.

[3] Karl Johan Åström y Björn Wittenmark, *Computer Controlled Systems: Theory and Design*. Prentice-Hall, 1984.

[4] P. Mosterman, S. Prabhu, A. Dowd, J. Glass, T. Erkkinen, J. Kluza y R. Shenoy, *Embedded Real-Time Control via MATLAB, Simulink, and xPC Target*. Boston, 2005.

[5] Cristian Bazan Orobio, *Sistema de prototipado rápido de control para un motor DC con RTAI-LAB*. Tesis de pregrado Universidad del Cauca, 2012.

[6] Lasse Eriksson, Vesa Hölttä y Malte Misol, *Rapid control prototyping tutorial with application examples*. Sim-Serv, 2004.

[7] Chen, X.M., Gong, X.L., Zhou, H.X., Xu, Z.B., Xu, Y.G., y Kang, C.J., *An Economical Rapid Control Prototyping System Design with Matlab/Simulink and TMS320F2812 DSP: The International MiltiConference of Engineers and Computer Scientists 2010 Voll IMECS*. Hong Kong, 2010.

[8] Yamir Hernando Bolaños Muñoz y Luisa Fernanda Pineda Calvache, *Sistema didáctico para la implementación de controladores digitales*. Tesis de pregrado Universidad del Cauca, 2011.

[9] Balduino Blanqué Molina, *Simulación interactiva de motores de reluctancia autoconmutados*. Tesis doctoral Universidad Politécnica de Catalunya, 2007.

[10] Arduino CC, *http://www.arduino.cc*. [Accesado 15-Dic-2012].

[11] MathWorks, *http://www.mathworks.com/matlabcentral/fileexchange/39037-apm2-simulink-blockset*. [Accesado 15-Dic-2012].

[12] Sim2lab, *User's Guide*. 2011.

[13] Lubin Kerhuel, *http://www.kerhuel.eu/*. [Accesado 6-Jun-2012].

[14] Aimagin, *https://www.aimagin.com/*. [Accesado 15-Oct-2012].

[15] Jaime Leybón, María Ramírez y Verónica Taboada *Sensor foto-eléctrico aplicado al movimiento de los dedos de las manos*, Red de Revistas Científicas de América Latina, el Carribe, España y Portugal, Págs. 59-60, 2005.

[16] Paolo Robertp, *Tecniche di trasduzione di segnali biometrici in riabilitazione funzionale*. Università degli Studi di Padova, 2012.

[17] Pablo Andrés Espinosa y Hernán Augusto Pogo, *Diseño y construcción de un guante prototipo electrónico capaz de traducir el lenguaje de señas de una persona sordomuda al lenguaje de letras*. Tesis de pregrado Universidad Politécnica Salesiana, 2013.

[18] Flex Sensor special edition length, *Datasheet*. 2013.

[19] Bob Mammano, *Current sensing solutions for power supply designers*. Texas Instruments, 2001.

[20] Carlos de la Hoz Najarro, *Puesta en marcha del sensor fuerza/par JR3*. Universidad Carlos III de Madrid, 2011.

[21] Byung June Choi y Jooyoung Chun and Hyouk Ryeol Choi, *Development of anthropomorphic robot hand whit tactile sensor: SKKU Hand II*. Sungkyunkwan University, 2008.

[22] Fio STD , *Datasheet*. 2010.

[23] Interlink Electronics, *Force Sensing Ressistor Integration Guide and Evaluation Parts Catalog*. 2010.

[24] Allegro MicroSystems Inc, *Automotive Grade, Fully Integrated, Hall Effect-Based Linear Current Sensor with 2.3kVRMS Voltage Isolation and a Low-Resistance Current Conductor*. 2006-2009.

[25] Jose S. López Díaz, *Adaptación y traducción Normas ANSI/ISA S 5.1-1984(R1992)* . 2003.

[26] Santiago Galán Morales, *Interfaz Gráfica Para La Simulación De Modelos Dinámicos*. Universidad Carlos III de Madrid, 2011.

[27] Gonzalo Fernández de Córdoba Martos, *Creación de Interfaces Gráficas de Usuario (GUI) con MatLab*. Salamanca, 2007.

[28] Diego Orlando Barragán Guerrero, *Manual de interfaz gráfica de usuario en Matlab, parte 1*. 2008.

[29] Carlos Felipe Rengifo, *Identificación de sistemas dinámicos en lazo cerrado*. Tesis de Maestria Universidad del Valle, 1999.

[30] Karl Astrom, *Control PID avanzado*. Págs. 226-228, 2009.

[31] Katsuhiko Ogata, *Modern Control Engineering*. Pág. 709, 2002.

[32] Richad C. Dorf, *Sistemas de control moderno*. Pág. 258, 2005.

[33] Carlos Gaviria, Jaime Diaz y Victor Mosquera, *Agarre estable de objetos con una prótesis de mano robótica*. Universidad del Cauca, 2008.

[34] Ligos Corporation, *V-Realm Builder: User's guide and reference*. 1996-1997.

[35] Rafael Gutiérrez Olivar, Jesús Lambás Pérez, Esther Pascual Albarracín y Tomás Vázquez Gallego, *Solid Edge v16: Guía de Referencia - Diseño Gráfico*. 2006-2007.

[36] The MathWorks, Inc., *Fuzzy Logic Toolbox User's Guide*. 1995-1999.

[37] Yubel Mendoza y Carlos Silva, *Técnicas de Diseño y Compensación*. Universidad Fermín Toro, 2003.

[38] Elena Muñoz España, *Diseño por Respuesta en Frecuencia, Sistemas de Control Analógico*. Universidad del Cauca, 2010.

Anexo A

Planos diseño en Eagle®

A.1. Tarjeta de acondicionamiento de señales

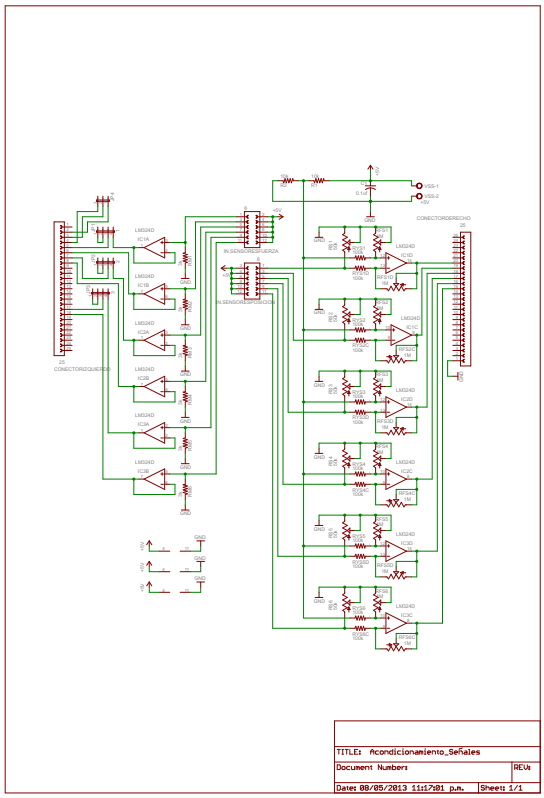


Figura A.1: Esquemático tarjeta de acondicionamiento de señales.

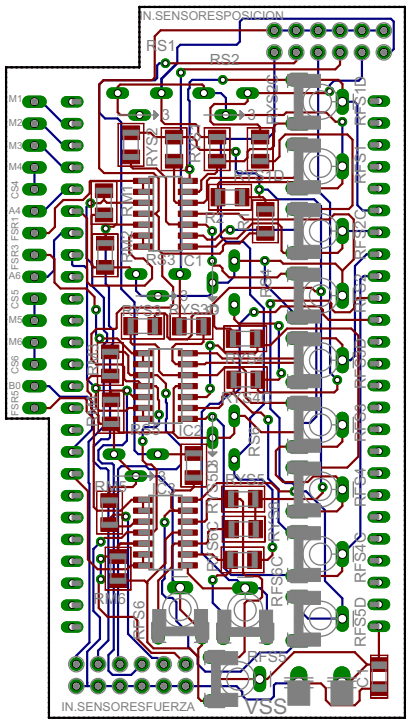


Figura A.2: Diseño PCB tarjeta de acondicionamiento de señales.

A.2. Tarjeta de sensores de corriente

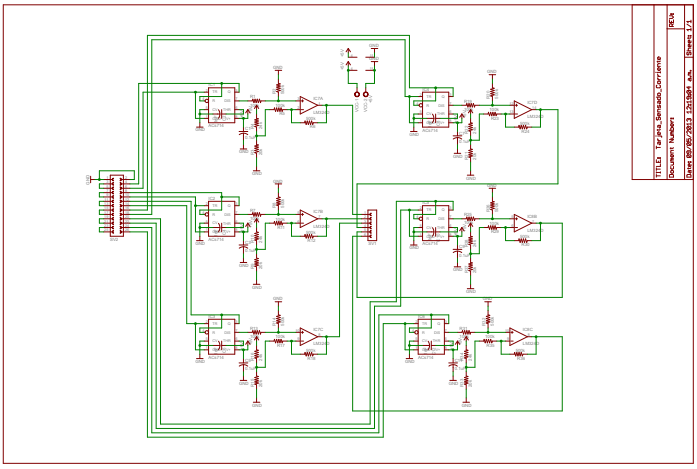


Figura A.3: Esquemático tarjeta sensado de corriente.

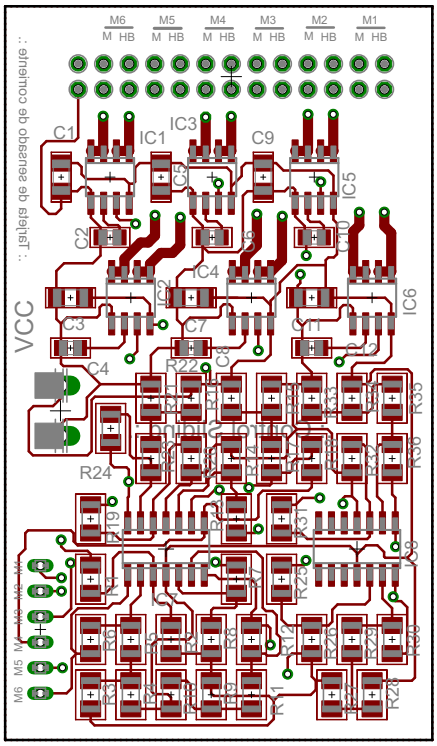


Figura A.4: Diseño PCB tarjeta sensado de corriente.

A.3. Tarjeta de potencia

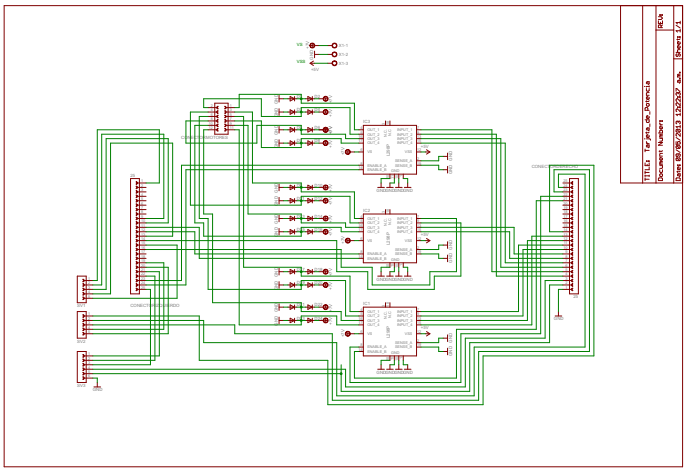


Figura A.5: Esquemático tarjeta de potencia.

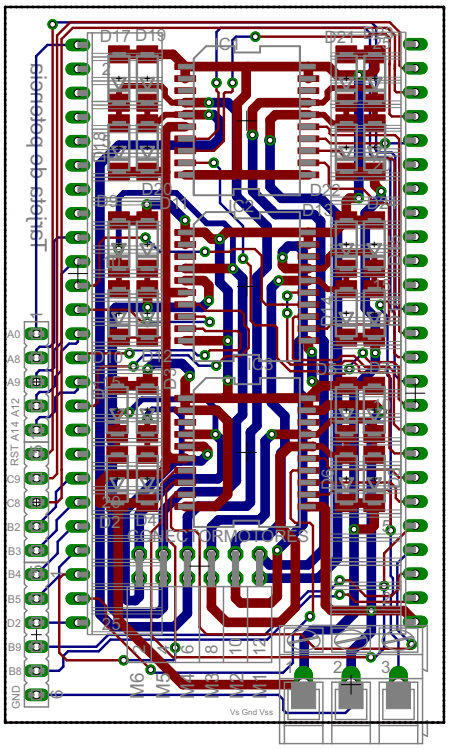


Figura A.6: Diseño PCB tarjeta de potencia.

Anexo B

Exportar piezas de Solid Edge® a V-Realm Builder 2.0.

Ya construidas cada una de las piezas que conforman el prototipo de mano robótica en SolidEdge®, se procede a guardarlas como Imagen/VRML con extensión *.wrl, como se muestra en la *figura B.1*.

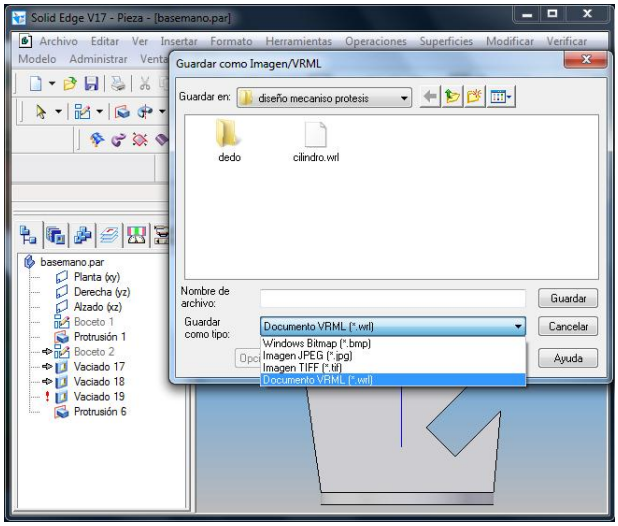


Figura B.1: Guardar piezas en CAD a VRML.

Para acceder al Toolbox de realidad virtual V-Realm Builder 2.0 se despliega la librería de Simulink, seleccionando el Toolbox Simulink 3D Animation, al escoger esta opción se obtiene la vista mostrada en la *figura B.2*, en la cual se muestran los diferentes bloques de realidad virtual que se pueden seleccionar.

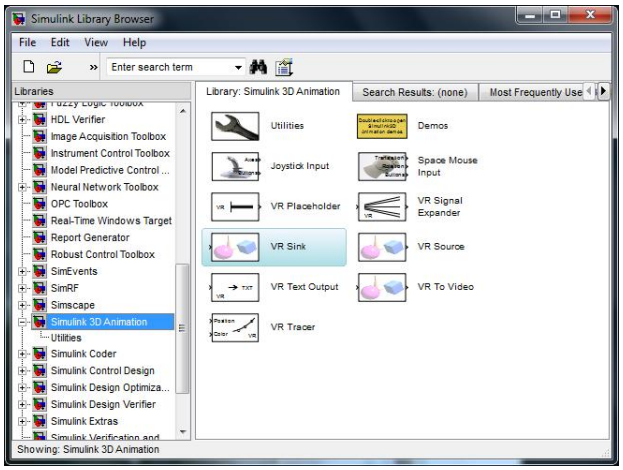


Figura B.2: Herramienta de realidad virtual en la librería de Simulink.

El bloque VR Sink se utiliza para conectar un modelo de realidad virtual construido en la herramienta V-Realm Builder 2.0, con un modelo implementado en Simulink. Éste se inserta en un archivo de Simulink en blanco, luego se procede dar doble clic para abrir el cuadro de parámetros "Parameters: VR Sink" y se da clic en "New" con lo que se abre una nueva ventana con el programa V-Realm Builder 2.0, como se ve en la *figura B.3*.

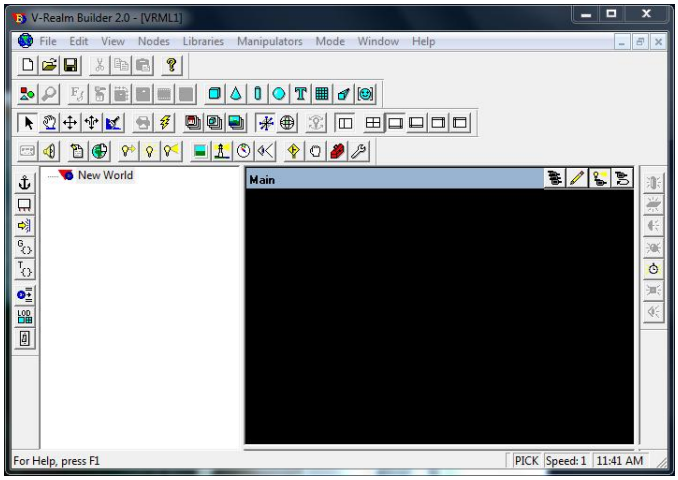


Figura B.3: Ventana de V-Realm Builder 2.0.

En la parte superior se observa la paleta de componentes, donde se encuentran todos los objetos necesarios para la construcción de la escena virtual, así como las acciones que se pueden aplicar. A la izquierda de la pantalla se puede ver un árbol donde se muestran todos los nodos de la escena, donde se accede a sus campos para modificar sus valores. En el centro se visualiza la escena que se está construyendo.

Para empezar a ensamblar el prototipo de mano robótica primero de debe insertar una nueva transformación, la cual será la del padre principal, que en este caso es la palma de la mano, *figura B.4*.

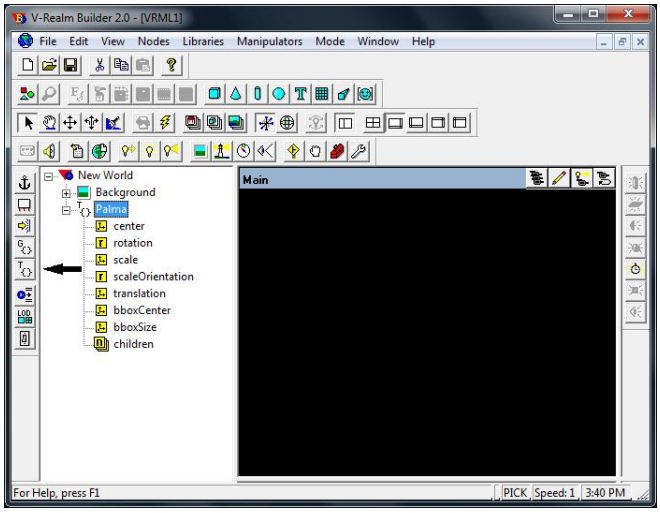


Figura B.4: Agregando una nueva transformada.

Para importar la palma de la mano, ésta se debe abrir desde la carpeta donde fue guardada con SolidEdge® con extensión *.wrl, como se ve en la *figura B.5*.

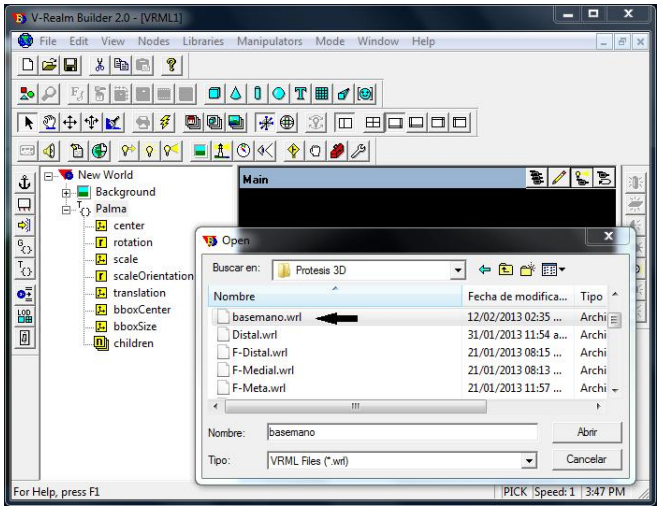


Figura B.5: Abriendo piezas guardadas en VRLM.

Una vez abierta la pieza se le da copiar en el recuadro señalado Group que aparece al final, ver *figura B.6*, con lo que se copia todo el árbol de propiedades de la imagen.

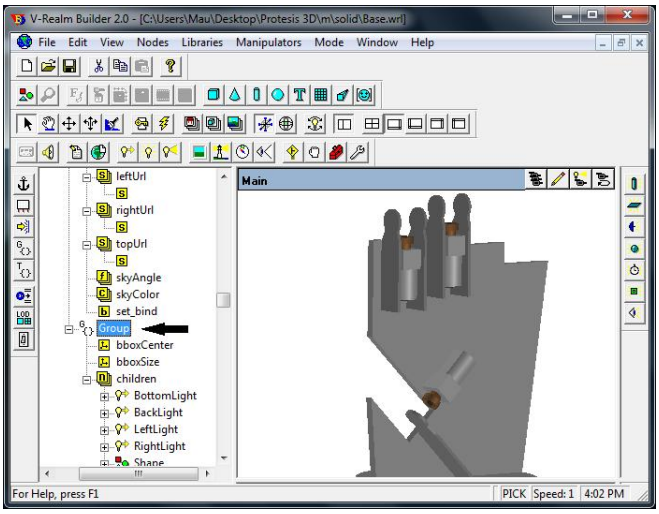


Figura B.6: Copiando el grupo.

Se regresa al archivo donde se inició el proyecto y en la trasformación que se creó en la *figura B.4*, en la parte children se pega el grupo que se copió en la imagen anterior, ver *figura B.7*.

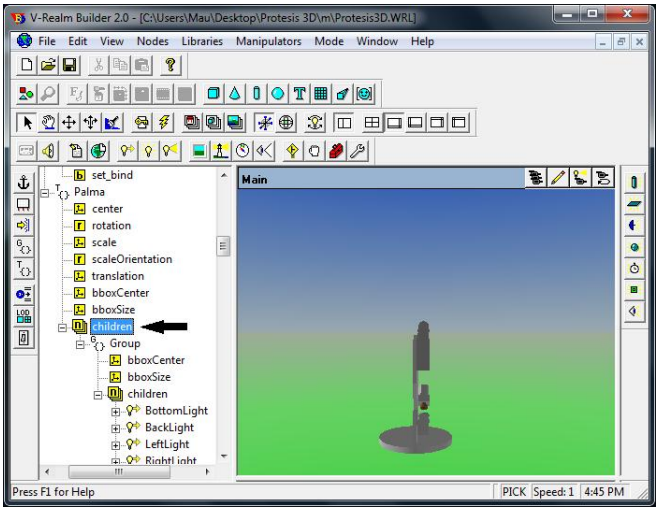


Figura B.7: Pegando el grupo en el proyecto.

El mismo procedimiento se sigue para insertar todas las piezas que forman el prototipo de mano robótica, es necesario trasladar y rotar las piezas para que queden bien ensambladas y en el lugar indicado, siguiendo la estructura jerárquica establecida en el Capítulo 3, hasta llegar al resultado mostrado a continuación, *figura B.8*.

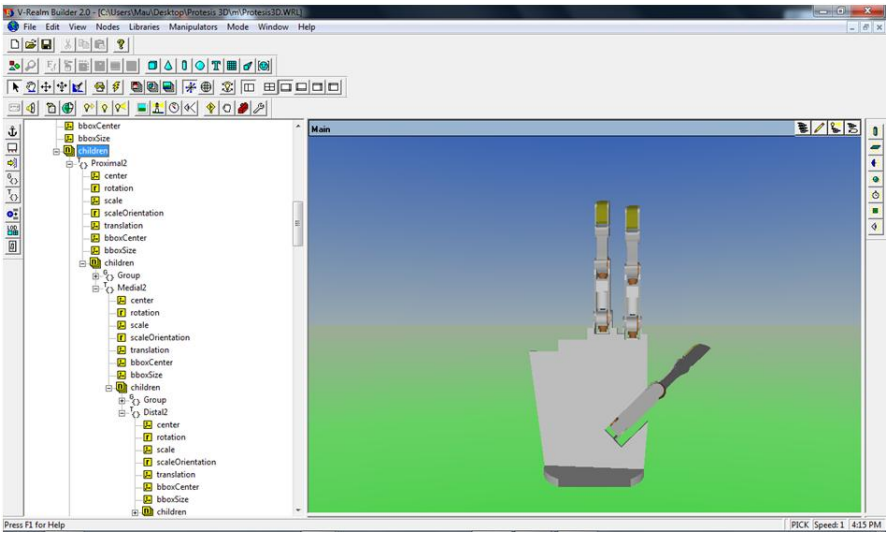


Figura B.8: Prototipo de mano robótica en V-Realm Builder 2.0.

Con el prototipo ensamblado y listo se procede a acceder al bloque de realidad virtual VR Sink donde se despliega una interfaz para cargarlo y configurarle las propiedades del mundo virtual. Esta interfaz sigue la estructura de árbol jerárquica y está formada por los objetos y las acciones implementadas en el escenario virtual conectado con el bloque, mostrado en la *figura B.9*.

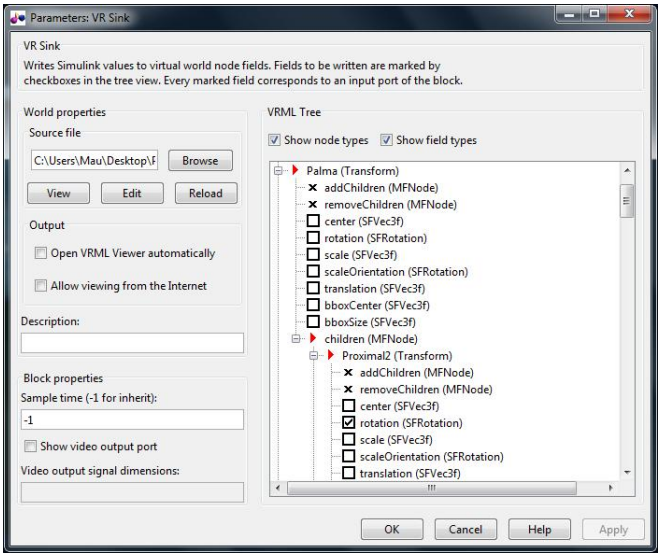


Figura B.9: Interfaz de configuración del mundo virtual.

Aplicando la configuración de las propiedades se finaliza con la construcción del modelo virtual y queda listo para medir su desempeño.

Anexo C

Construcción de la GUI

C.1. Diseño de la GUI

La construcción de la interfaz gráfica de usuario se puede iniciar de dos maneras:

1. Ejecutando la siguiente instrucción en la ventana de comandos.

   >>guide

2. Haciendo un clic en el ícono que muestra la *figura C.1*.

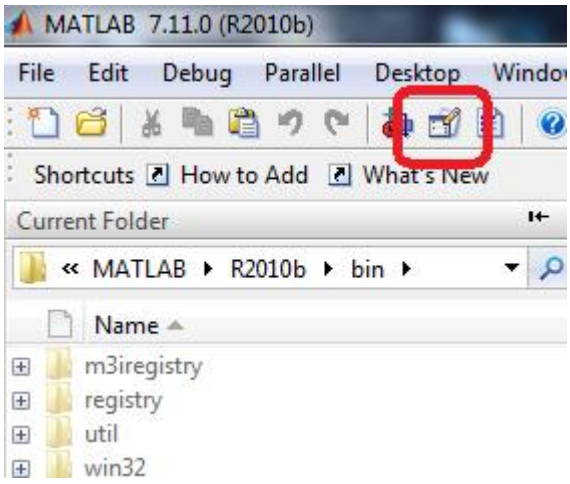


Figura C.1: Icono GUI.

Se presenta el siguiente cuadro de diálogo que muestra la *figura C.2*.

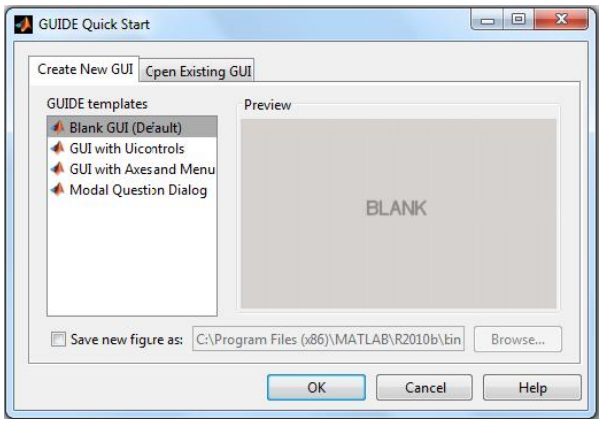


Figura C.2: Ventana de inicio GUI.

Se escoge la opción Blank GUI (Default), y se da clic en OK, se despliega el entorno de diseño para la GUI, *figura C.3*.

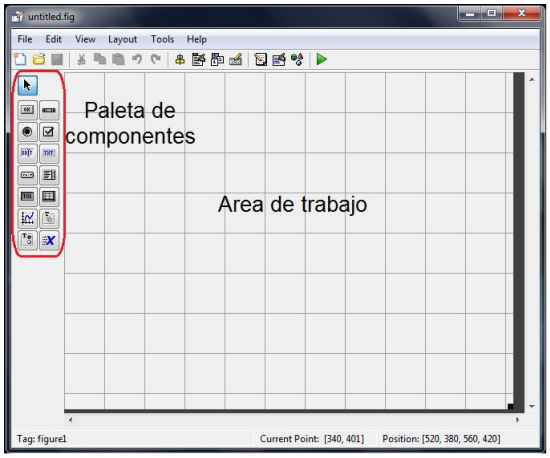


Figura C.3: Entorno de diseño GUI.

C.1.1. GUI del panel de control general

La siguiente *Tabla C.1* muestra una descripción de los componentes usados en esta interfaz:

Componente	Tag	Descripción
Push Button	CREAR_ CONTROLADOR	Despliega el archivo *.mdl para construir el controlador.
Push Button	AYUDA_CREAR	Muestra una ventana emergente con un mensaje de ayuda para la opción Crear controlador.

Push Button	PROTOTIPAR	Despliega el archivo *.mdl donde se hace la copia del controlador diseñado.
Push Button	AYUDA_PROTO	Muestra una ventana emergente con un mensaje de ayuda para la opción Prototipar.
Push Button	DESCARGAR	Ejecuta las funciones para compilar, generar y descargar el código sobre la tarjeta.
Push Button	AYUDA_DESCARGA	Muestra una ventana emergente con un mensaje de ayuda para la opción descarga.
Push Button	REPORTE	Genera la construcción del reporte.
Push Button	AYUDA_REPORTE	Muestra una ventana emergente con un mensaje de ayuda para la opción Reporte.
Push Button	PID	Despliega los archivos *.mdl con el controlador PID desarrollado.
Push Button	FUZZY	Despliega los archivos *.mdl con el controlador FUZZY desarrollado.
Push Button	C_ADELANTO	MDespliega los archivos *.mdl con el controlador por adelanto desarrollado.
Push Button	SALIR	Termina la ejecución de la GUI.
Radio Button	P_IND	Genera la gráfica comparativa para el dedo Índice falange proximal.
Radio Button	M_IND	Genera la gráfica comparativa para el dedo Índice falange medial.
Radio Button	P_MED	Genera la gráfica comparativa para el dedo Medio falange proximal.
Radio Button	M_MED	Genera la gráfica comparativa para el dedo Medio falange medial.
Radio Button	P_PUL	Genera la gráfica comparativa para el dedo Pulgar falange proximal.
Radio Button	M_PUL	Genera la gráfica comparativa para el dedo Pulgar falange medial.

Pop-up Menu	AGARRES	Provee la lista de agarres a seleccionar.
Axes	Axes1	Muestra las gráficas comparativas de las articulaciones.
Axes	Axes2	Muestra la imagen de acuerdo al agarre seleccionado.
Panel	******	Agrupa los botones.

Tabla C.1: Componentes usados en la GUI de control general.

El diseño de la GUI del panel control general se muestra a continuación, *figura C.4*.

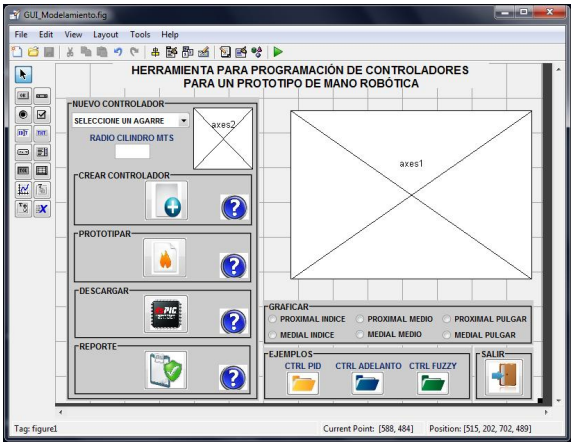


Figura C.4: GUI panel de control general.

C.1.2. Funciones y procedimientos desarrollados en la GUI del panel control general

Una vez que los elementos que conforman la GUI están en posición se procede a editar las funciones de llamada (Callback) de cada uno de ellos, escribiendo el código de Matlab que se ejecutará cuando la aplicación sea utilizada. A continuación se presenta el pseudocódigo de las funciones y procedimientos implementados, en primer lugar la función principal, para después detallar cada uno de las funciones y procedimientos contenidos en ella.

C.1.2.1. Funciones principales

Estas funciones se crean de forma automática al guardar el archivo *.fig, una de las cuales no se debe editar, debido a que contiene la configuración interna que Matlab hace de la GUI.

Algoritmo C.1 Función Principal no editable.

```
function varargout = GUI_Modelamiento(varargin)
% GUI_MODELAMIENTO MATLAB code for GUI_Modelamiento.fig
% GUI_MODELAMIENTO, by itself, creates a new GUI_MODELAMIENTO or raises
% the existing singleton*.
%
% H = GUI_MODELAMIENTO returns the handle to a new GUI_MODELAMIENTO or the

% handle to the existing singleton*.
%
% GUI_MODELAMIENTO('CALLBACK',hObject,eventData,handles,...) calls the local
% function named CALLBACK in GUI_MODELAMIENTO.M with the given input arguments.
%
% GUI_MODELAMIENTO('Property','Value',...) creates a new GUI_MODELAMIENTO or
% raises the existing singleton*. Starting from the left, property value pairs
% are applied to the GUI before GUI_Modelamiento_OpeningFcn gets called. An
% unrecognized property name or invalid value makes property application
% stop. All inputs are passed to GUI_Modelamiento_OpeningFcn via varargin.
%
% *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
% instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES
%
% Edit the above text to modify the response to help GUI_Modelamiento
%
% Last Modified by GUIDE v2.5 27-Jun-2013 15:47:15
%
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @GUI_Modelamiento_OpeningFcn, ...
                   'gui_OutputFcn',  @GUI_Modelamiento_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
```

```
                    'gui_Callback',       [ ] )
if  nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if  nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

La siguiente función si es editable y ésta se ejecuta al correr la GUI, aquí se implementó el código de personalización para los botones y las imágenes iniciales para los axes.

Algoritmo C.2. Función principal editable.

```
function GUI_Modelamiento_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to GUI_Modelamiento (see VARARGIN)

% Choose default command line output for GUI_Modelamiento
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);
% ————PROGRAMACIÓN DE LAS IMÁGENES EN LOS AXES————
axes(handles.axes1)  % función de llamada al axes1
% se lee y asigna la imagen del escudo de la Universidad del Cauca
background = imread('escudo.jpg')
axis off;
imshow(background);  % se visualiza la imagen del escudo en el axes1.
% De igual forma se programa el axes2, solo cambia el nombre de la imagen.
axes(handles.axes2)
```

```
background = imread('matlab.jpg')
axis off;
imshow(background);
% ————PROGRAMACIÓN DE LAS IMÁGENES EN LOS BOTONES ————
[a,map]=imread('nuevo.jpg')
% se realiza el ajuste del tamaño de la imagen en los ejes X y Y
[r,c,d]=size(a);
x=ceil(r/60);
y=ceil(c/45);
g=a(1:x:end,1:y:end,:);
g(g==255)=5.5*255;
% se visualiza la imagen en el botón con TAG NUEVO_MODELO
set(handles.NUEVO_MODELO,'CData',g);
% Esta misma programación se repite para todos los demás botones
% —-Se Bloquean los botones para irlos habilitando paso a paso—-
set(handles.CREAR_CONTROLADOR,'Enable', 'off');
set(handles.PROTOTIPAR,'Enable', 'off');
set(handles.DESCARGAR,'Enable', 'off');
set(handles.REPORTE,'Enable', 'off');
set(handles.P_IND,'Enable', 'off');
set(handles.M_IND,'Enable', 'off');
set(handles.P_MED,'Enable', 'off');
set(handles.M_MED,'Enable', 'off');
set(handles.P_PUL,'Enable', 'off');
set(handles.M_PUL,'Enable', 'off');
set(handles.C_ADELANTO,'Enable', 'off');
set(handles.PID,'Enable', 'off');
set(handles.C_FUZZY,'Enable', 'off');
```

En las siguientes funciones se programa el menú de la barra de herramientas que proporciona la GUI.

Algoritmo C.3. Funciones de la barra de herramientas.

```
function ArchivoMenu_Callback(hObject, eventdata, handles)
% Con esta función se busca y abra un documento
```

```
function Abrir_Callback(hObject, eventdata, handles)
[FileName Path]=uigetfile({'*.docx;*.pdf;*.m;*.mdl;*.fig'},'Abrir documento' );
if  isequal(FileName,0)
return
else
open(strcat(Path,FileName));
end
% Esta función despliega Simulink
function SimulinkMenu_Callback(hObject, eventdata, handles)
simulink;
% Con esta función se abre el documento de ayuda
function AyudaMenu_Callback(hObject, eventdata, handles)
open ('ayuda.pdf')
% Esta función da por terminada la ejecución de la GUI
function ResetMenu_Callback(hObject, eventdata, handles)
set(handles.CREAR_CONTROLADOR,'Enable', 'off');
set(handles.PROTOTIPAR,'Enable', 'off');
set(handles.DESCARGAR,'Enable', 'off');
set(handles.REPORTE,'Enable', 'off');
set(handles.P_IND,'Enable', 'off');
set(handles.M_IND,'Enable', 'off');
set(handles.P_MED,'Enable', 'off');
set(handles.M_MED,'Enable', 'off');
set(handles.P_PUL,'Enable', 'off');
set(handles.M_PUL,'Enable', 'off');
set(handles.C_ADELANTO,'Enable', 'off');
set(handles.PID,'Enable', 'off');
set(handles.C_FUZZY,'Enable', 'off');
Descarga;
Simulink.BlockDiagram.deleteContents('Descarga')
save_system('Descarga'); % Guarda Descarga.mdl
close_system('Descarga', 0); % Cierra Descarga.mdl
Modelamiento;
Simulink.BlockDiagram.deleteContents('Modelamiento')
save_system('Modelamiento'); % Guarda Modelamiento.mdl
```

```
close_system('Modelamiento', 0); % Cierra Modelamiento.mdl
Descarga1;
Simulink.BlockDiagram.deleteContents('Descarga1')
save_system('Descarga1'); % Guarda Descarga1.mdl
close_system('Descarga1', 0); % Cierra Descarga1.mdl
% Con esta función se despliega una grafica conparativa de la posición de las seis articulaciones
function GraficarMenu_Callback(hObject, eventdata, handles)
% Se importan los datos del To Worckspace
P_Indice = evalin('base', 'P_Indice');
M_Indice = evalin('base', 'M_Indice');
P_Medio = evalin('base', 'P_Medio');
M_Medio = evalin('base', 'M_Medio');
P_Pulgar = evalin('base', 'P_Pulgar');
M_Pulgar = evalin('base', 'M_Pulgar');
tout = evalin('base', 'tout');
% Se genera la figura de las seis graficas
figure, hold 'on',
subplot(3,2,1), hold 'on',
plot(tout,P_Indice)
title('Articulacion MCP dedo índice'); legend( 'y(t)', 'r(t)')
xlabel('Tiempo'); xlabel('Ángulo (rad)')
% Así mismo para las demas gráficas
```

C.1.2.2. Funciones del panel principal

A continuación se presenta la función que carga las distintas posiciones de las articulaciones para cada agarre, debido a la extensión del código solo están los casos dos, tres y seis, ya que los demás son similares al dos, solo que en estos las posiciones de las articulaciones varían de acuerdo a la tabla 5.4 del Capítulo 5.

Algoritmo C.4. Función de elección de agarres.

```
function AGARRES_Callback(hObject, eventdata, handles)
% Habilito los botones para crear un nuevo modelo o trabajar con un ejemplo desarrollado
set(handles.CREAR_CONTROLADOR,'Enable', 'on');
set(handles.C_ADELANTO,'Enable', 'on');
set(handles.PID,'Enable', 'on');
```

```
set(handles.C_FUZZY,'Enable', 'on');
Tipo_Agarre=get(handles.AGARRES,'Value'); % Se accede al agarre seleccionado
switch Tipo_Agarre
    case 2
% Abre el archivo HOST_MANO_MANDO_TRAYECTORIAS.mdl para asignar agarre
HOST_MANO_MANDO_TRAYECTORIAS;
T_aga = 1;
T_agarre = num2str(T_aga);
set_param('HOST_MANO_MANDO_TRAYECTORIAS/Agarre', 'value', T_agarre);
save_system('HOST_MANO_MANDO_TRAYECTORIAS');
close_system('HOST_MANO_MANDO_TRAYECTORIAS', 0);
% Se abre el archivo leer_angulos.mdl y se obtinen los ángulos finales para el agarre cilíndrico
find_system('Name', 'leer_angulos');
open_system('leer_angulos');
rad = get(handles.RADIO,'String'); % Se obtiene el valor del radio digitado
set_param('leer_angulos/Radio en metros', 'value', rad);
set_param(gcs,'SimulationCommand', 'Start');
Modelamiento; % Abre Modelamiento.mdl para borrar su contenido
% Se borra el contenido de Modelamiento.mdl
Simulink.BlockDiagram.deleteContents('Modelamiento');
save_system('Modelamiento'); % Guarda Modelamiento.mdl
agarres; % Abre agarres.mdl para copiar el tipo de agarre
sys = 'Modelamiento';
Simulink.SubSystem.copyContentsToBlockDiagram('agarres/cilindrico', sys)
save_system('agarres'); % Guarda agarres.mdl
close_system('agarres', 0); % Cierra agarres.mdl
set_param('Modelamiento/Radio en metros', 'value', rad);
axes(handles.axes2); % Se asigna la imagen del agarre en el axes2
i1=imread('cilindrico.jpg');
imshow(i1);
save_system('Modelamiento'); % Guarda Modelamiento.mdl
save_system('leer_angulos'); % Guarda leer_posiciones.mdl
% Importo los angulos del agarre cilíndrico
angulos = evalin('angulos', 'P_Indice');
% Obtengo los angulos finales del agarre cilíndrico
```

```
MCP_IND = angulos(end,1);
PIP_IND = angulos(end,2);
MCP_MED = angulos(end,4);
PIP_MED = angulos(end,5);
MCP_PUL = angulos(end,7);
PIP_PUL = angulos(end,8);
% Convierto los valores a string y actualizar las posiciones en Modelamiento.mdl
A_MCP_IND = num2str(MCP_IND);
A_PIP_IND = num2str(PIP_IND);
A_MCP_MED = num2str(MCP_MED);
A_PIP_MED = num2str(PIP_MED);
A_MCP_PUL = num2str(MCP_PUL);
A_PIP_PUL = num2str(PIP_PUL);
set_param('Modelamiento/Posiciones/Pos-Proxi-Indice', 'value', A_MCP_IND);
set_param('Modelamiento/Posiciones/Pos-Medial-Indice', 'value', A_PIP_IND);
set_param('Modelamiento/Posiciones/Pos-Proxi-Medio', 'value', A_MCP_MED);
set_param('Modelamiento/Posiciones/Pos-Medial-Medio', 'value', A_PIP_MED);
set_param('Modelamiento/Posiciones/Pos-Proxi-Pulgar', 'value', A_MCP_PUL);
set_param('Modelamiento/Posiciones/Pos-Medial-Pulgar', 'value', A_PIP_PUL);
close_system('Modelamiento', 0);
    case 3
% Abre el archivo HOST_MANO_MANDO_TRAYECTORIAS.mdl para asignar agarre
HOST_MANO_MANDO_TRAYECTORIAS;
T_aga = 2;
T_agarre = num2str(T_aga);
set_param('HOST_MANO_MANDO_TRAYECTORIAS/Agarre', 'value', T_agarre);
save_system('HOST_MANO_MANDO_TRAYECTORIAS');
close_system('HOST_MANO_MANDO_TRAYECTORIAS', 0);
Modelamiento; % Abre el archivo Modelamiento.mdl
% Se borra el contenido de Modelamiento.mdl
Simulink.BlockDiagram.deleteContents('Modelamiento');
save_system('Modelamiento'); % Guarda Modelamiento.mdl
agarres; % Abre agarres.mdl para copiar el tipo de agarre
sys = 'Modelamiento';
Simulink.SubSystem.copyContentsToBlockDiagram('agarres/presicion', sys)
```

```
save_system('agarres'); % Guarda agarres.mdl
close_system('agarres', 0); % Cierra agarres.mdl
axes(handles.axes2); % Se asigna la imagen del agarre en el axes2
i1=imread('prisicion.jpg');
imshow(i1);
save_system('Modelamiento'); % Guarda agarres.mdl
close_system('Modelamiento', 0); % Cierra agarres.mdl
% el código para los demás casos son similares al anterior
% .....
    case 4
% .....
    case 5
% .....
    case 6
Editar_Grados;
close(handles.figure1)
end
```

En la siguiente función se presentan los pasos en la creación de los controladores que serán simulados y puestos a punto para posteriormente ser programados y probados sobre el prototipo real.

Algoritmo C.5. Función crear controlador.

```
function CREAR_CONTROLADOR_Callback(hObject, eventdata, handles)
% ----Habilito los botones para el siguiente paso ----
set(handles.PROTOTIPAR,'Enable', 'on');
set(handles.P_IND,'Enable', 'on');
set(handles.M_IND,'Enable', 'on');
set(handles.P_MED,'Enable', 'on');
set(handles.M_MED,'Enable', 'on');
set(handles.P_PUL,'Enable', 'on');
set(handles.M_PUL,'Enable', 'on');
% ----SE ABREN LOS ARCHIVOS NECESARIOS PARA AJUSTES INICIALES ----
Modelamiento; % Abre Modelamiento.mdl que es donde se construye y simula el controlador
Descarga; % Abre Descarga.mdl para borrar su contenido
```

```
save_system('Descarga'); % Guarda Descarga'.mdl
agarres;
sys = 'Descarga';
Simulink.SubSystem.copyContentsToBlockDiagram('agarres/descarga', sys)
save_system('agarres');
close_system('agarres', 0);
save_system('Descarga');
close_system('Descarga', 0); % Cierra descarga.mdl
close_system('leer_angulos', 0); % Cierra descarga.mdl
```

La siguiente función se encarga de la copia de los controladores desarrollados en el paso anterior para pegarlos en el subsistema del archivo que será codificado y descargado sobre la tarjeta de la prótesis real.

Algoritmo C.6. Función Prototipar.

```
function PROTOTIPAR_Callback(hObject, eventdata, handles)
% ----Habilito el boton para el siguiente paso ----
set(handles.DESCARGAR,'Enable', 'on');
% ----SE ABREN LOS ARCHIVOS NECESARIOS PARA AJUSTES INICIALES ----
Modelamiento; % Abre Modelamiento.mdl para copiar los controladores creados
Descarga; % Abre Descarga.mdl para pegar los controladores en el subsistema
DESsys = 'Descarga';
sys = 'Descarga1';
new_system(sys) % Se crea Descarga1.mdl, donde se copian momentáneamente los controladores
open_system(sys) % Abre Descarga1.mdl
% ----SE REALIZA EL PROCESO DE COPIAR Y PEGAR LOS CONTROLADORES ----
% Copia los controladores en Descarga1.mdl
Simulink.SubSystem.copyContentsToBlockDiagram('Modelamiento/Controladores', sys);
save_system('Descarga1');
close_system('Modelamiento', 0); % Cierra Modelamiento.mdl
% Copia los controladores en el subsistema
Simulink.BlockDiagram.copyContentsToSubSystem('Descarga1', 'Descarga/Modulo . . .
. . . Controlador');
Simulink.BlockDiagram.deleteContents('Descarga') % Borra el contenido de Descarga.mdl
save_system('Descarga1');
close_system('Descarga1', 0);
```

```
% —SE CREAN LAS LINEAS INTERCONECTANDO LOS BLOQUES EN Descarga—
add_line(DESsys,'Add/1', 'Modulo Controlador/1', 'autorouting', 'on')
add_line(DESsys,'Modulo Controlador/1', 'Demux/1', 'autorouting', 'on')
save_system('Descarga');
opcion = questdlg('¿Hizo uso de yin para el diseño del controlador¿, . . .
      'Salir',  . . .
      'si', 'no', 'si');
switch  opcion,
  case  'si',
add_line(DESsys,'Mux4/1', 'Modulo Controlador/2', 'autorouting', 'on')
add_line(DESsys,'Target HID Receive/1', 'Modulo Controlador/3', 'autorouting', 'on')
  case  'no',
delete_line('Descarga', 'conversion vol-pos/1', 'Mux4/1')
delete_line('Descarga', 'conversion vol-pos/2', 'Mux4/2')
delete_line('Descarga', 'conversion vol-pos/3', 'Mux4/3')
delete_line('Descarga', 'conversion vol-pos/4', 'Mux4/4')
delete_line('Descarga', 'conversion vol-pos/5', 'Mux4/5')
delete_line('Descarga', 'conversion vol-pos/6', 'Mux4/6')
delete_block([ DESsys'Descarga', '/Mux4' ])
save_system('Descarga'); % Guarda Descarga.mdl
        return
end
save_system('Descarga'); % Guarda Descarga.mdl
```

La ejecución de la siguiente función inicializa la generación y descarga del código sobre la tarjeta que gobierna la prótesis real de mano.

Algoritmo C.7. Función descargar.

```
function DESCARGAR_Callback(hObject, eventdata, handles)
% ——-Habilito el boton para el siguiente paso ——-
set(handles.REPORTE,'Enable', 'on');
% —EJECULA LA FUNCION QUE GENERA LA CONSTRUCCION Y DESCARGA EN LA
TARJETA—
rtwbuild('Descarga');
GUI_Control;
```

La siguiente función genera la creación del reporte que contiene en forma detallada el código, los diagramas de bloques y configuraciones.

Algoritmo C.8. Función Reporte.

```
function REPORTE_Callback(hObject, eventdata, handles)
% —EJECULA LA FUNCION PARA REPORTE DEL CÓDIGO GENERADO—
rtwreport('Descarga');
```

A continuación se presentan las funciones que despliegan cada uno de los documentos de ayuda para cada subpanel del panel principal.

Algoritmo C.9. Funciones de ayuda en los subpaneles.

```
%–DESPLIEGA EL DOCUMENTO DE AYUDA EN EL SUBPANEL CREAR MODELO–
function AYUDA_CREAR_Callback(hObject, eventdata, handles)
helpdlg('Construya los controladores para las articulaciones en el archivo *.mdl . . .', 'Ayuda');
% — DESPLIEGA EL DOCUMENTO DE AYUDA EN EL SUBPANEL DESCARGA —
function AYUDA_DESCARGA_Callback(hObject, eventdata, handles)
helpdlg('Se copiarán los controladores construidos en el paso anterior, recuerde . . .', 'Ayuda');
% — DESPLIEGA EL DOCUMENTO DE AYUDA EN EL SUBPANEL PROTOTIPAR —
function AYUDA_PROTO_Callback(hObject, eventdata, handles)
helpdlg('Se dará inicio a la creación del código y descarga sobre la tarjeta. . . .', 'Ayuda');
% — DESPLIEGA EL DOCUMENTO DE AYUDA EN EL SUBPANEL REPORTE —
function AYUDA_REPORTE_Callback(hObject, eventdata, handles)
helpdlg('Se generará la documentación del código, diagramas de bloques y otras . . .', 'Ayuda');
```

C.1.2.3. Funciones del panel gráficas

En la función que se presenta a continuación se genera la visualización de las gráficas comparativas del seguimiento que se obtiene de las posiciones deseadas, debido a la extensión del código y puesto que es similar para cada articulación, solo se mostrará la programación para el dedo índice en la falange proximal.

Algoritmo C.10. Función de graficas comparativas.

```
function P_IND_Callback(hObject, eventdata, handles)
opcion=get(handles.P_IND,'value'); % Se evalúa si está o no pulsado el radio button
switch opcion
```

```
        case 1 % En caso de pulsar el radio button
            Modelamiento;
    % Inhabilito los demas radio button —
            set(handles.M_IND,'Enable', 'off');
            set(handles.P_MED,'Enable', 'off');
            set(handles.M_MED,'Enable', 'off');
            set(handles.P_PUL,'Enable', 'off');
            set(handles.M_PUL,'Enable', 'off');
            axes(handles.axes1)
            options = simset('SrcWorkspace', 'current'); % Modifica el archivo Simulink
            sim('Modelamient',[ ], options); % Simula el sistema dinámico del archivo Simulink
            plot(tout,P_Indice) % Grafica los valores
            xlabel('Tiempo (Segundos)')
            ylabel('Ángulo (Grados)')
            title('Gráfica comparativa del dedo Índice-Proximal') % Se designa el título de la gráfica
            grid on % Cuadrícula
        otherwise % En caso de dejar de pulsar el radio button
    % Habilito los demas radio button —
            set(handles.M_IND,'Enable', 'on');
            set(handles.P_MED,'Enable', 'on');
            set(handles.M_MED,'Enable', 'on');
            set(handles.P_PUL,'Enable', 'on');
            set(handles.M_PUL,'Enable', 'on');
            axes(handles.axes1)
            i1=imread('escudo.jpg') % Se visualiza la imagen del escudo de la universidad
            imshow(i1);
    end
```

C.1.2.4. Funciones del panel ejemplos

La siguiente función despliega archivos de Simulink con los controladores desarrollados y probados en simulación.

Algoritmo C.11. Funciones para ejemplos.

```
function PID_Callback(hObject, eventdata, handles)
% Inhabilito los demas radio button —
```

```
set(handles.CREAR_CONTROLADOR,'Enable', 'off');
set(handles.C_ADELANTO,'Enable', 'off');
set(handles.C_FUZZY,'Enable', 'off');
% Habilito el boton para el siguiente paso —
set(handles.PROTOTIPAR,'Enable', 'off');
ModelamientoPID;
Descarga1;
sys = 'Desacarga1';
Simulink.SubSystem.copyContentsToBlockDiagram('ModelamientoPID/Controladores', sys)
save_system('Descarga1');
close_system('ModelamientoPID', 0);
Modelamiento;
delete_line('Modelamiento', 'Data Type Conversion1/1', 'Controladores/1')
delete_line('Modelamiento', 'Data Type Conversion2/1', 'Controladores/2')
delete_line('Modelamiento', 'Data Type Conversion/1', 'Controladores/3')
delete_line('Modelamiento', 'Controladores/1', 'Modelamiento de Articulaciones/1')
Simulink.SubSystem.deleteContents('Modelamiento/Controladores');
sysMOD = 'Modelamiento';
Simulink.BlockDiagram.copyContentsToSubSystem('Descarga1','Modelamiento/Controladores');
add_line(sysMod,'Data Type Conversion1/1', 'Controladores/1', 'autorouting', 'on')
add_line(sysMod,'Data Type Conversion2/1', 'Controladores/2', 'autorouting', 'on')
add_line(sysMod,'Data Type Conversion/1', 'Controladores/3', 'autorouting', 'on')
add_line(sysMod,'Controladores/1', 'Modelamiento de Articulaciones/1', 'autorouting', 'on')
Simulink.BlockDiagram.deleteContents('Desacarga1');
save_system('Descarga1');
close_system('Descarga1', 0);
save_system('Modelamiento');
% .....
% La programación para los otros dos controladores ejemplo es muy similar
function FUZZY_Callback(hObject, eventdata, handles)
% .....
% .....
function C_ADELANTO_Callback(hObject, eventdata, handles)
% .....
% .....
```

C.1.2.5. Función salir

Esta función termina la ejecución de la aplicación.

Algoritmo C.12. Función del panel salir.

```
function SALIR_Callback(hObject, eventdata, handles)
opcion = questdlg('¿Desea cerrar la aplicación¿, . . .
        'Salir', . . .
        'si', 'no', 'si');
switch  opcion,
    case  'si',
 % Reseteo la aplicación dejandola en condiciones iniciales
          ResetMenu_Callback(hObject, eventdata, handles)
          delete(gcf)
          close 'all'
   case  'no',
   return
end
```

C.1.3. GUI de ejecución y control

Con esta interfaz el usuario podrá ejecutar la ley de control diseñada, cambiar los estados de la prótesis de mano robótica, graficar y guardar datos. La siguiente *Tabla C.2* muestra una descripción de los componentes usados en esta interfaz:

Componente	Tag	Descripción
Push Button	START	Despliega y ejecuta el archivo *.mdl que interactúa con la tarjeta de la prótesis de mano.
Push Button	STOP	Termina la ejecución de archivo *.mdl para graficar o guardar datos.
Push Button	REPOSO	Pone en estado de reposo a la prótesis de mano.
Push Button	CONTROL	Pone en estado de control a la prótesis de mano.
Push Button	LIBRE	Pone en estado libre a la prótesis de mano.

Push Button	GRAFICAR	Despliega las gráficas comparativas con los datos tomados.
Componente	AJUSTAR	Despliega Descarga.mdl para hacer ajustes a la ley de control.
Push Button	GUARDAR	Guarda los archivos *.mat en la carpeta donde se está trabajando el proyecto.
Push Button	SALIR	Termina con el proceso de ejecución de la ley de control de la maro robótica.

Tabla C.2: Componentes usados en la GUI de control y ejecución.

El diseño de la GUI de control y ejecución se muestra a continuación, *figura C.5*.

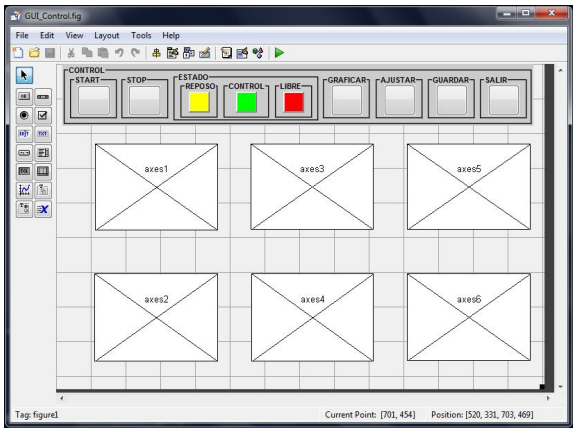


Figura C.5: GUI de control y ejecución.

C.1.4. Funciones de la GUI de ejecución y control

A continuación se presenta el pseudocódigo de las funciones y procedimientos implementados en los botones de esta interfaz.

C.1.4.1. Función del botón start

Con la ejecuciaón de este botón se pone en marcha el archivo *.mdl, quedando listo para los cambios de estado de la prótesis mano robótica.

Algoritmo C.13. Función del botón start.

```
function START_Callback(hObject, eventdata, handles)
find_system('Name', 'HOST_MANO_MANDO_TRAYECTORIAS');
open_system('HOST_MANO_MANDO_TRAYECTORIAS);
```

dacambiar los

```
set(handles.REPOSO,'Enable', 'on');
set(handles.CONTROL,'Enable', 'on');
set(handles.LIBRE,'Enable', 'on');
set(handles.GRAFICAR,'Enable', 'off');
set_param(gcs,'SimulationCommand', 'Start');
```

C.1.4.2. Función del botón stop

Al ejecutar este botón se detiene el proceso y se habilitan los botones para graficar o guardar los datos.

Algoritmo C.14. Función del del botón stop.

```
function STOP_Callback(hObject, eventdata, handles)
find_system('Name', 'HOST_MANO_MANDO_TRAYECTORIAS');
set(handles.GUARDAR,'Enable', 'on');
set(handles.GRAFICAR,'Enable', 'on');
set_param(gcs,'SimulationCommand', 'Stop');
```

C.1.4.3. Funciones de los botones de estado

Estos tres botones cumplen la función del cambio de estado de la protesis de mano robótica (reposo, control y libre).

Algoritmo C.15. Funciones de los botones de estado

```
function LIBRE_Callback(hObject, eventdata, handles)
 % Se iguala el valor de la variable estado a tres que es el correspondiente a libre
Estado = 3;
Estado = num2str(Estad);
 % Se actualizan los valores de la constante estado en el archivo *.mdl
set_param('HOST_MANO_MANDO_TRAYECTORIAS/Estado', 'value' ,Estado);
save_system('HOST_MANO_MANDO_TRAYECTORIAS');
 % De igual forma se establecen estados siguientes solo se varia el valor de la variable estado
function CONTROL_Callback(hObject, eventdata, handles)
 % .....
function REPOSO_Callback(hObject, eventdata, handles)
 % .....
```

C.1.4.4. Función del botón graficar

Mediante este botón se visualizarán las gráficas comparativas correspondientes a las articulaciones y se podrá evaluar el desempeño de la ley de control en el agarre seleccionado.

Algoritmo C.16. Función del botón graficar.

```
function GRAFICAR_Callback(hObject, eventdata, handles)
% Reseteo cada axes para que los datos no se superpongan
axes(handles.axes1)
cla = reset;
% Importo los datos a graficar del To Worckspace
sa1 = evalin('base', 'sa1');
rt1 = evalin('base', 'rt1');
% Asigno los valores a los axes para graficarlos
axes(handles.axes1),hold on,
plot(sa1.time,sa1.signals.values)
plot(rt1.time,rt1.signals.values, 'r'), off;
title('Articulacion MCP dedo índice');
xlabel('Tiempo'); ylabel('Ángulo (rad)');
% De igual forma se codifica para las demas gráficas
```

C.1.4.5. Función del botón ajustar

Con este botón se regresará al archivo Descarga.mdl, con el fin de ajustar el controlador variando los valores de los parámetros.

Algoritmo C.17. Función del botón ajustar.

```
function AJUSTAR_Callback(hObject, eventdata, handles)
Descarga;
```

C.1.4.6. Función del botón guardar

Mediante este botón se guardarán los archivos *.mat que se generan al ejecutar la ley de control.

Algoritmo C.18. Función del botón guardar.

```
function GUARDAR_Callback(hObject, eventdata, handles)
```

```
% Importo los datos a guardar del To Worckspace
sa1 = evalin('base', 'sa1');
rt1 = evalin('base', 'rt1');
sa2 = evalin('base', 'sa2');
rt2 = evalin('base', 'rt2');
sa3 = evalin('base', 'sa3');
rt3 = evalin('base', 'rt3');
sa4 = evalin('base', 'sa4');
rt5 = evalin('base', 'rt5');
sa6 = evalin('base', 'sa6');
rt6 = evalin('base', 'rt6');
% Guardo los datos en la carpeta del proyecto
save('sa1');
save('sa2');
save('sa3');
save('sa4');
save('sa5');
save('sa6');
save('rt1');
save('rt2');
save('rt3');
save('rt4');
save('rt5');
save('rt6');
```

En esta GUI también se encuentra el botón de salir, el cual presenta una programación similar al de la GUI del panel de control general, por lo que no se mostrará nuevamente.

C.1.5. GUI auxiliar para el controlador por adelanto

Con esta interfaz el usuario podrá diseñar un controlador por adelanto, conociendo e ingresando los parámetros de la planta y de diseño. La siguiente *Tabla C.3* muestra una descripción de los componentes usados en ésta interfaz:

Componente	Tag	Descripción
Push Button	CALCULO	Hace el cálculo de los parámetros del controlador por adelanto.
Push Button	BODE	Despliega los diagramas de bode en los axes de la GUI.
Push Button	SALIR	Termina la ejecución de la GUI.
Edit Text	Kplanta	Se ingresa el valor de la constante K de la planta.
Edit Text	Wnplanta	Se ingresa el valor Wn de la planta.
Edit Text	Sitaplanta	Se ingresa el valor de Sita de la planta.
Edit Text	kvplanta	Se ingresa el valor de Kv para el diseño del controlador.
Edit Text	MFDplanta	Se ingresa el valor del MFD para el diseño del controlador.
Static Text	K	Muestra el valor de la constante Kc del controlador diseñado.
Static Text	Z	Muestra el valor del cero Zc del controlador diseñado.
Static Text	P	Muestra el valor del polo Pc del controlador diseñado..

Tabla C.3: Componentes usados en la GUI para el controlador por adelanto.

El diseño de la GUI para el cálculo del controlador por adelanto se presenta a continuación, *figura C.6*.

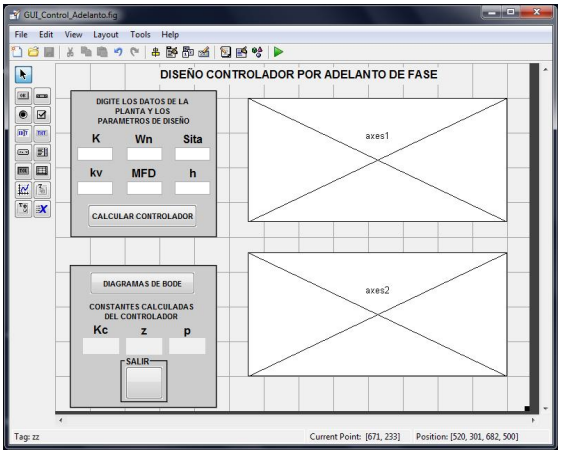


Figura C.6: GUI para controlador por adelanto.

C.1.6. Funciones de la GUI para el controlador por adelanto

A continuación se presenta el pseudocódigo de las funciones y procedimientos implementados en los botones de esta interfaz.

C.1.6.1. Función para el cálculo del controlador por adelanto

La siguiente función hace el cálculo de los parámetros del controlador por adelanto, previamente se le deben ingresar los datos de la planta y los parámetros de diseño.

Algoritmo C.19. Función del botón calculo.

```
function CALCULO_Callback(hObject, eventdata, handles)
% Datos de la planta que se digitan en los Edit Text
a = get(handles.Kplanta, 'String');
b = get(handles.Wnplanta, 'String');
c = get(handles.Sitaplanta, 'String');
% Cambio el tipo de dato a double para poder realizar los calculos
KK = str2double(a);
Wn = str2double(b);
sita = str2double(c);
% Obtengo la funcion de transferencia de la planta
num = KK*Wn^2;
den = [1 2*sita*Wn 0];
fprintf('**Planta del Sistema**')
planta = tf(num,den)
% Exporto los datos de la planta al To Worckspace
assignin('base', 'planta', planta);
% Requerimientos de diseño que se digitan en los Edit Text
d = get(handles.kvplanta, 'String');
e = get(handles.MFDplanta, 'String');
% Cambio el tipo de dato a double para poder realizar los calculos
kv = str2double(d);
MDF = str2double(e);
% Se hace el cauculo de k
kva = polyval(num,0)/polyval(deconv(den,[1 0]),0);
k = kv/kva;
% Se continúa el diseño sobre plantaaux = k*planta
```

```
numaux = k*num;
denaux = den;
plantaaux = tf(numaux,denaux);
% Se halBODEla los vectores de Ganancia (gandB) y Fase (fase)
w = logspace(-1,2,500);
[gan,fase] = bode(numaux,denaux,w);
gandB = 20*log10(gan);
% Se halla el margen de fase: MF
```

Termina la ejecución de archivo *.mdl para graficar o guardar datos.

```
Vectindices = find(gandB<0);
indice = Vectindices(1);
mf = 180-(-fase(indice));
```

Termina la ejecución de archivo *.mdl para graficar o guardar datos.

```
% Se halla la fase a compensar: Fadic
Fadic = MFD-mf+5;
% Se halla alfa
alfa = (1-sin(Fadic*pi/180))/(1+sin(Fadic*pi/180));
% Se halla la ganancia del Compensador
r = 20*log10(sqrt(1/alfa));
% Se halla la nueva frecuencia de cruce de ganancia: wn
Vectindice2 = find(gandB<-r);
indice2 = Vectindice2(1)
wm = w(indice2);
% Se halla T
T = 1/sqrt(alfa)/wm;
% Parámetros del Compensador
Zc = 1/T;
Pc = 1/alfa/T;
Kc = k/alfa;
fprintf('**Compensador Diseñado**')
Comp = tf(Kc*[1 Zc],[1 Pc])
Pc = 1/alfa/T;
% Visualizo los parametros hallados en la GUI
set(handles.K, 'String',Kc);
set(handles.Z, 'String',Zc);
```

```
set(handles.P, 'String',Pc);
fprintf('**Planta Compensada**')
plantacompensada = series(planta,Comp)
% Exporto los datos de la plantacompensada al To Worckspace
assignin('base', 'plantacompensada', plantacompensada);
% Visualizo la grafica de la respuesta en el tiempo en el axes
axes(handles.axes2)
% Importo los datos de las plantas del To Worckspace
planta = evalin('base', 'planta');
plantacompensada = evalin('base', 'plantacompensada');
plot(step(feedback(planta,1))), hold(on, grid on;
plot(step(feedback(plantacompensada,1)), 'r')
title('Respuesta en el Tiempo')
legend('Planta Sin Compensador', 'Planta Compensada')
hold off
```

C.1.6.2. Función para las gráficas de la respuesta en frecuencia

Al pulsar este botón se desplegarán las gráficas de los diagramas de Bode.

Algoritmo C.20. Función del botón Bode.

```
function BODE_Callback(hObject, eventdata, handles)
% Importo los datos de las plantas del To Worckspace
planta = evalin('base', 'planta');
plantacompensada = evalin('base', 'plantacompensada');
axes(handles.axes1)
P = bodeoptions;
P.MagVisible = 'off';
bodeplot(planta,P);
hold on, grid on
bodeplot(plantacompensada,P, 'r');
legend('Planta Sin Compensador', 'Planta Compensada')
title('Respuesta en Frecuencia')
hold off axes(handles.axes2)
P = bodeoptions;
```

```
P.PhaseVisible = 'off';
bodeplot(planta,P);
hold on, grid on
bodeplot(plantacompensada,P, 'r');
hold off
```

El botón salir al igual que los anteriores se codificó de forma similar y no se mostrará en esta sección.

C.1.7. GUI auxiliar para editar las posiciones deseadas en las articulaciones

Con esta interfaz el usuario podrá digitar el valor de los ángulos en grados para las articulaciones. La siguiente *Tabla C.4* muestra una descripción de los componentes usados en esta interfaz:

Componente	Tag	Descripción
Push Button	Posiciones	Actualiza los valores de las posiciones deseadas en Modelamiento*.mdl.
Edit Text	P_I	Posición deseada para la articulación PIP de dedo índice.
Edit Text	M_I	Posición deseada para la articulación MCP de dedo índice.
Edit Text	P_M	Posición deseada para la articulación PIP de dedo medio.
Edit Text	M_M	Posición deseada para la articulación MCP de dedo medio.
Edit Text	P_P	Posición deseada para la articulación PIP de dedo pulgar.
Edit Text	M_P	Posición deseada para la articulación MCP de dedo pulgar.

Tabla C.4: Componentes usados en la GUI editar posiciones.

El diseño de la GUI posiciones se presenta a continuación, *figura C.7*.

Figura C.7: GUI auxiliar editar posiciones.

C.1.8. Función usada en la GUI Editar Grados

Esta función recibe los valores digitados en los Edit Text y actualiza el archivo Modelamiento.mdl.

C.1.8.1. Función del botón Posiciones.

Algoritmo C.21. Función del botón Posiciones.

```
function Posiciones_Callback(hObject, eventdata, handles)
Modelamiento;
Simulink.BlockDiagram.deleteContents('Modelamiento')
save_system('Modelamiento')
agarres;
sys = 'Modelamiento';
Simulink.SubSystem.copyContentsToBlockDiagram('agarres/gancho', sys);
save_system('agarres')
close_system('agarres', 0);
PM = get(handles.P_M,'String');
MM = get(handles.M_M,'String');
PI = get(handles.P_I,'String');
MI = get(handles.M_I,'String');
PP = get(handles.P_P,'String');
MP = get(handles.M_P,'String');
set_param('Modelamiento/Posiciones/Pos-Proxi-Indice', 'value', PI);
set_param('Modelamiento/Posiciones/Pos-Medial-Indice', 'value', MI);
```

```
set_param('Modelamiento/Posiciones/Pos-Proxi-Pulgar', 'value', PM);
set_param('Modelamiento/Posiciones/Pos-Medial-Pulgar', 'value', MM);
set_param('Modelamiento/Posiciones/Pos-Proxi-Medio', 'value', PP);
set_param('Modelamiento/Posiciones/Pos-Medial-Medio', 'value', MP);
save_system('Modelamiento')
close_system('Modelamiento', 0);
GUI_Modelamiento;
close(handles.figure1)
```

Anexo D

Manual de usuario herramienta para la programación de controladores

A continuación se presenta la forma de funcionamiento de la herramienta, la creación, simulación y prototipado rápido de controladores, ejemplos desarrollados, así como algunas precauciones y configuraciones básicas para la tarjeta *FiO Std* de *STMicroelectronics* que gobierna el prototipo de mano real.

D.1. Software requerido

El computador donde funcionará la herramienta debe contar con los siguientes programas:

- Matlab R2010b: Esta versión de Matlab presentó mejor funcionamiento y compatibilidad con la tarjeta, además cuenta con Real-Time Workshop, Real-Time Workshop Embedded Coder, que son Toolbox requeridos para el prototipado rápido con la tarjeta.
- Realview MDKTM for ARM (v4.0).
- Microsoft .NET Framework (v3.5).

D.2. Ejecución inicial de la herramienta

D.2.1. Abrir Matlab R2010b

Ejecute Matlab R2010b haciendo doble clic sobre el acceso directo del escritorio del computador.

D.2.2. Direccionamiento de los archivos de la herramienta

Haga clic sobre la pestaña marcada con el recuadro, busque y direccione la carpeta *Herramienta* en el disco donde esté guardada y dele aceptar, *figura D.1*.

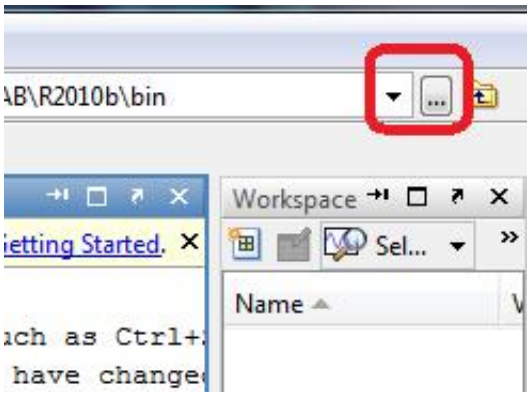


Figura D.1: Direccionamiento de la herramienta.

Asegúrese de que aparezcan todos los archivos en la ventana Current Folder a la parte izquierda de la pantalla de Matlab, *figura D.2*.

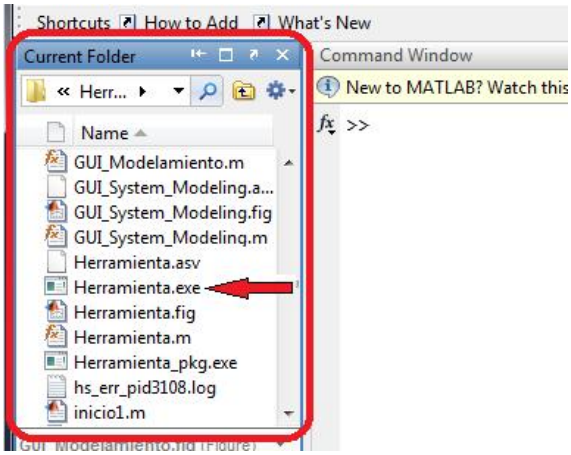


Figura D.2: Archivos de la herramienta.

D.2.3. Abrir el panel de presentación de la herramienta

En los archivos de la herramienta ubíquese y cliquee *Herramienta.exe*, se desplegará el panel de presentación de la herramienta como muestra la *figura D.3*.

Figura D.3: Panel de presentación de la interfaz gráfica de usuario.

De clic sobre el botón Continuar que aparece en la parte inferior derecha de la figura anterior, se desplegará el panel de control general de la interfaz gráfica, *figura D.4*.

Figura D.4: Panel de control general de la interfaz gráfica de usuario.

D.3. Creación de un controlador

D.3.1. Selección de agarre

A la izquierda de la herramienta se encuentra el panel principal llamado *NUEVO CONTROLADOR*, ubíquese en la parte superior y cliquee en la pestaña como se ve en la *figura D.5*, escoja un agarre o seleccione *MANUAL* si desea digitar los ángulos de posición para las articulaciones.

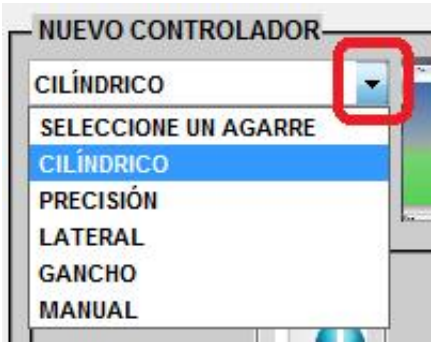


Figura D.5: Selección de agarres.

Note que la imagen que se tenía inicialmente del logo de Matlab cambia de acuerdo al agarre seleccionado, *figura D.6*.

Figura D.6: Selección del agarre cilíndrico.

Para el caso del agarre cilíndrico las posiciones finales de las articulaciones varían según el radio del cilindro. Si desea seleccionar este agarre, asegúrese de digitar previamente el valor del radio (metros), seguido a esto despliegue el menú de agarres y seleccione cilíndrico.

Si seleccionó *MANUAL* se despliega una GUI adicional (*figura D.7*). Digite las posiciones deseadas y de clic sobre el botón *Cargar Posiciones*.

Figura D.7: Interfaz gráfica adicional.

Al seleccionar un agarre se deben abrir por un breve momento los archivos *.mdl donde son cargados los ángulos de las posiciones.

D.3.2. Crear controlador

Pulse el botón central de este subpanel (*figura D.8*) para desplegar el archivo Modelamiento.mdl, *figura D.9*.

Figura D.8: Subpanel Crear controlador.

Ubíquese en el subsistema *Controladores* y construya la ley de control para las articulaciones.

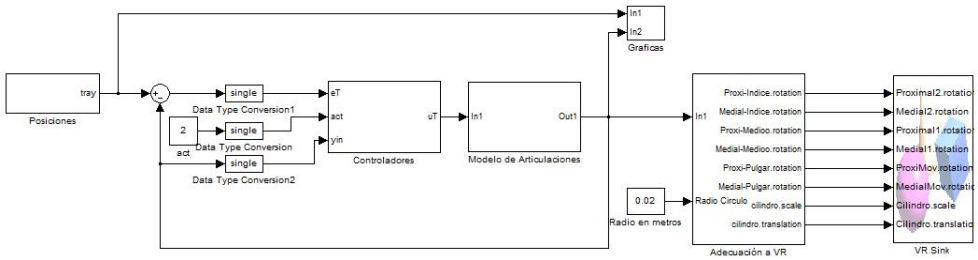


Figura D.9: Modelamiento.mdl para el agarre cilíndrico.

Simule y observe el desempeño de la ley de control desplegando el modelo virtual haciendo clic sobre el bloque VR Sink (*figura D.10*).

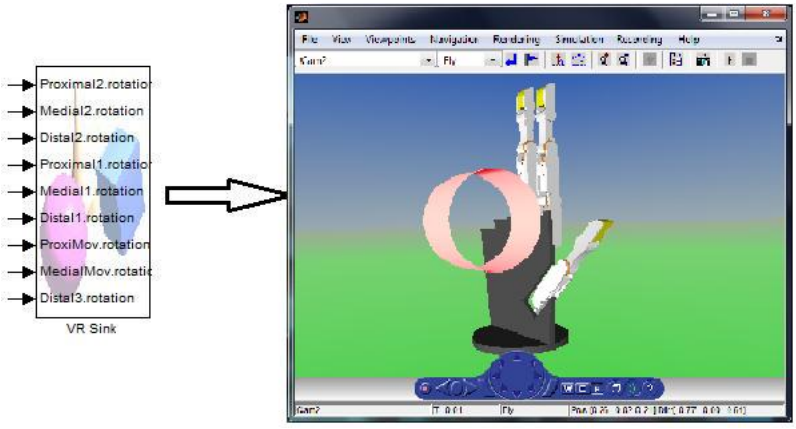


Figura D.10: Bloque VR Sink.

D.3.2.1. Gráficas comparativas

Este panel es de gran ayuda para medir el desempeño de la ley de control diseñada. Pulse un *radio button* (*figura D.11*), esto generará la visualización de la gráfica que compara el seguimiento que tiene la señal de salida sobre la posición deseada de la articulación.

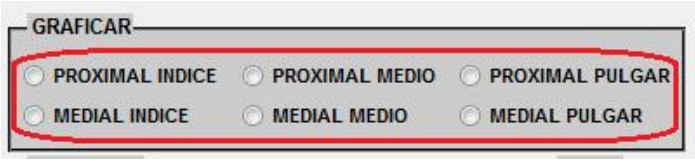


Figura D.11: Radio Buttons.

Este proceso tarda unos pocos segundos, al terminar se graficará (*figura D.12*) cambiando la imagen inicial del escudo de la Universidad del Cauca, al pulsar nuevamente el *radio button* cambiará nuevamente al estado inicial.

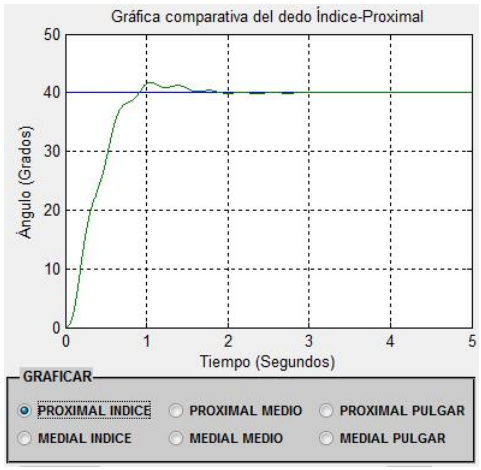


Figura D.12: Gráfica comparativa del dedo Índice Proximal.

Para graficar otra articulación debe mantener los demás *radio buttons* sin pulsar.

Asegúrese de probar y poner a punto la ley de control construida antes de proceder a prototipar, debido a que un mal diseño o unos malos parámetros pueden causar daños sobre el prototipo de mano real.

D.3.3. Prototipar

Pulse el botón para crear una copia de la ley de control diseñada anteriormente, *figura D.13*.

Figura D.13: Subpanel Prototipar.

En este proceso se hará copia de la ley de control diseñada, en un momento aparecerá un cuadro de diálogo, donde se pregunta si se diseñó el controlador tomando la señal *yin*, esto con el fin de generar la conexión de esta señal, responda si o no de acuerdo a su diseño.

En el archivo Descarga.mdl (*figura D.14*) que se desplegó busque y asegúrese de que se pegó la copia de la ley de control en el Bloque Módulo Controladores, también verifique las conexiones del controlador.

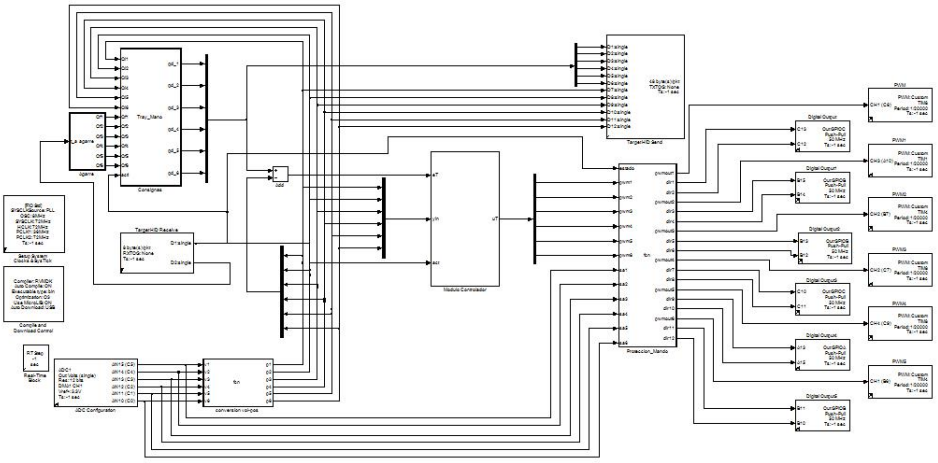


Figura D.14: Descarga.mdl.

Revise la configuración de los canales de los ADC en el bloque *ADC Configuration* que va del canal 10 al 16.

D.3.4. Descargar

Antes de pulsar el botón de descarga conecte el cable de la prótesis de mano robótica en el puerto USB del computador. Seguido a esto, tome la prótesis y ubique la pequeña ranura donde están expuestos el botón *reset* y el *switch 1* de estado (*figura D.15*).

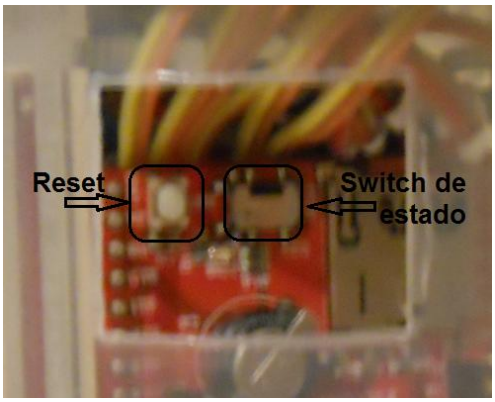


Figura D.15: Botón reset y switch 1 de estado de la tarjeta FiO.

Debe saber que el *switch 1* tiene dos estados *run* (hacia la derecha) y *program* (hacia la izquierda). Como primera medida antes de realizar la descarga, pulse el botón *reset* y enseguida ponga la tarjeta en estado *program* moviendo el switch 1 hacia la izquierda, notará que se prende un pequeño led de color verde, *figura D.16*.

Figura D.16: Modo de programación de la tarjeta FiO.

Teniendo esta configuración, ejecute la siguiente instrucción en la ventana de comandos, esto con el fin de que matlab reconozca la tarjeta de la prótesis.

>> request_productinfo (‘usb‘, ‘name‘)

Luego de que matlab haya reconocido la tarjeta, proceda a dar clic al botón de descarga, (*figura D.17*).

Figura D.17: Subpanel Descargar.

En primera instancia el archivo Descarga.mdl empezará a compilar, al terminar todos los bloques se pondrán de color rojo, *figura D.18*.

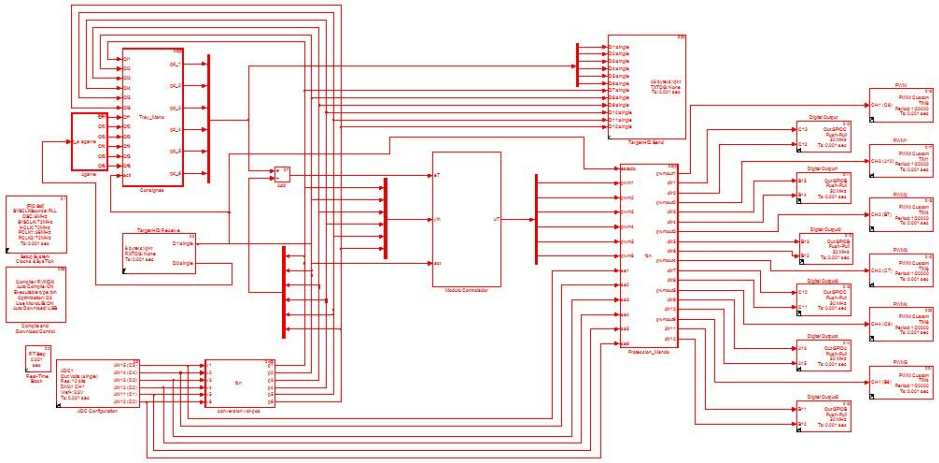


Figura D.18: Compilando Descarga.mdl.

Ya compilado el archivo se inicializará la creación y descarga del código sobre la tarjeta que gobierna la prótesis de mano real. Este proceso puede tardar unos pocos minutos, al terminar se despliega una pequeña ventana que indica que se creó el código y se está descargando en la tarjeta, *figura D.19*.

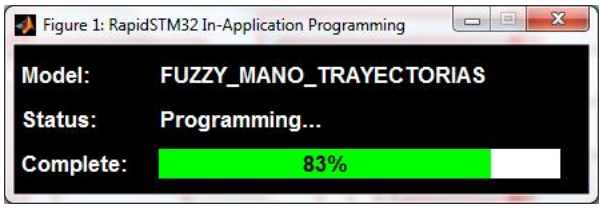


Figura D.19: Ventana de descarga del código sobre la tarjeta FiO

Al terminar la descarga se desplegará la GUI de ejecución y control (*figura D.20*), ésta permite la comunicación usb HID para la adquisición de datos, en este caso r(t) y e(t).

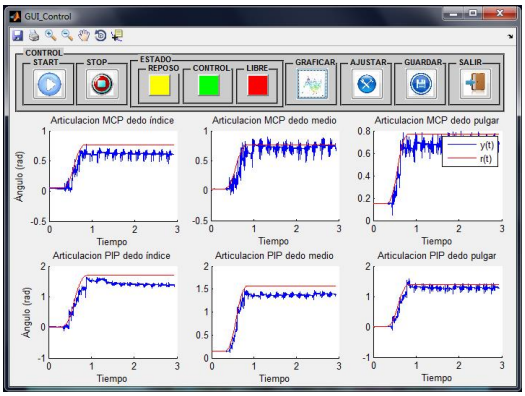


Figura D.20: GUI ejecución y control.

En esta GUI se podrá tener control para la ejecución de la ley de control, enviando el tipo de agarre y el estado de la prótesis de mano, los estados de la prótesis se definieron así:

1. Estado de reposo: Se moverán las articulaciones a una posición inicial, quedando lista para ejecutar el agarre.

2. Estado de ejecución de agarre o control: Se moverán las articulaciones hasta lograr el agarre escogido.

3. Estado libre: La prótesis no ejecutará ningún movimiento en las articulaciones.

De igual forma el tipo de agarre se definió así:

1. Agarre cilíndrico.

2. Agarre de precisión.

3. Agarre lateral.

4. Agarre de gancho.

Teniendo en cuenta las numeraciones presentadas anteriormente, el cambio de estado y de tipo de agarre se hace modificando el valor de dos constantes, ver *figura D.21*.

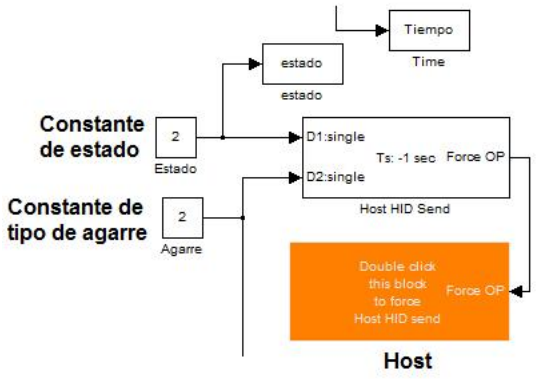


Figura D.21: Constantes de cambio de estado y tipo de agarre

A continuación se muestran los pasos a seguir para probrar la ley de control diseñada y descargada sobre la tarjeta, mediante la GUI de ejecución y control. La siguiente (*figura D.22*) muestra los botones que se deben ir ejecutando.

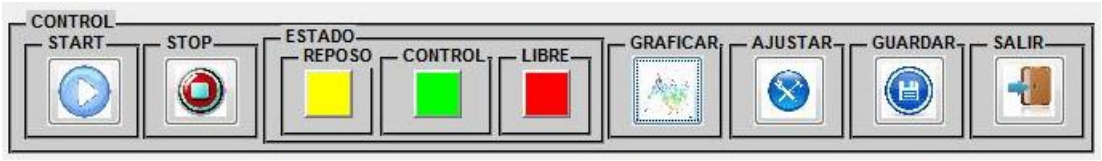


Figura D.22: Botones de la GUI de Ejecución y control.

1. Luego de la descarga del código sobre la tarjeta, cambie el *switch 1* a estado *run* moviéndolo hacia la derecha (*figura D.23*).

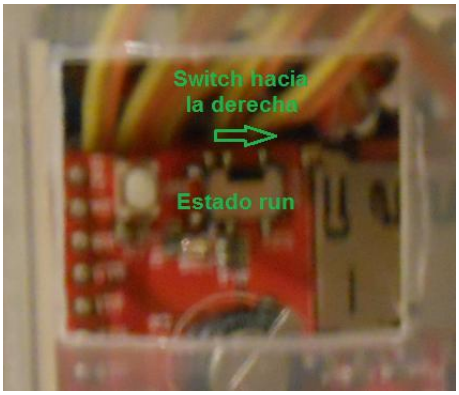


Figura D.23: Switch 1 en estado run.

2. Seleccione un tipo de agarre cambiando el valor de la constante agarre (valor entre 1 y 4), ver *figura D.21*.

3. De clic en el botón Start y ponga la prótesis de mano a estado de reposo, dando clic al botón amarillo, *figura D.24*.

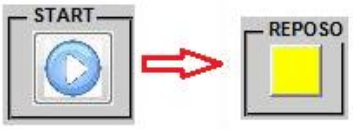


Figura D.24: Pasar prototipo a estado de reposo.

4. De doble clic sobre el bloque Host, las articulaciones se moverán a posición inicial, ver *figura D.21*.

5. Detenga la ejecución dando clic sobre el botón Stop y resetee la tarjeta pulsando el botón Reset, *figura D.25*.

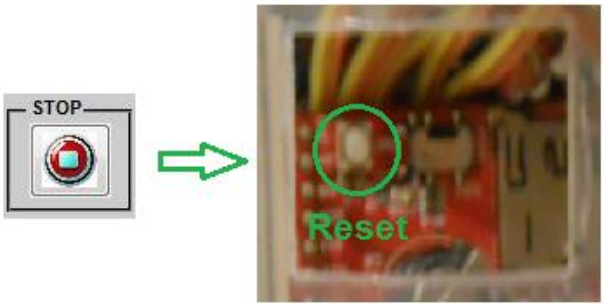


Figura D.25: Preparando la prótesis para ejecutar la ley de control.

6. De clic nuevamente al botón Start y doble clic al bloque Host, espere unos segundos y cambie el estado de la tarjeta a control dando clic sobre el botón verde, la prótesis ejecutará la ley de control y hará el agarre seleccionado. Espere unos segundos mientras se toman suficientes datos, *figura D.26*.

Figura D.26: Ejecución de la ley de control.

7. Cambie el estado de la prótesis a libre dando clic sobre el botón rojo y detenga la ejecución dando clic al botón Stop, *figura D.27*.

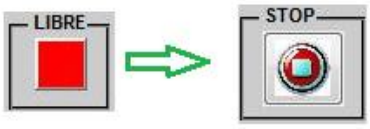


Figura D.27: Pasar la prótesis a estado libre.

8. De clic sobre el botón Graficar (*figura D.28*) y observe el comportamiento de la ley de control diseñada. Repita los pasos hasta aquí hasta estar seguros del comportamiento de la ley de control variando el tipo de agarre.

Figura D.28: Botón para graficar los datos obtenidos.

9. De clic sobre el botón Guardar (*figura D.29*) cuando obtenga unos buenos datos, estos se guardarán en la carpeta donde se está trabajando el proyecto en archivos *.mat.

Figura D.29: Botón para guardar los datos obtenidos.

Si va a ajustar la ley de control en el archivo Descarga.mdl de clic sobre el botón Ajustar (*figura D.30*) y modifique el controlador. Cuando vaya a hacer una nueva programación de la tarjeta diríjase al apartado D3.4 (Descargar) y repita desde ahí los pasos de este manual de usuario.

Figura D.30: Botón para ajustar la ley de control.

D.3.5. Reporte

Para la creación del reporte pulse el botón que se observa a continuación en la *figura D.31*.

Figura D.31: Subpanel Reporte.

Ente proceso puede tardar algunos minutos, al terminar se desplegará el archivo codegen.html y aparecerá *Report complete* en la ventana de comandos de Matlab.

Cada subpanel cuenta con un botón de ayuda, púlselo en caso de tener alguna duda al respecto.

D.4. Desplegar ejemplos desarrollados

Despliegue los ejemplos desarrollados en este trabajo de grado pulsando el botón sobre el tipo de ley de control que desea observar, *figura D.32*.

Figura D.32: Panel de ejemplos de los controladores diseñados.

Esto cargará el controlador sobre el archivo Modelamiento.mdl, con lo cual se puede realizar el proceso de prototipado siguiendo los pasos desde el apartado D3.3 (Prototipar), úselos como guía para la construcción de su ley de control.

Para el controlador por adelanto se desplegará una GUI auxiliar (*figura D.33*), en la cual podrá obtener de forma sencilla los parámetros de esta ley de control, solo digite los datos de la planta y los parámetros de diseño.

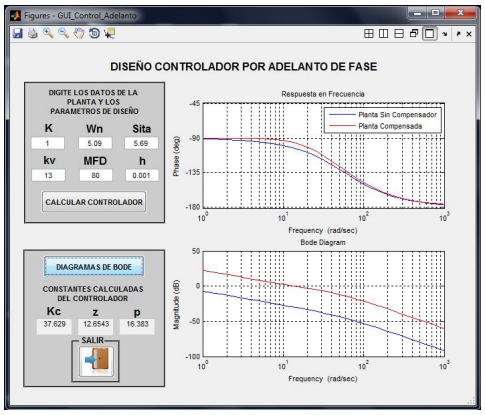


Figura D.33: GUI para controlador por adelanto.

Seguido a esto de clic sobre el botón Calcular controlador, con lo que se obtendrán las constantes del controlador (Kc, Zc y Pc).

D.5. Opciones de la barra de herramientas

La barra de herramientas (*figura D.34*) le ofrece algunas acciones útiles para el manejo de la herramienta, úselas en caso de ser requeridas.

Figura D.34: Barra de herramientas.

Entre las acciones que se encuentran se tienen:

- Archivo: Muestra “Abrir documentos”, aquí se puede buscar para abrir cualquier documento con extensión *.m, *.mdl, *.docx y *.pdf, direcciónelo sobre los discos y carpetas del computador.

- Simulink: Esta acción despliega la ventana de Simulink cuando lo requiera.

- Ayuda: Esta acción es de gran utilidad debido a que despliega el manual de usuario de la herramienta que le servirá como guía general para el manejo de la interfaz gráfica.

- Reset: Cumple la función de reinicio dejando los archivos en estado inicial, cuando se requiera en cualquier momento del proceso de ejecución de la aplicación.

- Graficar: Despliega una gráfica comparativa para la prueba del desempeño de la ley de control en simulación.

D.6. Salir

Pulse este botón (*figura D.35*) en caso de finalizar el uso de la aplicación.

Figura D.35: Panel para salir de la aplicación.

Se cerrarán todos los archivos y figuras, también se deja la Herramienta en estado inicial terminando el proceso de prototipado rápido.

Anexo E

Controlador difuso a través de la herramienta Fuzzy Logic de Matlab®.

Haciendo uso de la herramienta Fuzzy Logic [36] de Matlab se diseñó un controlador difuso, para las articulaciones del prototipo de mano robótica. Este Toolbox permite elegir el método de inferencia, el método de defusificación, definir los conjuntos de entrada y salida e incorporar las reglas que definen el comportamiento del control como se muestra en la *figura E.1*. El cuadro rojo resalta la entrada, el azul la salida y el negro el controlador difuso.

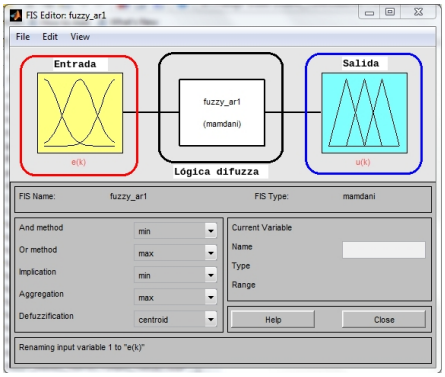


Figura E.1: Ventana FIS Editor.

E.1. Planteamiento del controlador difuso.

El esquema de control *feedback* mostrado en la *figura E.2* corresponde a una articulación del prototipo mecánico. El objetivo es determinar esfuerzo de control ($u(t)$) a partir del error articular existente ($e(t)$), haciendo que la salida ($y(t)$) siga la referencia.

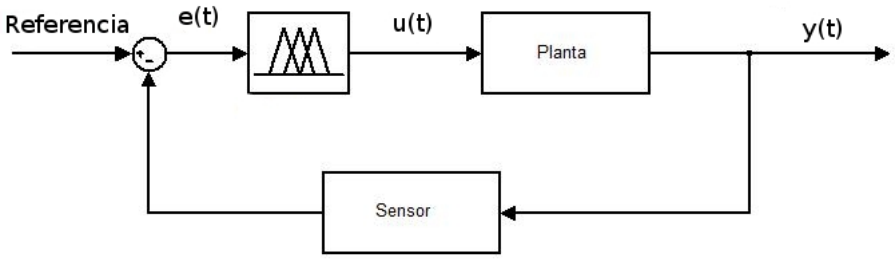


Figura E.2: Representación del diagrama de bloques del sistema fuzzy completo.

E.2. Descripción de la entrada y salida del controlador difuso.

- Señal de error (eT): Es la entrada del controlador, corresponde a la resta entre la señal de referencia y la señal que se obtiene del sensor. Su valor va desde -90 a 90 grados, debido a que tanto la entrada de referencia y la salida del sensor tienen 0 a 90 grados que corresponden al giro máximo que se puede aplicar a cada articulación.

- Esfuerzo de control (uT): Corresponde a la salida del controlador difuso. Su rango va de -93 a 93. El valor absoluto de uT se escribe en el canal PWM de salida, conectado al puente H. Finalmente este regula la energía en forma de voltaje aplicada al motor de imán permanente.

E.3. Diseño del controlador difuso.

Se diseñará el controlador difuso tomando como base tanto para la entrada como para la salida tres conjuntos difusos que representarán diferentes estados. Se obtendrá finalmente el controlador difuso y de determinará si es viable implementarlo de acuerdo a la simulación obtenida en Simulink.

E.3.1. Definición de los conjuntos difusos de la entrada (eT) y la salida (uT).

Para definir los conjuntos de entrada del control se utiliza la ventana de edición Membership Function Editor. Se elige el número de conjuntos y el tipo de función de pertenencia (triangular, trapezoidal, etc.). Las características del conjunto se determinan en el renglón de parámetros.

Conjuntos difusos para la entrada (eT), *figura E.3*.

1. Egn: error grande negativo.

2. Em: error medio.

3. Egp: error grande positivo.

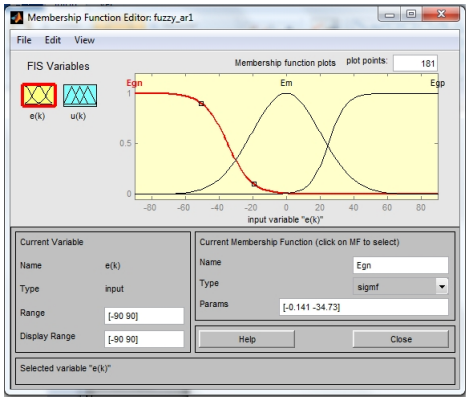


Figura E.3: Conjuntos difusos para la entrada en Membership Function Editor.

De la misma forma que las variables de entrada, se edita los conjuntos de la variable de salida. Es posible elegir el número de conjuntos, la forma del mismo y los puntos que forman al conjunto en el renglón "Params", también se puede definir el rango de la variable en renglón "Range".

Conjuntos difusos para la salida (uT), *figura E.4*.

1. Ugn: esfuerzo de control grande negativo.

2. Um: esfuerzo de control medio.

3. Ugp: esfuerzo de control grande positivo.

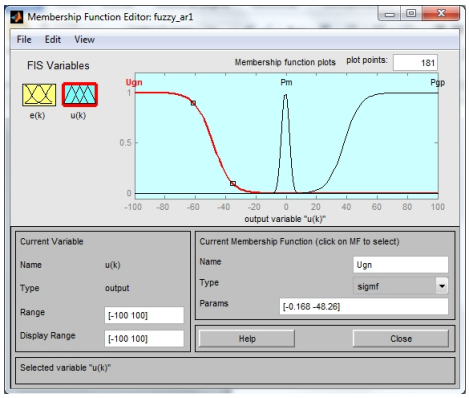


Figura E.4: Conjuntos difusos para la salida en Membership Function Editor.

E.3.2. Definición de las reglas para el controlador difuso.

El siguiente paso es definir las reglas que regirán su comportamiento. En la *figura E.5* se muestra el editor de reglas. En este caso en particular se establecen tres reglas. Cada regla relaciona un conjunto de entrada con un conjunto de salida. Los conectores para las reglas son AND o OR, y las premisas y conclusiones lógicas se diseñan mediante afirmaciones tipo *if.....then*.

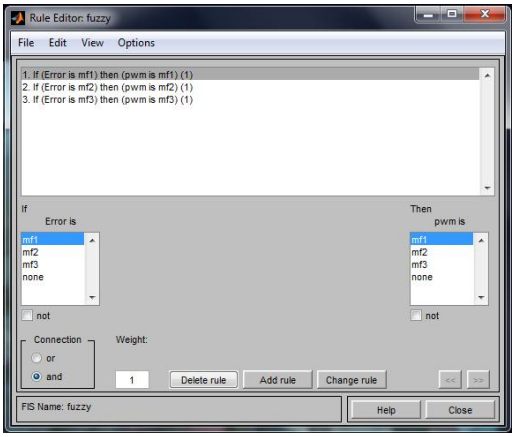


Figura E.5: Ventana Rule Editor.

E.3.3. Superficie correspondiente al control difuso: defusificación

Para que el controlador pueda expresarse con el medio externo, existe una etapa de defusificación que transforma de valores normalizados a valores crisp que pueden ser aplicados sobre el actuador, *figura E.6*.

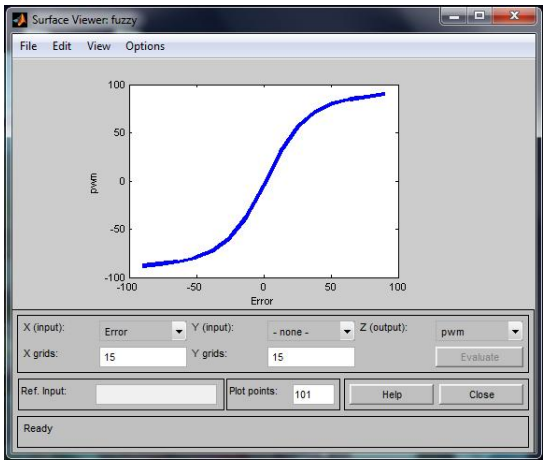


Figura E.6: Superficie del controlador difuso diseñado.

E.3.4. Visualización de las reglas representadas por medio de los conjuntos.

Una vez finalizada la configuración del controlador es posible observar las reglas que se generaron en el editor de reglas y cómo se activan cuando se ingresan valores de entrada. En la *figura E.7* se muestra la ventana de las reglas "Rule Viewer".

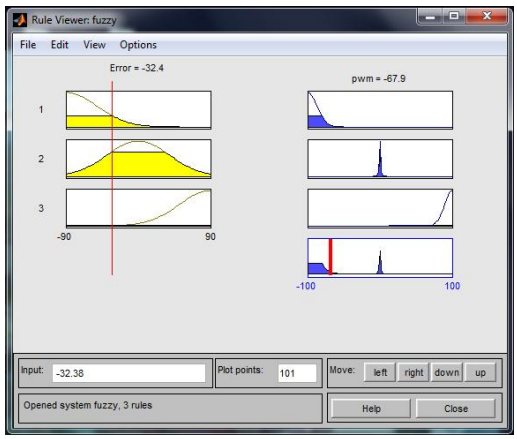


Figura E.7: Ventana Rule Viewer.

En la anterior imagen se pueden observar las variables de entrada así como los conjuntos que las conforman en color amarillo. Las líneas rojas representa los valores de entrada que se ingresan al controlador y se pueden observar los grados de membresía a los que corresponden dentro del conjunto difuso. La variable de salida con sus conjuntos difusos en color azul muestran las reglas que se cumplieron y en la parte inferior de la columna se puede apreciar el polígono generado en la inferencia.

E.4. Simulación del controlador difuso.

Concluido el diseño del controlador difuso se procedió a probar su comportamiento en el siguiente esquema de Simulink, *figura E.8*.

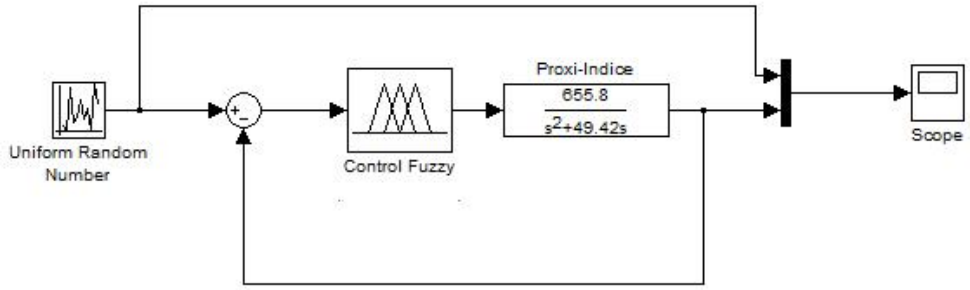


Figura E.8: Esquema preliminar de prueba.

Como se puede observar se utiliza el bloque Uniform Random Number para generar cambios aleatorios en la señal de entrada del controlador, la planta corresponde al modelo identificado de la articulación proximal del dedo índice, obteniendo la siguiente *figura E.9*.

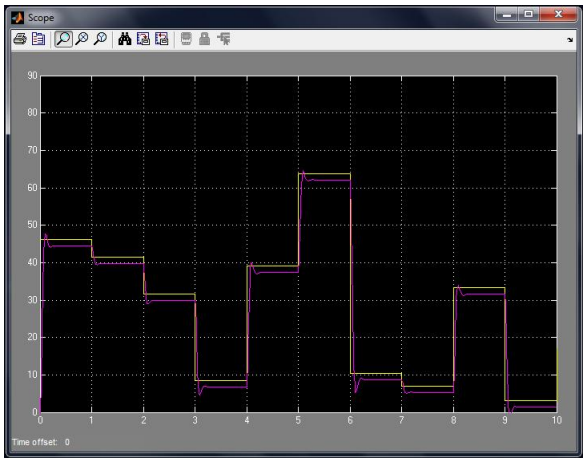


Figura E.9: Comportamiento del controlador difuso.

Como se aprecia se obtuvo un seguimiento de la señal aunque presenta algo de error, se evidencia la necesidad de calibrar el controlador, sin embargo se procedió a simular el controlador difuso para las seis articulaciones del prototipo de mano robótica para un agarre de precisión mediante el siguiente esquema, *figura E.10*.

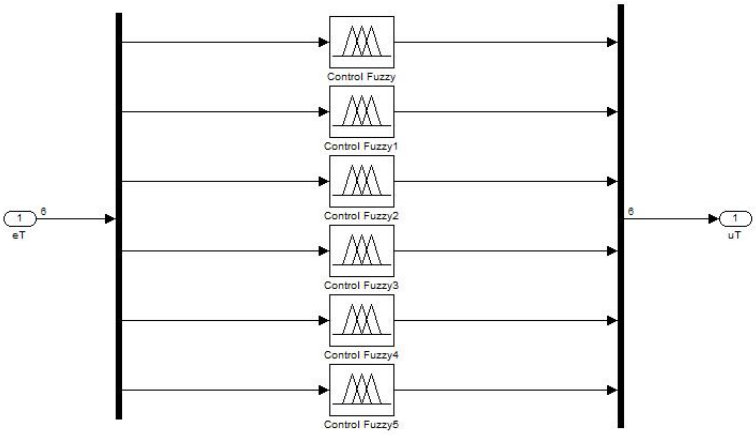


Figura E.10: Bloque del controlador difuso para el agarre de precisión.

En la *figura E.11* se puede observar el seguimiento obtenido de la referencia, de esta manera se confirma el error existente y la necesidad de realizar una nueva calibración para cada controlador difuso.

Figura E.11: Comportamiento del controlador difuso para el agarre de precisión.

Luego de hacer un ajuste a los controladores difusos se logra disminuir el error, a continuación (*figura E.12*) se observa el comportamiento obtenido para el agarre de precisión sobre el modelo virtual.

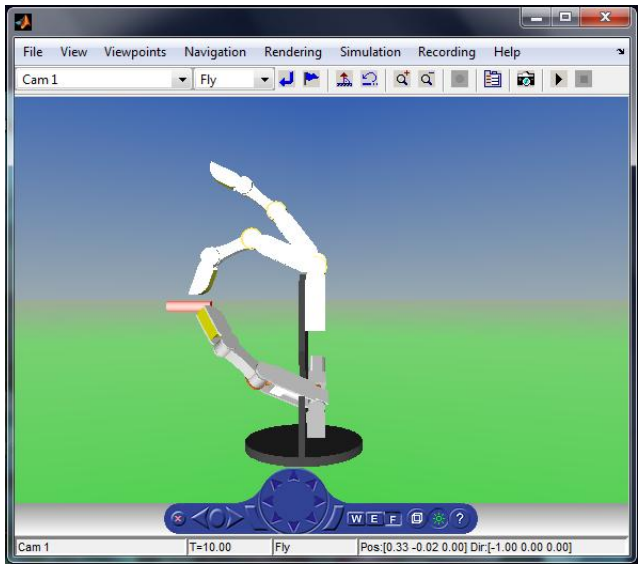


Figura E.12: Control difuso para el agarre de precisión.

Se observa un buen seguimiento con un error pequeño, la siguiente *figura E.13* muestra la respuesta del controlador para todas las articulaciones.

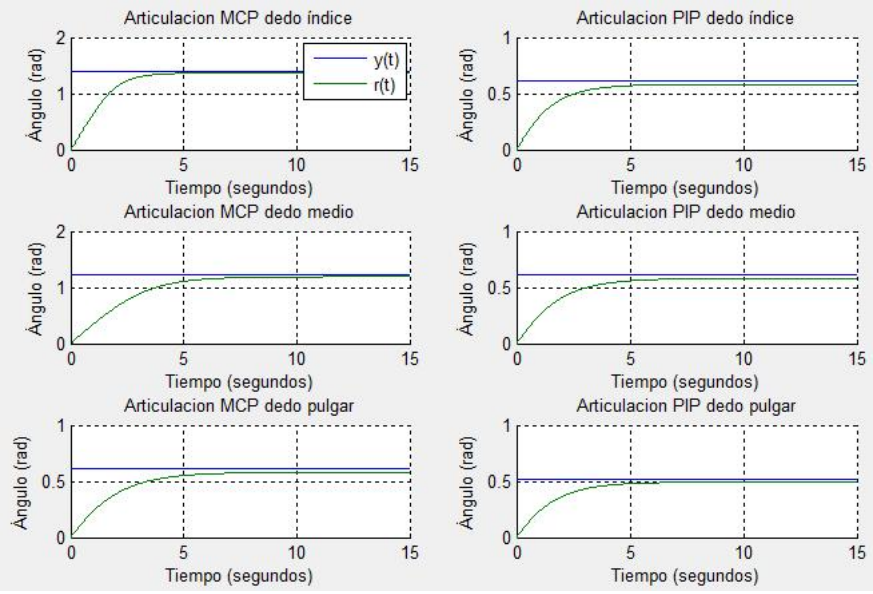


Figura E.13: Respuesta del control difuso en simulación para el agarre de precisión.

E.5. Implementación del controlador difuso

En la *figura E.14* se observa el esquema de bloques usado para obtener la respuesta de los modelos de las articulaciones ante la consigna seguida por el prototipo mecánico

mediante ley de control fuzzy.

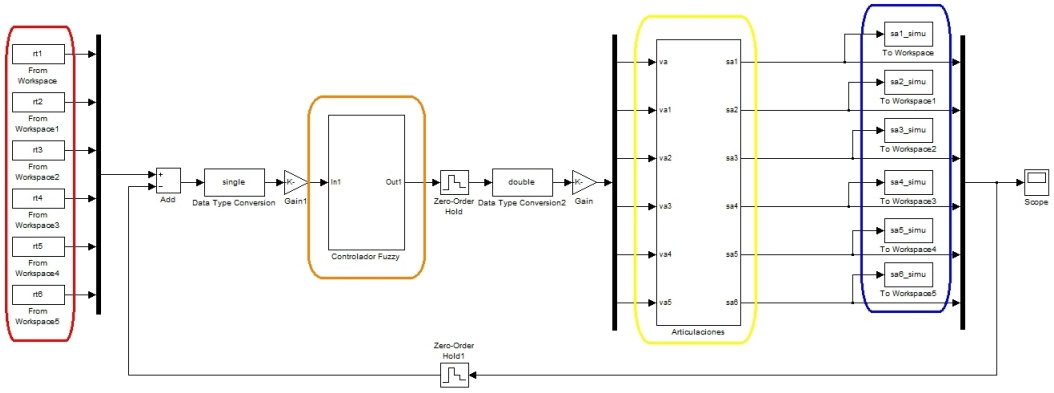


Figura E.14: Esquema para adquisición de datos simulados control fuzzy.

En el rectángulo rojo se observan los *from workspace* que contienen las estructuras con tiempo, donde están almacenadas las consignas articulares. En el rectángulo naranja se resalta el controlador fuzzy implementado mediante la utilidad *Fuzzy Logic de Matlab®*. En el rectángulo amarillo se resalta los modelos descritos en la *Tabla 5.2* para cada una de las articulaciones del prototipo mecánico. En el rectángulo azul se almacena la salidas individuales de cada modelo.

E.5.1. Agarre cilíndrico

En las *figuras E.15, E.16 y E.17*, se observan los datos simulados y experimentales para un agarre cilíndrico, para un cuerpo de radio 2,25cm.

Para un agarre cilíndrico, efectuado con una ley de control fuzzy, se puede observar mejores seguimientos. Particularmente los movimientos son más suaves y mediante una calibración muy fina, es posible lograr errores de seguimiento bastante pequeños. Sin embargo, al igual que el controlador PID, se requiere de la sintonización individual. En la *figura E.17* se observa un mejor seguimiento de la articulación PIP del dedo pulgar del prototipo mecánico. En el diseño de este controlador se considera la zona muerta del actuador, haciendo que para errores pequeños haya esfuerzo de control considerable, para generar giro sobre la articulación.

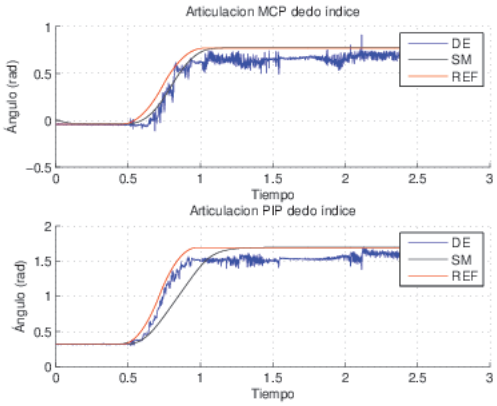


Figura E.15: Control fuzzy, agarre cilíndrico: dedo índice.

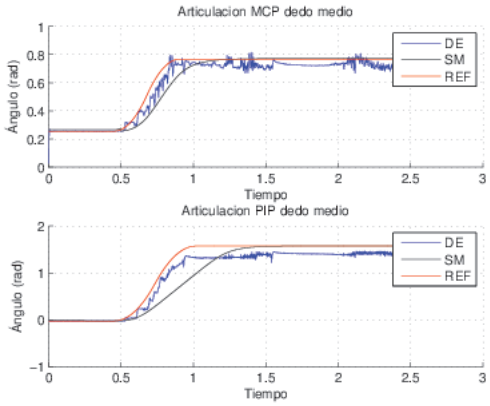


Figura E.16: Control fuzzy, agarre cilíndrico: dedo medio.

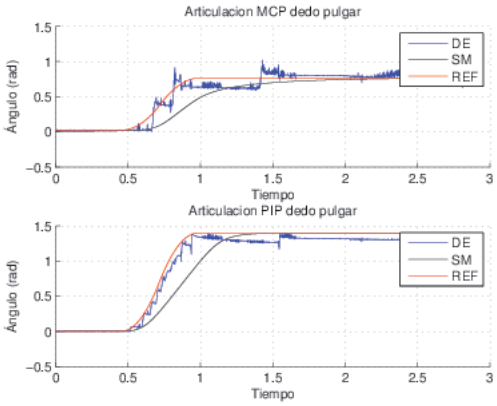


Figura E.17: Control fuzzy, agarre cilíndrico: dedo pulgar.

E.5.2. Agarre de gancho

En las *figuras E.18, E.19 y E.20*, se observan los datos simulados y experimentales para un agarre primitivo tipo gancho.

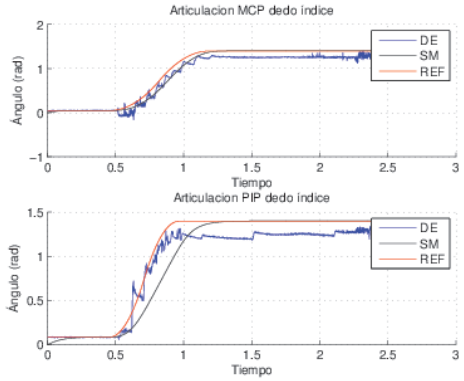


Figura E.18: Control fuzzy, agarre de gancho: dedo índice.

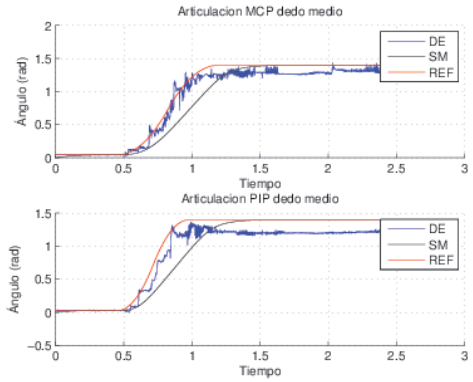


Figura E.19: Control fuzzy, agarre de gancho: dedo medio.

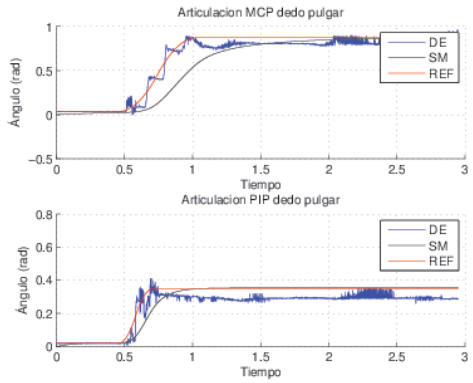


Figura E.20: Control fuzzy, agarre de gancho: dedo pulgar.

Anexo F

Un modelo matemático de un motor DC de imán permanente

Un motor DC es un transductor de par, que convierte energía eléctrica en forma de corriente continua en energía mecánica rotacional. La constitución básica de un motor DC de imán permanente puede observarse en la *figura F.1*.

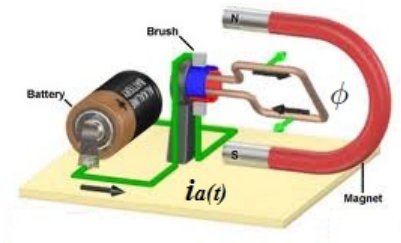


Figura F.1: Constitución básica de un motor DC de imán permanente.

El par generado es directamente proporcional al flujo del campo magnético y a la corriente eléctrica circundante por el inducido y se relaciona mediante la *(Ec. F.1)*:

$$t_e(t) = K_i \cdot \phi \cdot i_a(t) \tag{F.1}$$

Donde $t_e(t)$ es el par generado $(N \cdot m)$, ϕ es el flujo magnético (W_b), $i_a(t)$ es la corriente del inducido (A) y K_i es la constante proporcional $(N \cdot m/W_b \cdot A)$. Dado que el flujo magnético ϕ es constante y aplicando la Transformada de Laplace:

$$T_e(s) = k_m \cdot I_a(s) \tag{F.2}$$

Donde k_m es la constante de par $(N \cdot m/A)$. Inmediatamente al giro del inducido o

eje de rotor, se genera una *fuerza contraelectromotriz*, la cual es proporcional al flujo del campo magnético y a la velocidad del eje, esta relación se expresa mediante la *(Ec. F.3)*.

$$e_b(t) = K_i \cdot \phi \cdot w_m(t) \tag{F.3}$$

Donde $e_b(t)$ es la *fuerza contraelectromotriz* (v) y $w_m(t)$ es la velocidad angular del rotor (rad/s). Las ecuaciones dinámicas del sistema que describen la relación de causa y efecto del diagrama circuital mostrado en la *figura F.2* son la *(Ec. F.4)* y *(Ec. F.7)*, para la parte de constitución eléctrica y mecánica respectivamente.

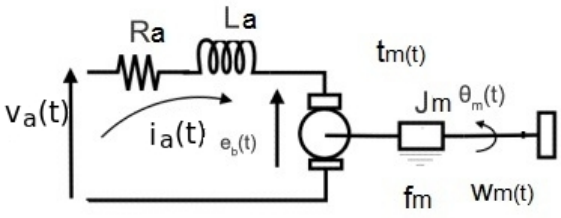


Figura F.2: Configuración de un motor DC de imán permanente.

Variables	**Constantes**
$v_a(t)$: voltaje aplicado	L_a : inductancia de armadura
$e_b(t)$: fuerza contraelectromotriz	R_a : resistencia de armadura
$i_a(t)$: corriente de inducido	k_b : constante de la fuerza contraelectromotriz
$\theta(t)$: posición angular	ϕ : flujo magnético
$w_m(t)$: velocidad angular	k_m : constante de par
$t_e(t)$: par generado	J_m: inercia del rotor
	f_m: coeficiente de fricción viscosa

Tabla F.1: Variables y constantes de un motor DC.

Aplicando la *Ley de Kirchhoff* para tensiones sobre la malla se obtiene:

$$v_a(t) = R_a \cdot i_a(t) + L_a \cdot \frac{di_a(t)}{dt} + K_b \cdot w_m(t) \tag{F.4}$$

Aplicando Transformada de Laplace se obtiene:

$$V_a(s) = R_a \cdot I_a(s) + L_a \cdot s \cdot I_a(s) + K_b \cdot W_m(s) \tag{F.5}$$

Resolviendo para $I_a(s)$:

$$I_a(s) = \frac{V_a(s) - k_b \cdot W_m(s)}{(L_a \cdot s + R_a)} \tag{F.6}$$

Para la parte mecánica se tiene:

$$t_e(t) = J_m \cdot \frac{dw_m(t)}{dt} + f_m \cdot w_m(t) \tag{F.7}$$

Aplicando Transformada de Laplace se obtiene:

$$T_e(s) = J_m \cdot s \cdot W_m(s) + f_m \cdot W_m(s) \tag{F.8}$$

$$T_e(s) = (J_m \cdot s \cdot + f_m) \cdot W_m(s) \tag{F.9}$$

Igualando la *(Ec. F.2)* y la *(Ec. F.9)*:

$$k_m \cdot I_a(s) = (J_m \cdot s \cdot + f_m) \cdot W_m(s) \tag{F.10}$$

Resolviendo para $I_a(s)$:

$$I_a(s) = \frac{(J_m \cdot s \cdot + f_m) \cdot W_m(s)}{k_m} \tag{F.11}$$

Igualando la *(Ec. F.6)* y la *(Ec. F.11)*:

$$\frac{V_a(s) - k_b \cdot W_m(s)}{(L_a \cdot s + R_a)} = \frac{(J_m \cdot s \cdot + f_m) \cdot W_m(s)}{k_m} \tag{F.12}$$

$$k_m \cdot V_a(s) - k_m \cdot k_b \cdot W_m(s) = (L_a \cdot s + R_a) \cdot (J_m \cdot s + f_m) \cdot W_m(s) \tag{F.13}$$

$$k_m \cdot V_a(s) = W_m(s) \cdot ((L_a \cdot s + R_a) \cdot (J_m \cdot s + f_m) + k_m \cdot k_b) \tag{F.14}$$

$$\frac{W_m(s)}{V_a(s)} = \frac{k_m}{(L_a \cdot s + R_a) \cdot (J_m \cdot s + f_m) + k_m \cdot k_b} \tag{F.15}$$

La *(Ec. F.15)* relaciona la velocidad angular con el voltaje aplicado al inducido del motor DC. Para obtener la función de transferencia que relaciona la posición del rotor con el voltaje aplicado se integra a ambos lados la ecuación *(Ec. F.15)* multiplicando por $1/s$.

$$G(s) = \frac{\theta(s)}{V_a(s)} = \frac{k_m}{s \cdot ((L_a \cdot s + R_a) \cdot (J_m \cdot s + f_m) + k_m \cdot k_b)} \tag{F.16}$$

En la *figura F.3* se muestra la función de transferencia de un motor DC en diagrama de bloques, donde la señal de salida es la velocidad angular $W_m(s)$.

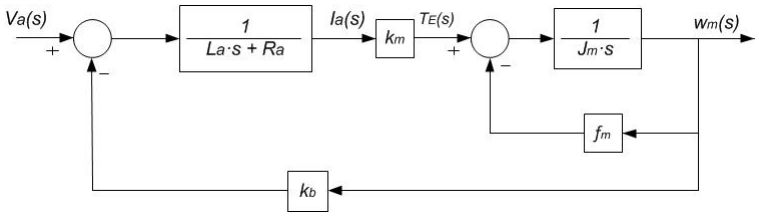


Figura F.3: Diagrama de bloques para salida velocidad angular.

En la *figura F.4* se muestra la función de transferencia de un motor DC en diagrama de bloques, donde la señal de salida es la posición angular $\theta(s)$.

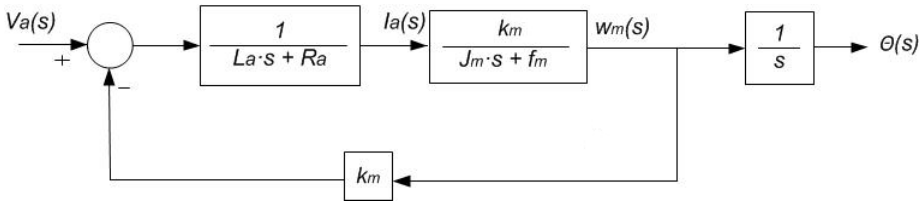


Figura F.4: Diagrama de bloques para salida posición angular.

Realmente $k_b = k_m$ pero generalmente se representan por separado ya que los fabricantes suelen dar valores distintos para cada una de ellas debido al sistema de unidades que utilizan. Si se cumple que $L_a \ll R_a$, el término $L_a \cdot s + R_a$ se puede aproximar a R_a, así mismo convenientemente se aplica que $k_b = k_m$. Esto reduce el orden del modelo matemático, como el mostrado en la *figura F.5.*

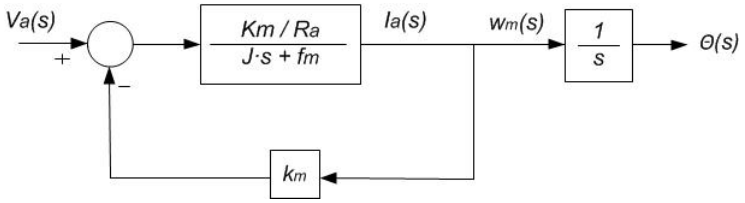


Figura F.5: Reducción del orden del modelo matemático de un motor DC.

Desarrollando el diagrama mostrado en la *figura F.5.*

$$\frac{\theta(s)}{V_a(s)} = \frac{1}{s} \cdot \left(\frac{(k_m/R_a)/(J_m \cdot s + f_m)}{1 + (K_m^2/R_a)/(J_m \cdot s + f_m)} \right) \tag{F.17}$$

$$\frac{\theta(s)}{V_a(s)} = \frac{1}{s} \cdot \left(\frac{k_m/R_a}{J_m \cdot s + f_m + K_m^2/R_a} \right) \tag{F.18}$$

$$\frac{\theta(s)}{V_a(s)} = \frac{1}{s} \cdot \left(\frac{k_m/(J_m \cdot R_a)}{s + (f_m + K_m^2/R_a)/J_m} \right) \tag{F.19}$$

F

El modelo representado en la *(Ec. F.19)* puede ser representado mediante la estructura mostrada en la *(Ec. F.20)*.

$$\frac{\theta(s)}{V_a(s)} = \frac{1}{s} \cdot \left(\frac{w_o^2}{s + 2 \cdot \xi \cdot w_o} \right) \tag{F.20}$$

Donde w_o y ξ pueden ser calculados mediante las *(Ec. F.21)* y *(Ec. F.22)* respectivamente.

$$w_o = \sqrt{\frac{k_m}{J_m \cdot R_a}} \tag{F.21}$$

$$\xi = \frac{f_m + (k_m^2)/(R_a)}{2 \cdot J_m \cdot \sqrt{(k_m)/(J_m \cdot R_a)}} \tag{F.22}$$

R_a y L_a del motor eléctrico pueden medirse directamente mediante un multímetro, J_m y f_m pueden ser encontrados mediante un proceso de identificación donde se realice una adquisición de datos de entrada $v_a(k)$ y de salida $\theta(k)$.