UNIVERSIDAD
DE MÁLAGA

ESCUELA DE INGENIERÍAS INDUSTRIALES

Departamento de Ingeniería de Sistemas y Automática

Áreas de Conocimiento: Modelado de sistemas

TRABAJO FIN DE MÁSTER

Modelado dinámico de una garra subactuada para la interacción física humano-robot (pHRI)

Máster en Ingeniería Mecatrónica

Autor: Adrián Bañuls Arias

Tutor: Dr. Jesús Manuel Gómez de Gabriel

Cotutor: Dr. Juan Manuel Gandarias Palacios

MÁLAGA, Octubre de 2020

Esta pagina ha sido dejada intencionadamente en blanco

DECLARACIÓN DE ORIGINALIDAD DEL PROYECTO/TRABAJO FIN DE MÁSTER

D./ Dña.: Adrián Bañuls Arias

DNI/Pasaporte: 44668456H Correo electrónico: adrianb.arias3@gmail.com

Titulación: Máster en Ingeniería Mecatrónica

Título del Proyecto/Trabajo: Modelado dinámico de una garra subactuada para la interacción física humano-robot (pHRI).

DECLARA BAJO SU RESPONSABILIDAD

Ser autor/a del texto entregado y que no ha sido presentado con anterioridad, ni total ni parcialmente, para superar materias previamente cursadas en esta u otras titulaciones de la Universidad de Málaga o cualquier otra institución de educación superior u otro tipo de fin.

Así mismo, declara no haber trasgredido ninguna norma universitaria con respecto al plagio ni a las leyes establecidas que protegen la propiedad intelectual, así como que las fuentes utilizadas han sido citadas adecuadamente.

En Málaga, a 7 de Octubre de 2020

Fdo.: Adrián Bañuls Arias

Esta pagina ha sido dejada intencionadamente en blanco

Resumen

En el presente trabajo se ha desarrollado el modelo de un manipulador de cadena cerrada y mecanismo de tipo cinco barras, el cual consiste en una garra subactuada con dos dedos, cada uno de ellos con dos falanges y con dos grados de libertad. El modelo estudiado en este proyecto es de naturaleza híbrida, ya que combina la dinámica del actuador con la estática del manipulador. Para aplicar este método, el primer paso es hallar el modelo dinámico del servomotor Dynamixel *XM430-W210-R*, mediante la obtención de sus parámetros y el refinamiento de los mismos a través del software *Matlab & Simulink*. Una vez conocida la respuesta de par del motor en función de la entrada de voltaje PWM, se ha desarrollado el modelo estático de uno de los dos dedos, calculando para ello su matriz de transferencia y su jacobiano, y contemplando un caso de interacción garra-entorno específico. El objetivo es obtener las fuerzas de contacto que ejerce cada falange sobre un objeto determinado, además de conocer las posiciones articulares de ambas falanges en función de la entrada de voltaje PWM y de la elasticidad de dicho objeto.

Palabras clave

Modelo dinámico, Garra subactuada, Modelo Híbrido, Mecanismo de cinco barras, Modelo de un servomotor, pHRI.

Índice general

Acrónimos **10**

Glosario **12**

Lista de figuras **16**

Lista de tablas **20**

1. Introducción **25**

1.1. Objeto 26

1.2. Antecedentes 27

1.3. Metodología del proyecto 27

1.4. Herramientas del proyecto 28

1.5. Estructura de la memoria 29

2. Modelado del servomotor Dynamixel **31**

2.1. Introducción 32

2.1.1. Características del motor 32

2.1.2. Configuración del equipo 33

2.2. Modelado dinámico del motor 35

2.3. Implementación del modelo en *Simulink* 38

2.4. Experimentación y Validación del modelo 41

2.4.1. Experimentos con entrada PWM constante 42

2.4.2. Experimentos con entrada PWM de tipo rampa 44

2.4.3. Reajuste de parámetros 45

2.4.4. Análisis con entrada PWM de tipo secuencia de impulsos 48
2.5. Conclusión de la validación del modelo 50

3. Modelado de la garra subactuada 51

3.1. Introducción 52
3.2. Modelo cinemático 53
3.3. Modelo estático 55
3.3.1. Matriz de transferencia 57
3.3.2. Cálculo del jacobiano 60
3.3.3. Modelo de interacción con el entorno 61
3.4. Modelo híbrido 63

4. Conclusiones y líneas de trabajo futuro 67

4.1. Conclusiones 68
4.2. Líneas de trabajo futuro 69

Bibliografía 70

A. Código programado en *Matlab* 71

A.1. *Script* para lectura y envío de datos del actuador 72
A.2. Código de las funciones empleadas en el modelo híbrido del manipulador . 83
A.2.1. Matriz de transferencia 83
A.2.2. Jacobiano 84
A.2.3. Interacción con el entorno 84

B. Cálculo de la constante de elasticidad del muelle 85

C. Validación y reajuste de parámetros para un modelado dinámico tradicional del actuador 87

D. Plano del brazo del servomotor 95

E. Fundamentos del modelo dinámico del manipulador basado en el principio de Gibbs–Appell 97

E.1. Desarrollo matemático 98
E.2. Código en *Matlab* para el cálculo de I_F^C 101
E.2.1. Código *fcn_sigma* 106
E.2.2. Código *fcn_psi* 106
E.2.3. Código *fcn_U* 106
E.2.4. Código *fcn_xi* 107
E.2.5. Código *fcn_gamma* 107
E.2.6. Código *fcn_jacob* 107

Acrónimos

PWM Pulse-Width Modulation(Modulación por Ancho de Pulsos)

RPM Revolutions per Minute(Revoluciones por Minuto)

GDL Grados de Libertad

pHRI Physical Human-Robot Interaction(Interacción Física Humano-Robot)

Esta pagina ha sido dejada intencionadamente en blanco

Glosario

Modelado del actuador

$\theta_m, \dot{\theta}_m, \ddot{\theta}_m$**:** Ángulo, velocidad y aceleración del eje del motor

$\theta_l, \dot{\theta}_l, \ddot{\theta}_l$**:** Ángulo, velocidad y aceleración del eje de carga del motor

u**:** Voltaje entre los terminales de entrada

e**:** Fuerza contraelectromotriz (EMF)

i**:** Intensidad del circuito

R**:** Resistencia de armadura del circuito del motor

L**:** Inductancia de armadura del circuito del motor

N**:** Relación de transmisión de los engranajes del motor

K_t**:** Constante de par del motor

K_ω**:** Constante de velocidad del motor

J_m**:** Momento de inercia del conjunto rotor y engranajes visto en el eje del motor

J_l**:** Momento de inercia de la carga conectada al eje de carga del motor

J_{me}**:** Momento de inercia del conjunto rotor, engranajes y disco de conexión de carga, visto en el eje de carga

b_m**:** Coeficiente de fricción del conjunto rotor y engranajes visto en el eje del motor

b_l**:** Coeficiente de fricción de la carga conectada al eje de carga del motor

b_{me}**:** Coeficiente de fricción del conjunto rotor y engranajes, visto en el eje de carga

τ_m**:** Torque producido por el motor

τ_l**:** Torque transmitido a la carga

τ_{fm}**:** Torque de fricción en el eje motor

τ_{fl}**:** Torque de fricción en el eje de carga

η**:** Rendimiento del motor

Modelado del manipulador

I_F^O**:** Matriz de inercia del sistema en fase de vuelo para mecanismo de cadena cinemática abierta

I_F^C**:** Matriz de inercia del sistema en fase de vuelo para mecanismo de cadena cinemática cerrada

$\ddot{\theta}_F^O$**:** Vector de aceleraciones generalizadas en fase de vuelo para mecanismo de cadena cinemática abierta

$\ddot{\theta}_F^C$**:** Vector de aceleraciones generalizadas en fase de vuelo para mecanismo de cadena cinemática cerrada

R_F^O**:** Términos dinámicos restantes (Coriolis y fuerzas centrífugas en fase de vuelo para mecanismo de cadena cinemática abierta

R_F^O**:** Términos dinámicos restantes (Coriolis y fuerzas centrífugas en fase de vuelo para mecanismo de cadena cinemática cerrada

${}^i\mathbf{j}_{j,:}^T$**:** Vector que contiene la j-ésima fila y cada columna de la matriz jacobiana de la i-ésima articulación

f_{X_i}**:** Fuerza de restricción i

${}^{ref}\mathbf{r}_{O_i}$**:** Vector posición de la articulación i-ésima descrito en el sistema de referencia inercial

M_{tot}**:** Masa total del manipulador

$\mu_i(\eta)$**:** Masa por unidad de longitud del eslabón i-ésimo

$J_i(\eta)$**:** Momento de inercia por unidad de longitud del eslabón i-ésimo

${}^i\vec{\mathbf{r}}_{Q/O_i}$**:** Posición de un elemento diferencial Q con respecto al sistema de coordenadas local del eslabón i-ésimo

${}^i\tilde{\mathbf{r}}_{Q/O_i}$**:** Matriz antisimétrica relacionada con el vector ${}^i\vec{\mathrm{r}}_{Q/O_i}$

${}^iR_k$**:** Matriz de tamaño 3×3 que muestra la orientación del sistema de referencia local del cuerpo k-ésimo con respecto al del cuerpo i-ésimo.

${}^i\vec{\mathbf{r}}_{O_i/O_j}$**:** Posición de un vector O_i con respecto al vector O_j, en el sistema de coordenadas local del eslabón i-ésimo

$x_{i,1}x_{2,1}x_{3,1}$**:** Sistema de referencia local del eslabón i-ésimo, cuyo origen se posiciona al comienzo del propio eslabón. $x_{i,1}$ va a lo largo del eslabón i-ésimo, desde Oi a $Oi+1$, y $x_{i,3}$ se coloca en la articulación sobre la que rota el eslabón i-ésimo respecto al anterior (i-1)

${}^{ref}X_1^{ref}X_2^{ref}X_3$**:** Sistema de referencia inercial situado en el suelo

X_1, X_2, X_3**:** Movimientos translacionales del sistema de coordenadas local del primer eslabón en las direcciones del sistema de referencia inercial del suelo

${}^i\vec{\omega}_i$**:** Velocidad angular del eslabón i-ésimo

θ_j**:** Orientación del eslabón j-ésimo

l_i**:** Longitud del eslabón i-ésimo

Esta pagina ha sido dejada intencionadamente en blanco

Índice de figuras

1.1. Garra subactuada cuyo objetivo es medir la fuerza que ejerce un objeto (un antebrazo en el caso de la imagen) en un plano frontal utilizando únicamente la información propioceptiva de los servomotores y de las articulaciones pasivas ([1]) 26

1.2. Esquema de la metodología de este trabajo 28

2.1. Servomotor Dynamixel *XM430-W210-R* 32

2.2. Montaje para alimentar y controlar los motores Dynamixel 34

2.3. Interfaz del software *R+ Manager 2.0* 34

2.4. Circuito eléctrico equivalente de un motor DC 35

2.5. Relación PWM - Voltaje 38

2.6. Modelo del servomotor en *Simulink* para obtener su velocidad en el eje de carga a partir del voltaje PWM de entrada 39

2.7. Sistema en *Simulink* que obtiene los datos de funcionamiento reales del motor y los compara con los obtenidos por el modelo para una señal de voltaje PWM determinada, en este caso de 500 niveles 39

2.8. Modelo del servomotor en *Simulink* para obtener su posición en el eje de carga a partir del voltaje PWM de entrada, con realimentación de par externo 40

2.9. Relación Posición angular - Par del muelle 40

2.10. Montaje elaborado para los experimentos y la obtención de un par externo generado por el muelle sobre el motor en función de la posición angular de este 41

2.11. Respuesta inicial en posición (grados) del motor real (en azul) frente a la respuesta obtenida con el modelo (en naranja), ante una entrada PWM de 8 niveles 42

2.12. Respuesta inicial en posición (grados) del motor real (en azul) frente a la respuesta obtenida con el modelo (en naranja), ante una entrada PWM de 16 niveles 43

2.13. Respuesta inicial en posición (grados) del motor real (en azul) frente a la respuesta obtenida con el modelo (en naranja), ante una entrada PWM de 40 niveles 43

2.14. Entrada PWM de tipo rampa 44

2.15. Respuesta inicial en posición (grados) del motor real (en azul) frente a la respuesta obtenida con el modelo (en naranja), ante una entrada PWM de tipo rampa 44

2.16. Comparativa de las respuestas entre modelo y motor para diferentes entradas antes del ajuste de los parámetros 45

2.17. Comparativa de las respuestas entre modelo y motor para diferentes entradas después del ajuste de los parámetros (izquierda), y evolución de los valores de estos (derecha) 46

2.18. Respuesta del modelo (en naranja) frente a la del motor (en azul) ante diferentes entradas, tras el reajuste 47

2.19. Entrada PWM de tipo secuencia de impulsos 48

2.20. Respuesta en posición (grados) del motor real (en azul) frente a la respuesta obtenida con el modelo (en naranja), ante una entrada PWM de tipo secuencia de impulsos 48

2.21. Modelo del servomotor en *Simulink* modificado para introducir la fricción estática del motor 49

2.22. Respuesta en posición (grados) del motor real (en azul) frente a la respuesta obtenida con el modelo (en naranja), ante una entrada PWM de tipo secuencia de impulsos y con la fricción estática modelada 49

3.1. Modelo 3D del manipulador, con la arquitectura de un dedo en detalle (Fig. 3.1b) 52

3.2. Mecanismo de cinco barras que constituye un dedo de la garra subactuada 53

3.3. Esquema del modelo híbrido implementado 56

3.4. Modelo simplificado de un objeto determinado para poder simular la interacción física entre este y las falanges del manipulador mediante el modelo estático desarrollado 57

3.5. Manipulador de dos falanges y cuatro barras, modelo para el cual calcular la matriz de transferencia ([10]) 58

3.6. Zona inferior del manipulador de cinco barras 59

3.7. Bloque *Matriz de transferencia* 60

3.8. Bloque *Jacobiano* 61

3.9. Interacción de la garra con la esfera 62

3.10. Bloque *Interacción con el entorno* 63

3.11. Modelo híbrido implementado en *Simulink* 63

B.1. Relación Estiramiento-Fuerza 86

C.1. Respuesta en velocidad (rad/s) del motor real (en azul) frente a la respuesta obtenida con el modelo (en naranja) ante una entrada PWM de 200 niveles 88

C.2. Respuesta en velocidad (rad/s) del motor real (en azul) frente a la respuesta obtenida con el modelo (en naranja) ante una entrada PWM de 500 niveles 89

C.3. Respuesta en velocidad (rad/s) del motor real (en azul) frente a la respuesta obtenida con el modelo (en naranja) ante una entrada PWM de 700 niveles 89

C.4. Estimación de parámetros obtenida con la función de costes basada en la mínima suma del error cuadrático 90

C.5. Estimación de parámetros obtenida con la función de costes basada en la mínima suma del error absoluto 91

C.6. Respuesta en velocidad (rad/s) del motor real (en azul) frente a la respuesta obtenida con el modelo (en naranja) ante una entrada PWM de 200 niveles, tras el reajuste 92

C.7. Respuesta en velocidad (rad/s) del motor real (en azul) frente a la respuesta obtenida con el modelo (en naranja) ante una entrada PWM de 500 niveles, tras el reajuste 92

C.8. Respuesta en velocidad (rad/s) del motor real (en azul) frente a la respuesta obtenida con el modelo (en naranja) ante una entrada PWM de 700 niveles, tras el reajuste 93

Esta pagina ha sido dejada intencionadamente en blanco

Índice de tablas

2.1. Especificaciones hardware del motor a tener en cuenta 33
2.2. Valores originales de los parámetros del motor 42
2.3. Valores finales de los parámetros del motor 46
2.4. Error relativo cometido en régimen permanente antes y después del reajuste 47

B.1. Mediciones para el cálculo de la constante de elasticidad del muelle (K_m) . 86

C.1. Valores originales de los parámetros del motor 88
C.2. Error de la respuesta del modelo antes del reajuste de parámetros 90
C.3. Error, después del reajuste, de la respuesta del modelo 91
C.4. Valores reajustados de los parámetros del motor 91

Esta pagina ha sido dejada intencionadamente en blanco

Sección 1

Introducción

1.1. Objeto

El objeto de este trabajo de fin de máster (TFM en adelante) es el desarrollo del modelo dinámico de una garra subactuada mediante servomotores para la interacción física humano-robot (pHRI).

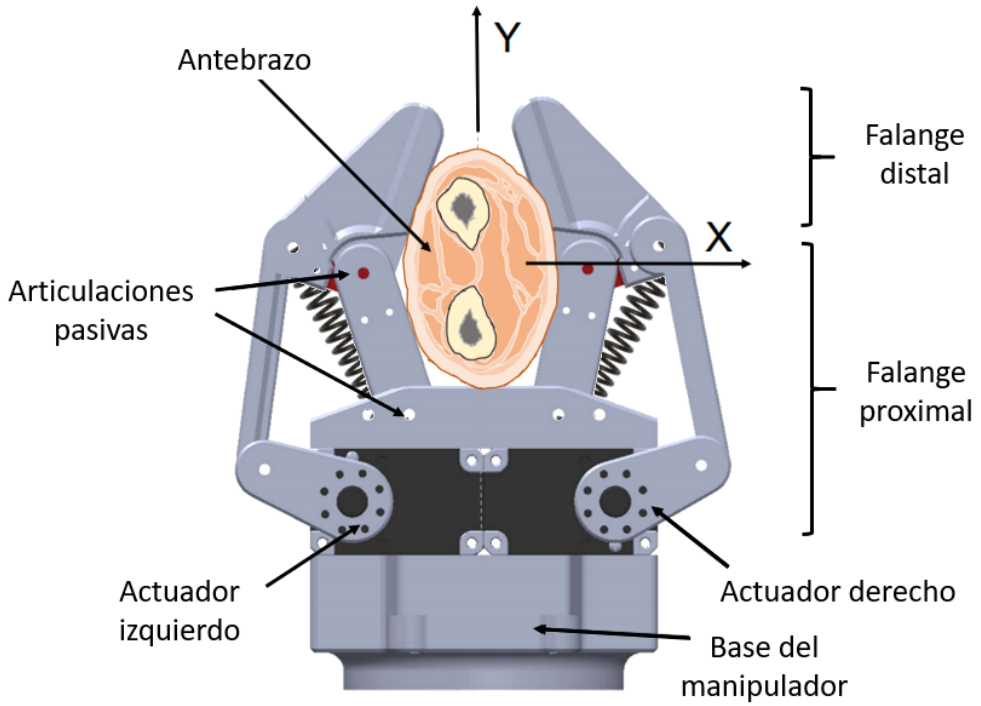


Figura 1.1: Garra subactuada cuyo objetivo es medir la fuerza que ejerce un objeto (un antebrazo en el caso de la imagen) en un plano frontal utilizando únicamente la información propioceptiva de los servomotores y de las articulaciones pasivas ([1])

Este manipulador se observa en la Figura 1.1. Se puede comprobar que dispone de dos actuadores, uno por cada dedo, y que representan las articulaciones activas del manipulador. Cada dedo presentan una arquitectura de cinco barras, formando dos falanges, denominadas proximal y distal. A lo largo de este proyecto, se va a trabajar con un único dedo por cuestiones de simplicidad. En lo que respecta al actuador empleado, este es un servomotor Dynamixel *XM430-W210-R*.

En una primera fase del proyecto se desarrolla el modelo dinámico del actuador, para lo cual se establecen unas condiciones de funcionamiento deseadas, se obtienen y se refinan sus parámetros mediante una serie de experimentos con diferentes entradas. Posteriormente, debido a la naturaleza del manipulador y de sus elementos se considera despreciable su carácter dinámico y se evalúa directamente su modelo estático, el cual se implementa en conjunto con el modelo dinámico del servomotor, creando un modelo híbrido del manipulador. El objetivo final es obtener un modelo para la garra subactuada que se caracterice por su precisión y fiabilidad, permitiendo así realizar simulaciones y estimaciones precisas de la fuerza que ejercerá la garra sobre un objeto determinado, sin necesidad de emplear para ello sensores de medición de fuerzas.

1.2. Antecedentes

La robótica subactuada es un campo emergente dentro del entorno de la robótica, puesto que su estudio se centra en el desarrollo de sistemas de control que aprovechen la dinámica natural de las máquinas, de forma que éstas tengan un mayor número de grados de libertad que de actuadores. Este hecho conlleva numerosas ventajas, entre las cuales se encuentran una mayor ligereza del sistema, un menor consumo de energía y un mejor rendimiento. Una parte importante del campo de la subactuación consiste en pinzas o garras subactuadas, las cuales comúnmente tienen el objetivo de adaptarse automáticamente a la forma del objeto que están agarrando. Este diseño adaptativo que proporciona la robótica subactuada es esencial para obtener una interacción física segura entre humano y robot (pHRI), un terreno en auge en los últimos años debido al aumento de la interacción entre humanos y máquinas en prácticamente todas las áreas ([2], [3]).

En lo relativo a los servomotores, las primeras patentes se remontan a comienzos del siglo XX ([4]). Desde entonces, sus prestaciones han mejorado hasta convertirse en un elemento indispensable, no solo en multitud de mecanismos, sino en terrenos tan exigentes como el ensamblaje de automóviles, las máquinas CNC o la industria robótica. Especialmente enfocados a esta última, se han diseñado una serie de servomotores de la gama Dynamixel, de la cual se va a emplear el modelo *XM430-W210-R* para este proyecto. Este tipo de servos destaca por su carácter bidireccional, de tal manera que el usuario no solo tiene la posibilidad de enviar comandos de posición o velocidad, sino que el propio servo aporta una realimentación al usuario en términos de posición, velocidad temperatura, corriente, etc. Estas características convierten los servomotores en la opción ideal para el diseño de una garra subactuada que presente un agarre preciso y adaptativo.

Sin embargo, a pesar del desarrollo experimentado por los servomotores, es complicado encontrar modelos dinámicos adecuados del comportamiento de estos componentes. La ausencia de un modelo preciso compromete el rendimiento del sistema de control, debido a los desajustes entre la planta real y el modelo. Para obtener la máxima precisión y seguridad posibles, es necesario minimizar estos desajustes, es decir, formalizar un correcto modelo dinámico del servo y del sistema, especialmente en tareas que entren dentro del campo pHRI y que por ende precisen de una fiabilidad máxima.

1.3. Metodología del proyecto

El proceso seguido para la realización de este proyecto se observa en forma de esquema en la Figura 1.2, y se puede dividir en los siguientes puntos:

1. Búsqueda de información bibliográfica sobre los temas a tratar: robótica subactuada, modelado dinámico de servomotores, mecanismos de cinemática cerrada, y sobre pHRI.

2. Desarrollo del modelo dinámico del servomotor e interpretación de sus parámetros.

3. Validación del modelo dinámico del servomotor.

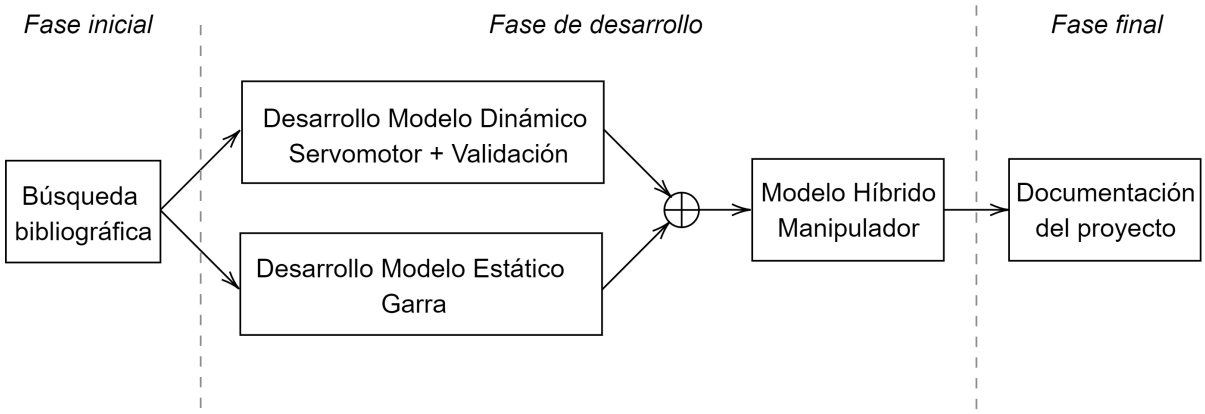


Figura 1.2: Esquema de la metodología de este trabajo

4. Desarrollo del modelo estático del manipulador.
5. Integración de dicho modelo con el modelo dinámico del actuador, creando así un modelo híbrido.
6. Documentación del proyecto.

1.4. Herramientas del proyecto

Las herramientas empleadas para la realización de este trabajo son las siguientes:

- Hardware:
  - Ordenador de sobremesa con:
    - Sistema operativo Windows 10.
    - Tarjeta gráfica MSI GeForce RTX 2060 Ventus OC 6GB GDDR6.
    - Procesador AMD Ryzen 7 2700X 4.3 Ghz.
    - Memoria RAM DDR4 con 16GB.
  - Servomotor Dynamixel *XM430-W210-R*.
  - Fuente de alimentación de 12V, un adaptador *SMPS2Dynamixel* para conectarla con el motor, y un convertidor *U2D2* para conectar el sistema con el PC.
  - Impresora 3D Prusa i3, con plástico PLA.
- Software:
  - Matlab & Simulink v2020.
  - SolidWorks v2018.
  - R+ Motion V2.0.
  - R+ Manager V2.0

1.5. Estructura de la memoria

El proyecto se estructura de la siguiente manera. La Sección 2 presenta el modelado del actuador, para lo cual se desarrolla su modelo dinámico, se realiza una interpretación de sus parámetros y se valida el modelo, reajustando dichos parámetros para incrementar la fiabilidad del modelo. La Sección 3 desarrolla el modelo cinemático y el estático de la garra subactuada, considerando un modelo simplificado de un objeto esférico para simular la interacción de dicho objeto con los dedos del manipulador. Al final de esta sección se implementan de forma conjunto el modelo dinámico del actuador y el estático de la garra. Por último, la Sección 4 presenta las conclusiones del trabajo y las líneas propuestas de trabajo futuro.

Por otra parte, los Anexos muestran información complementaria al proyecto: cálculos, experimentos adicionales y planos. En lo que respecta al código programado en Matlab, este se ha incluido en la memoria en los Anexos A y E, ya que este proyecto no va a ser impreso y por lo tanto no supone un malgasto de recursos. Asimismo, se haya disponible en un repositorio público (https://github.com/TaISLab/Code_dynamixel_model)

Sección 2

Modelado del servomotor Dynamixel

2.1. Introducción

Este capítulo contiene los pasos seguidos para obtener el modelo dinámico del motor Dynamixel *XM430-W210-R*. Se toma como punto de partida el modelo físico de un motor de corriente continua clásico, a partir del cual se realizan una serie de experimentos, definiendo para ello las condiciones de funcionamiento el motor en este proyecto. Con el fin de optimizar el modelo, se realiza un reajuste de los valores de los parámetros del motor mediante el software de *Simulink*. Finalmente, se obtiene la expresión del par en el eje del motor en función del voltaje PWM para diferentes tipos de entradas.

2.1.1. Características del motor

(a) Reductora con engranajes metálicos (b) Motor DC y electrónica

Figura 2.1: Servomotor Dynamixel *XM430-W210-R*

El servomotor elegido para este trabajo es el denominado *XM430-W210-R*, de la compañía Dynamixel, y puede observarse su interior, tanto su parte mecánica como su parte electrónica y su motor de corriente continua, en la Figura 2.1. Pertenece a una de las series más recientes producidas por la marca (la Serie-X), e incorpora algunas de las últimas prestaciones ofrecidas por Dynamixel:

- Admite 6 modos de operación con una arquitectura de control sofisticada: control de torque, control de velocidad, control de posición, control extendido de posición, control de posición basado en corriente, y control PWM.
- Estructura mecánica eficiente, lo que aumenta su rendimiento y le permite tener un tamaño reducido, mejorando así el diseño del manipulador en el que va a ser utilizado.
- Es el actuador de la Serie-X que ofrece mayor velocidad (RPM) sin carga aplicada.

- Amplia retroalimentación, que permite conocer en todo momento datos del motor como su posición, velocidad, corriente, trayectoria, temperatura y voltaje de entrada.

Tabla 2.1: Especificaciones hardware del motor a tener en cuenta

Parámetro	**Valor**
Relación de engranajes (N)	212.6
Tensión de alimentación (V_s)	12 V
Stall torque o par en parada (τ_{Ls})	2.7 Nm (2.3 A)
Velocidad máxima sin carga ($\omega_{max,nl}$)	77 rpm
Salida máxima PWM	885 niveles

Estas características han sido determinantes para elegir este modelo en cuestión. En lo que respecta a las especificaciones hardware, éstas se encuentran detalladas en la bibliografía ([5]). Sin embargo, en la Tabla 2.1 se muestran algunos de sus parámetros más destacables.

2.1.2. Configuración del equipo

Para trabajar con este motor se han utilizado los siguientes dispositivos adicionales:

- U2D2: convertidor de comunicación que permite controlar y operar los motores Dynamixel a través del PC. En este caso, se emplea su conector de 4 pines para la comunicación de tipo RS-485. Este dispositivo no proporciona fuente de voltaje al motor, por lo que es necesario emplear una fuente externa.
- SMPS2Dynamixel: elemento que funciona como intermediario entre los motores y la fuente de alimentación, con el fin de estabilizar el voltaje de entrada procedente de ésta.
- Fuente de alimentación: adaptador AC/DC que proporciona a su salida un voltaje de 12 V y una intensidad de 6 A.

En la Figura 2.2 se observa como se deben conectar dos motores en serie con los dispositivos mencionados previamente. Antes de probar el funcionamiento de los motores, es necesario realizar una configuración inicial, para la cual se emplea el software *R+ Manager 2.0*, proporcionado por el fabricante y cuya interfaz se muestra en la Figura 2.3. Con este programa, se asigna un identificador distinto a cada motor, se establece el parámetro conocido como *Baud rate* o tasa de baudios con el valor de 59600 bps, y se selecciona el modo de operación de control mediante PWM. De esta forma, la velocidad y el par de salida dependen del voltaje PWM de entrada, y la configuración inicial de los motores queda finalizada.

En caso de utilizar un único motor, tal y como ocurre en los experimentos realizados en este proyecto, el proceso de configuración y conexión es el mismo, asignando al motor el identificador $ID = 1$, y conectándolo entre el SMPS2Dynamixel y el dispositivo U2D2.

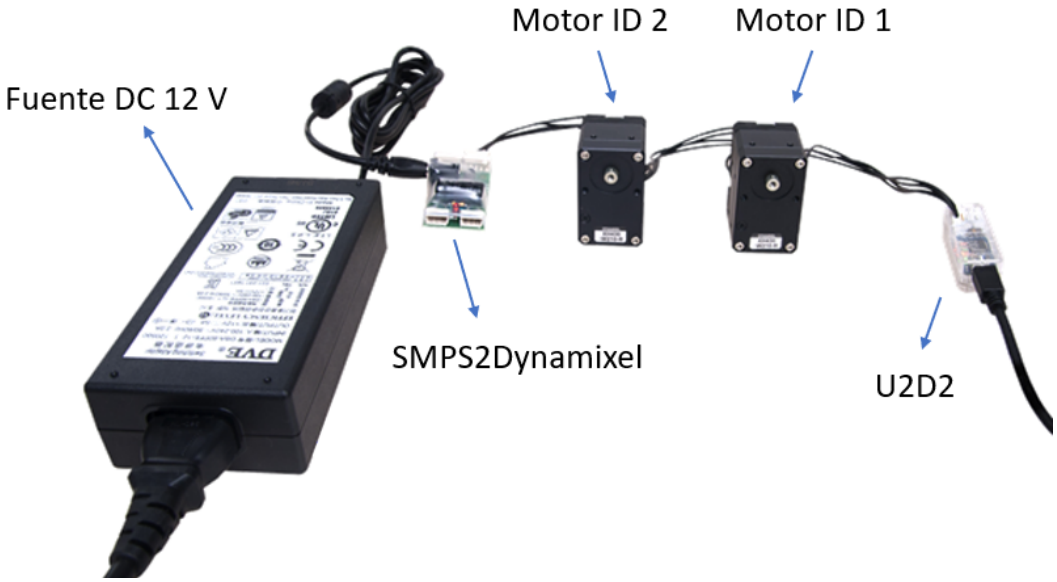


Figura 2.2: Montaje para alimentar y controlar los motores Dynamixel

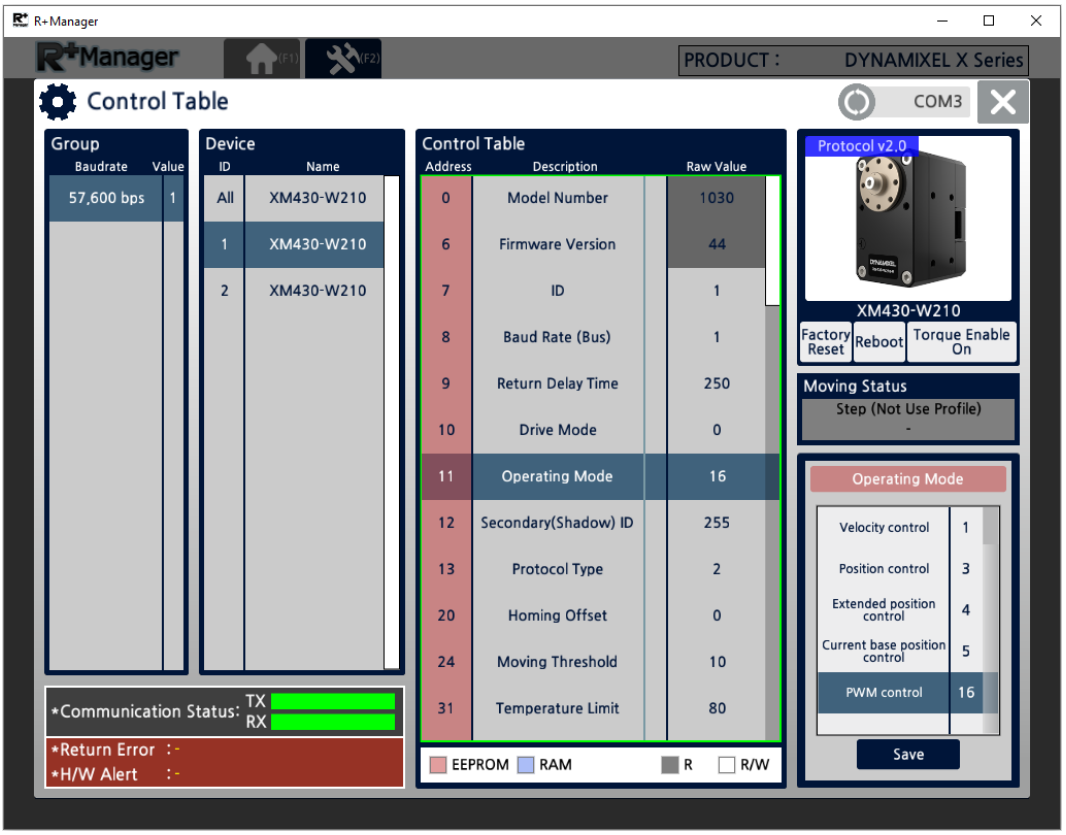


Figura 2.3: Interfaz del software *R+ Manager 2.0*

2.2. Modelado dinámico del motor

El modelo dinámico del motor se ha obtenido en base a los principios físicos que rigen un motor de corriente continua. Por lo tanto, se comienza evaluando las ecuaciones derivadas de su circuito eléctrico equivalente (Fig. 2.4), donde u es el voltaje entre los terminales de entrada, e es la fuerza contra-electromotriz, i es la corriente del circuito, y L y R son la inductancia y la resistencia de armadura, respectivamente.

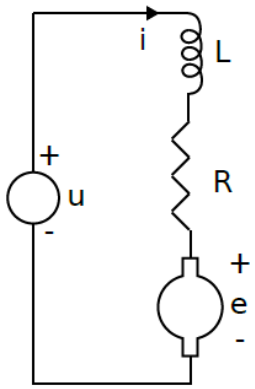


Figura 2.4: Circuito eléctrico equivalente de un motor DC

Además de estos, se deben tener en cuenta otros parámetros para poder trabajar con las ecuaciones del modelo del motor. Por un lado, se asume que este tiene unas constantes de torque y de velocidad, K_t y K_w. El ángulo de rotación del eje del motor es θ_m, su momento de inercia es J_m y su coeficiente de fricción es b_m. La notación en el caso del eje de carga es similar, cambiando únicamente los subíndices: θ_l, J_l, y b_l. Por otra parte, τ_m es el torque producido por el motor, τ_l es el que se transmite a la carga, y τ_{fm} yτ_{fl} son los torques de fricción en el motor y en la carga, respectivamente. Finalmente, N es la relación de transmisión de los engranajes del motor, y η es el rendimiento de este.

Las ecuaciones que se obtienen a partir del circuito eléctrico del motor ([6]) son las siguientes:

$$u(t) = Ri(t) + L\frac{di(t)}{dt} + e(t) \tag{2.1}$$

$$\tau_m(t) = K_t i(t) \tag{2.2}$$

$$e(t) = \omega_m(t)K_t = N\omega_l(t)K_t \tag{2.3}$$

$$J_l\frac{d\omega_l(t)}{dt} = \tau_l(t) - \tau_{fl}(t) \tag{2.4}$$

El objetivo del siguiente desarrollo matemático es obtener, a partir de las expresiones anteriores, una función de transferencia que permita obtener la velocidad en el eje de carga (Ω_l) en función del voltaje de entrada y de los parámetros del motor. En los

siguientes cálculos, no se utiliza ninguna carga, por lo que tanto la inercia J_l como la fricción b_l son nulas. Sin embargo, sí deben tenerse en cuenta tanto el momento de inercia como la fricción del rotor, del conjunto de engranajes y del disco de carga, vistos desde el eje de carga. Estos se denotan como J_{me} y b_{me}, respectivamente, y sus expresiones son las siguientes:

$$J_{me} = J_m N^2 \eta \tag{2.5}$$

$$b_{me} = b_m N^2 \eta \tag{2.6}$$

A partir de este momento, las variable J y b van a denotar el conjunto de todas las inercias y fricciones vistas desde el eje de carga, respectivamente, de forma que:

$$J = J_l + J_{me} \tag{2.7}$$

$$b = b_l + b_{me} \tag{2.8}$$

Por lo tanto, la Ecuación 2.4 quedaría tal que:

$$J\frac{d\omega_l(t)}{dt} = \tau_l(t) - \tau_{fl}(t) \tag{2.9}$$

Siendo τ_{fl} el torque de fricción en la carga, el cual se define como:

$$\tau_{fl}(t) = b\omega_l(t) \tag{2.10}$$

Con lo cual, la Ecuación 2.9 se podría reescribir como:

$$J\frac{d\omega_l(t)}{dt} = \tau_l(t) - b\omega_l(t) \tag{2.11}$$

Despejando el torque en el eje de carga (τ_l), y aplicando la transformada de Laplace a la expresión resultante se obtiene:

$$Js\Omega_l(s) + b\Omega_l(s) = \tau_l(s) \tag{2.12}$$

Asimismo, se aplica la transformada de Laplace a las Ecuaciones 2.1, 2.2 y 2.3.

$$U(s) = RI(s) + LsI(s) + E(s) \tag{2.13}$$

$$\tau_m(s) = K_t I(s) \tag{2.14}$$

$$E(s) = N\Omega_l(s)K_t \tag{2.15}$$

A partir de estas expresiones, el primer paso es despejar la intensidad $I(s)$ de la Ecuación 2.13, sustituyendo además la variable $E(s)$ por su expresión (Ec. 2.15):

$$U(s) - E(s) = I(s)(Ls + R) \Rightarrow I(s) = \frac{U(s) - NK_t\Omega_l(s)}{Ls + R} \tag{2.16}$$

Esta expresión se introduce en la Ecuación 2.14, obteniendo:

$$\tau_m(s) = K_t \frac{U(s) - NK_t\Omega_l(s)}{Ls + R} \tag{2.17}$$

Finalmente, para operar conjuntamente las Ecuaciones 2.12 y 2.17 es necesario convertir el torque en el eje de carga τ_l al torque visto desde el eje del motor, τ_m, siendo la relación entre ambos la siguiente:

$$\tau_l(s) = N\tau_m(s) \tag{2.18}$$

Por lo tanto, la Ecuación 2.12 quedaría tal que:

$$Js\Omega_l(s) + b\Omega_l(s) = N\tau_m(s) \Rightarrow \Omega_l(s)(Js + b) = N\tau_m(s) \tag{2.19}$$

Y sustituyendo el torque τ_m por su expresión de la Ecuación 2.17, tendríamos la siguiente igualdad, de la que se busca obtener la salida de velocidad $\Omega_l(s)$ en función de la entrada de voltaje $U(s)$.

$$\begin{aligned}
\Omega_l(s)(Js + b) = NK_t \frac{U(s) - NK_t\Omega_l(s)}{Ls + R} \Rightarrow \\
\Rightarrow \Omega_l(s)(Js + b)(Ls + R) + N^2K_t^2\Omega_l(s) = NK_tU(s) \Rightarrow \\
\Rightarrow \Omega_l(s)((Js + b)(Ls + R) + N^2K_t^2) = NK_tU(s) \Rightarrow \\
\Rightarrow \frac{\Omega_l(s)}{U(s)} = \frac{NK_t}{(Js + b)(Ls + R) + N^2K_t^2}
\end{aligned} \tag{2.20}$$

Esta sería la función de transferencia del motor DC que permite obtener la velocidad angular en el eje de carga en función del voltaje de entrada U.

Debido a que se ha hallado un modelo dinámico elaborado en la literatura consultada ([7]) para el mismo motor que se utiliza en este trabajo, se han tomado inicialmente los valores obtenidos por el autor en su investigación, a partir de los cuales se ha realizado una primera aproximación del modelo en el Anexo C. En él, se ha llevado a cabo un reajuste de estos valores para compensar cualquier posible disparidad entre los parámetros de un motor y otro. El único parámetro cuyo valor se ha podido comprobar empíricamente es el de la resistencia de armadura (R), una medida directa con un multímetro utilizando los terminales conectados al motor DC.

Por otra parte, se ha realizado un mapeado de voltaje en función de un comando PWM, desde -100 hasta 100 niveles. Se ha establecido este voltaje como límite del mapeado ya que en este proyecto se va a trabajar a bajas velocidades, con el fin de controlar la fuerza ejercida por la garra. En la Figura 2.5 se observa la recta obtenida, cuya interpolación lineal está definida por la siguiente ecuación:

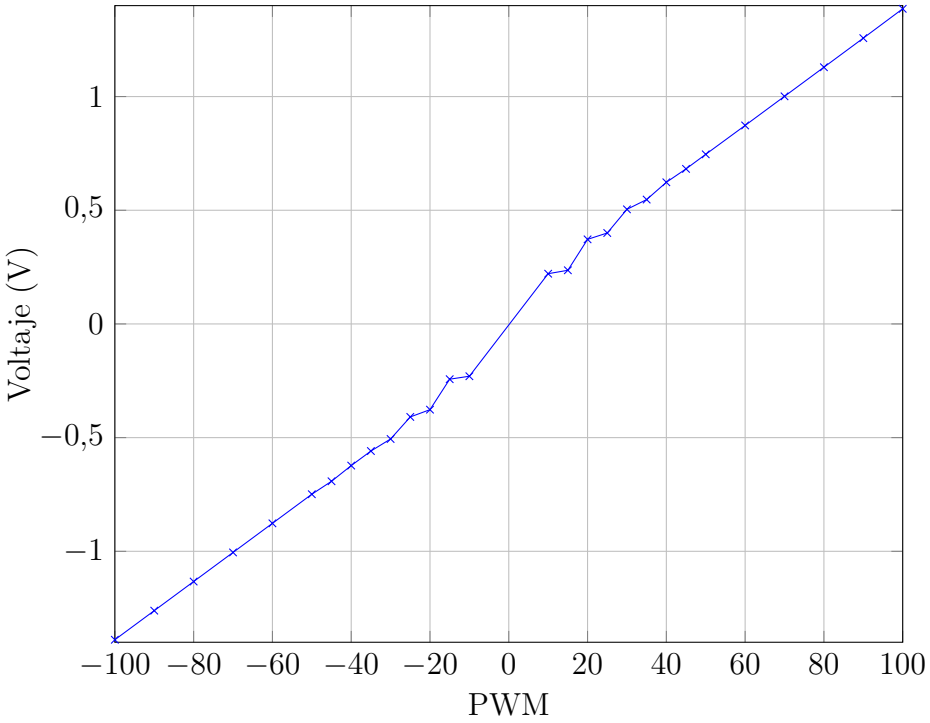


Figura 2.5: Relación PWM - Voltaje

$$u = 0{,}0145 \cdot u_{PWM} - 0{,}0027 \tag{2.21}$$

Donde se obtiene la tensión entre los terminales del motor (u) en función de la amplitud o voltaje de la entrada PWM (u_{PWM}).

Aunque no se va a entrar en detalle, los experimentos realizados en dicho artículo para la obtención del resto de parámetros de la Tabla C.1 son los siguientes:

- Pruebas dinámicas en diferentes condiciones de carga para estudiar la respuesta del motor en régimen transitorio y permanente.
- Pruebas de par estático en el eje en el motor mediante la aplicación una entrada PWM mientras se desplaza el eje de carga acoplado a un muelle.

Con las ecuaciones del modelo ya desarrolladas y los parámetros del motor calculados, es posible implementar el modelo en *Simulink* con el objetivo de validarlo.

2.3. Implementación del modelo en *Simulink*

La entrada del modelo es un comando PWM, al que se le dará diversas formas en los sucesivos experimentos. En este apartado, se emplea una señal escalón de amplitud variable, tras la cual se ha añadido una saturación que limita la entra PWM entre ±885 niveles, ya que este es el rango de voltaje que acepta el motor.

El voltaje de la entrada PWM (u_{PWM}) ya filtrado se introduce en la Ecuación 2.21 para obtener la tensión que cae entre los terminales del motor, U. Como consecuencia

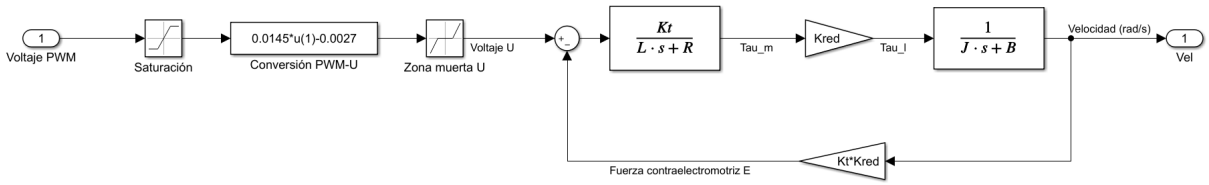


Figura 2.6: Modelo del servomotor en *Simulink* para obtener su velocidad en el eje de carga a partir del voltaje PWM de entrada

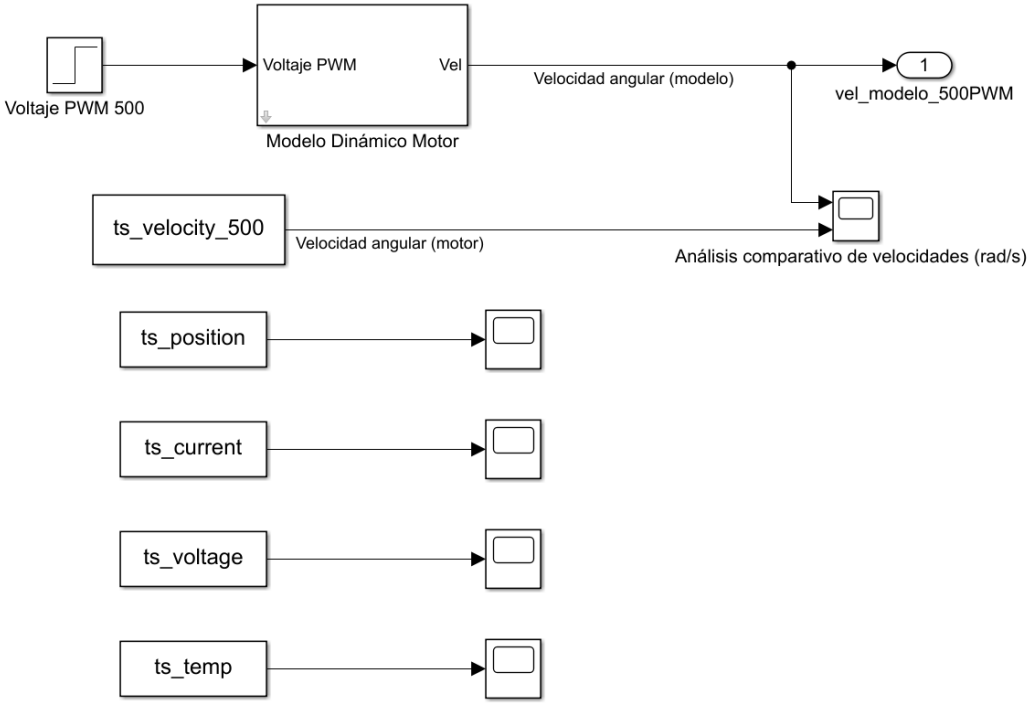


Figura 2.7: Sistema en *Simulink* que obtiene los datos de funcionamiento reales del motor y los compara con los obtenidos por el modelo para una señal de voltaje PWM determinada, en este caso de 500 niveles

de posibles errores en el mapeado de voltaje, éste presenta un pequeño valor para una entrada PWM nula, por lo que es necesario saturarlo. A continuación, se implementa la función de transferencia desarrollada en la Ecuación 2.20, con el propósito de obtener la velocidad en el eje de carga del motor a partir del voltaje de entrada, de los parámetros del motor tomados del artículo, y de la realimentación de la fuerza electromotriz, tal y como se observa en la Figura 2.6. Este modelo se implementa en un bloque denominado *Modelo Dinámico Motor* (Fig. 2.7). Se observan también las entradas de datos obtenidas al ejecutar el script *Lectura_de_datos_motor* de *Matlab*, y que aportan información (posición, voltaje, corriente y temperatura) sobre la respuesta y estado del motor ante una entrada PWM determinada.

En lo que respecta a los parámetros del motor, estos se introducen como variables cuyo valor se define en el *Model Workspace* del modelo. Inicialmente, se van a emplear los que se muestran en la Tabla C.1.

Aunque este modelo serviría para realizar el modelado dinámico tradicional de un motor de corriente continua, este proyecto presenta algunas características que requieren

un modelado diferente en ciertos aspectos. Por una parte, se va a trabajar a velocidades muy bajas, con el objetivo de controlar la fuerza ejercida por las falanges sobre un determinado cuerpo. Y por otro lado, es necesario tener presente un par externo, provocado por el propio objeto agarrado por las falanges. Dicho par se va a modelar empleando un muelle de constante elástica conocida, K_m, la cual se ha obtenido mediante el método estático en el Anexo B.

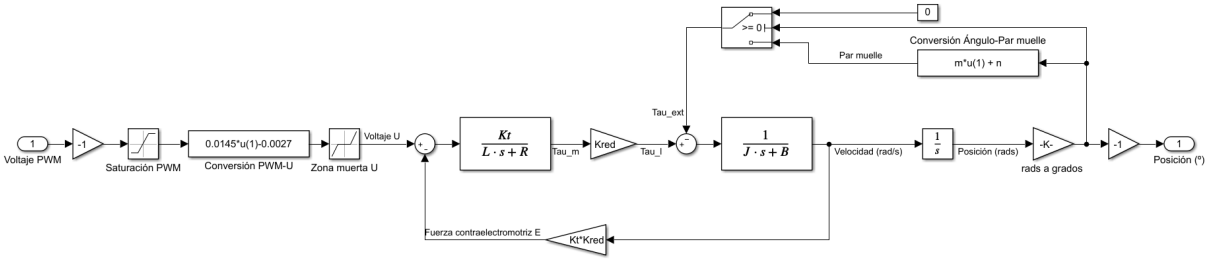


Figura 2.8: Modelo del servomotor en *Simulink* para obtener su posición en el eje de carga a partir del voltaje PWM de entrada, con realimentación de par externo

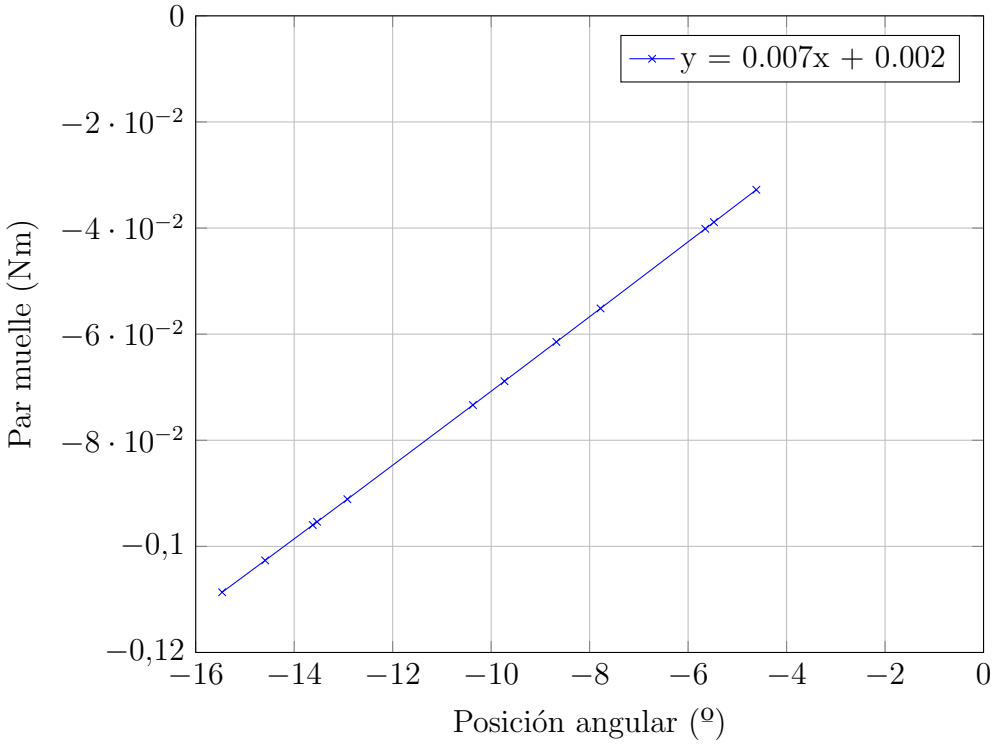


Figura 2.9: Relación Posición angular - Par del muelle

Por este motivo, aunque primeramente se ha ajustado el motor mediante el procedimiento tradicional (desarrollado en el Anexo C), se ha mejorado dicha calibración con un estudio del comportamiento del motor bajo las condiciones de par externo y de régimen de funcionamiento que requieren este proyecto. Para ello, el modelo dinámico (Fig. 2.8) incluye como realimentación el par ejercido por el muelle, y que se calcula en función de la posición angular del motor. Esta función se ha obtenido mediante la interpolación lineal de la curva mostrada en la Figura 2.9, que se ha creado a partir de las mediciones de la deformación del muelle para distintas posiciones angulares dentro de un rango muy reducido, ya que se trata de una aproximación para desplazamientos pequeños.

2.4. Experimentación y Validación del modelo

Para poder evaluar la precisión del modelo dinámico construido en *Simulink*, es necesario comparar la respuesta teórica obtenida con una respuesta real ante la misma entrada PWM. Para ello, se ha empleado el kit de desarrollo de software *Dynamixel SDK* en su implementación en *Matlab*. Aunque el kit ya trae un *script* para poner en funcionamiento el motor y leer la posición angular de su eje, se han realizado algunas modificaciones necesarias para adaptarlo al modelo *XM430-W210-R* en particular. Principalmente, se han introducido las direcciones de la tabla de control necesarias para la lectura y el envío de datos, y varias líneas de código han sido añadidas o modificadas con el fin de obtener una mayor realimentación del motor: datos de posición, velocidad, voltaje, corriente y temperatura. En el Anexo A se encuentra el código completo empleado en este apartado.

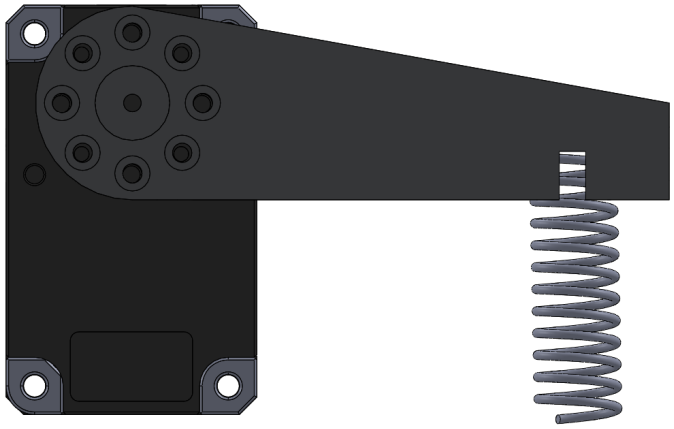


Figura 2.10: Montaje elaborado para los experimentos y la obtención de un par externo generado por el muelle sobre el motor en función de la posición angular de este

Tanto para la toma de los datos de deformación del muelle en función de la posición angular del motor como para los experimentos descritos en esta sección se ha empleado el montaje de la Figura 2.10. Este consiste en un brazo impreso en 3D (plano incluido en el Anexo D), con una masa de 7gr y sujeto al servomotor por un extremo mediante tornillos. En el otro extremo dispone de una apertura para introducir un muelle de 9mm de diámetro y 28mm de longitud, y colocarlo de forma perpendicular al brazo. En la Figura el montaje se encuentra en la posición angular inicial del motor, donde el muelle no está sufriendo deformaciones y el ángulo θ_a es nulo.

Como se quiere estudiar el comportamiento del motor en un régimen de funcionamiento a velocidades pequeñas, el rango de valores de voltaje PWM empleado para los experimentos se encuentra entre 8 niveles, que es el mínimo necesario para que el motor comience a moverse, y 40 niveles, un valor con el cual el muelle escogido se comprime casi por completo. Estas entradas se introducen en el modelo con signo negativo, ya que este es el signo del giro del motor en las agujas del reloj. Asimismo, en este sentido la posición angular obtenida es siempre negativa, aunque en las gráficas se va a mostrar su valor entero por cuestiones de simplicidad. Además, los resultados se encuentran expresados en grados.

Por otra parte, los valores iniciales de los parámetros del motor a reajustar son los que se muestran en la Tabla 2.2, y se han obtenido en el Anexo C a partir del refinamiento

Tabla 2.2: Valores originales de los parámetros del motor

Parámetro	**Valor inicial**
Constante de par (K_t)	0.0065 N/m
Momento de inercia del eje de carga (J)	0.00990708 kgm^2
Coeficiente de fricción del eje de carga (b)	0.01142741 Nms
Pendiente de la recta Posición angular - Par muelle (m)	0.007
Ordenada de la recta Posición angular - Par muelle (n)	0.002

de los parámetros utilizando un modelado tradicional. En lo que respecta al valor de la inductancia de armadura del motor (L), este se ha considerado nulo en todo momento, puesto que es posible despreciar su efecto según la literatura consultada ([8]).

2.4.1. Experimentos con entrada PWM constante

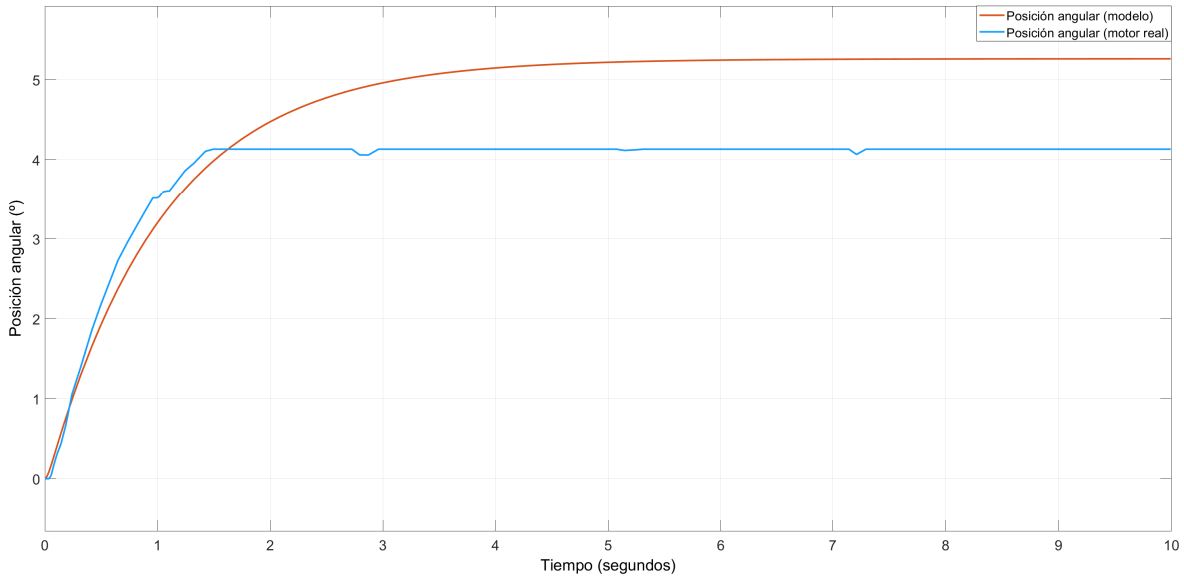


Figura 2.11: Respuesta inicial en posición (grados) del motor real (en azul) frente a la respuesta obtenida con el modelo (en naranja), ante una entrada PWM de 8 niveles

Como punto de partida en el análisis del comportamiento inicial del modelo, se han introducido tres entradas de PWM de amplitud constante. Al estar el rango estudiado de voltaje PWM entre 8 y 40 niveles, se han escogido estos dos extremos y un valor intermedio, 16 niveles. Los resultados obtenidos con los parámetros iniciales para estas tres amplitudes (8,16 y 40) se muestran en las Figuras 2.11, 2.12, y 2.13. Se puede comprobar que la posición angular calculada por el modelo sigue la forma de la que presenta realmente el motor. Aunque este es un indicio de que el modelo está correctamente desarrollado, el error en régimen permanente es elevado en los tres casos, subiendo desde los 0.7 grados para la entrada PWM de 8 niveles, hasta 11.33 grados para la mayor de ellas.

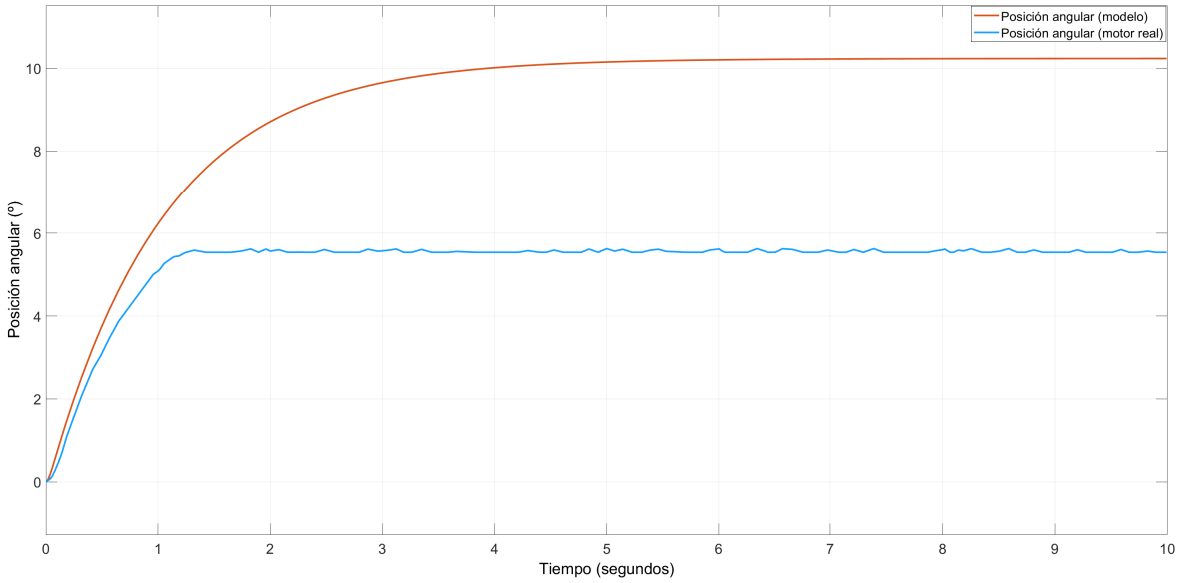


Figura 2.12: Respuesta inicial en posición (grados) del motor real (en azul) frente a la respuesta obtenida con el modelo (en naranja), ante una entrada PWM de 16 niveles

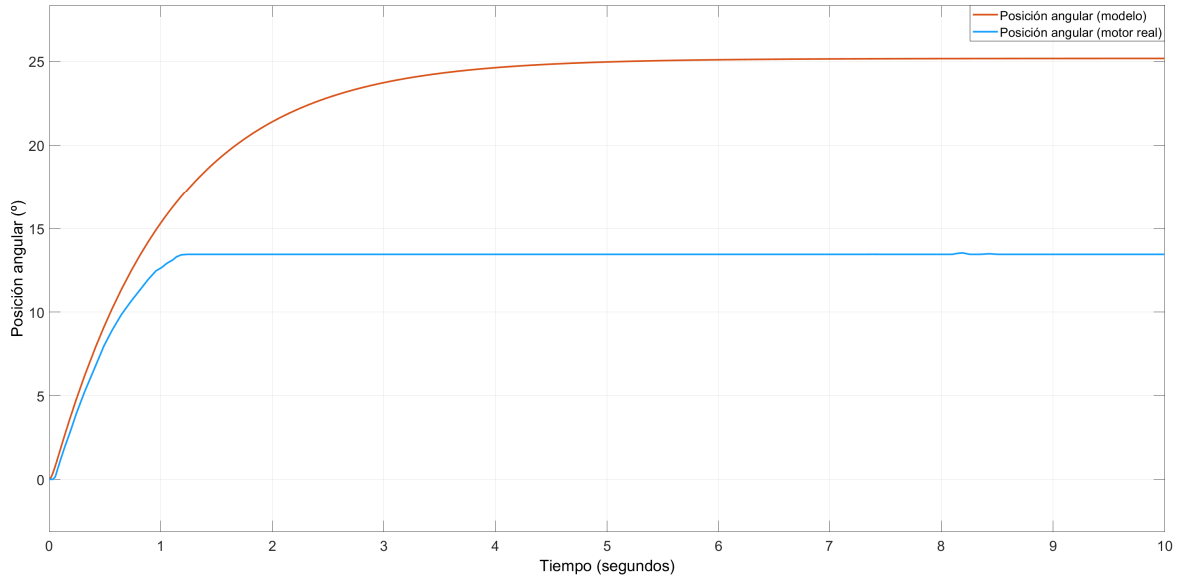


Figura 2.13: Respuesta inicial en posición (grados) del motor real (en azul) frente a la respuesta obtenida con el modelo (en naranja), ante una entrada PWM de 40 niveles

2.4.2. Experimentos con entrada PWM de tipo rampa

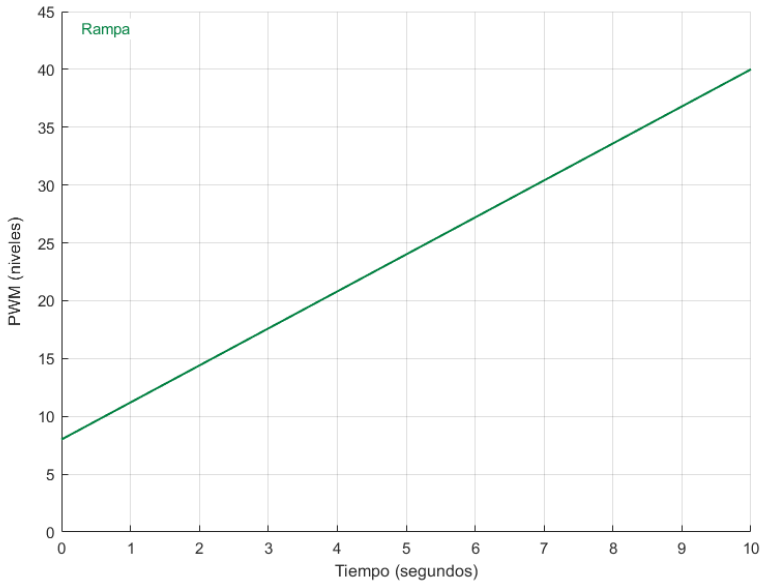


Figura 2.14: Entrada PWM de tipo rampa

El siguiente paso, antes de ajustar los parámetros, ha sido probar con una entrada de tipo rampa. Ésta aparece en la Figura 2.14, subiendo la amplitud PWM desde 8 niveles hasta 40 de forma constante durante 10 segundos.

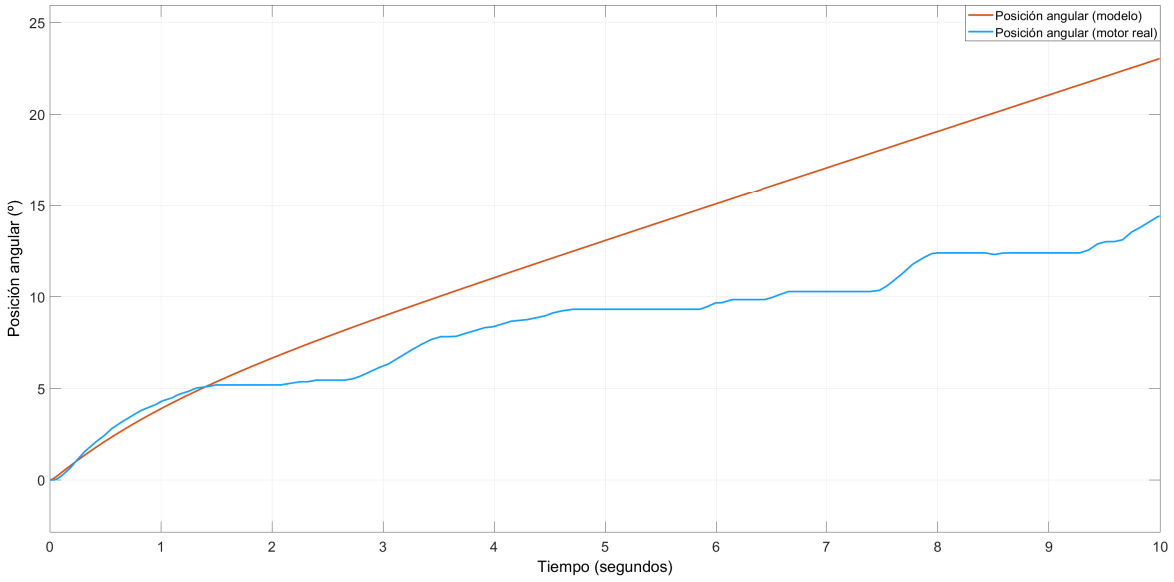


Figura 2.15: Respuesta inicial en posición (grados) del motor real (en azul) frente a la respuesta obtenida con el modelo (en naranja), ante una entrada PWM de tipo rampa

El resultado (Fig. 2.15) muestra que existen no linealidades en el motor que no han sido modeladas, como la fricción estática. Por este motivo, la recta azul (respuesta del motor real) permanece invariable durante ciertos intervalos, de tal forma que el motor mantiene su posición angular a pesar de que el voltaje aumenta. Estas zonas invariables terminan cuando se supera la fricción estática del motor.

Por otro lado, la respuesta del modelo, aunque es completamente lineal y no presenta los efectos de la fricción estática (ya que no se ha modelado por cuestiones de simplicidad), sigue la tendencia de la respuesta del motor pero sin ajustarse de forma precisa, tal y como ocurría con las entradas PWM de amplitud constante.

2.4.3. Reajuste de parámetros

Simulink posee una herramienta que permite realizar un afinamiento de los parámetros con el fin de que la respuesta del modelo se ajuste con mayor fidelidad a la del motor real. Esta aplicación se denomina *Parameter Estimation*, y para utilizarla únicamente se debe seleccionar la salida del modelo que se desea ajustar y los parámetros cuyos valores se van a modificar para realizar el afinamiento. A partir de ahí, la aplicación modifica los parámetros dentro de un rango de valores especificado y durante una serie de iteraciones, hasta alcanzar un número máximo de éstas o un límite en la optimización. La función de costes en la que se basa el ajuste de los parámetros se puede programar para buscar la mínima suma del error cuadrático o del error absoluto. Ambas se han probado para el ajuste de este modelo y han ofrecido el mismo resultado.

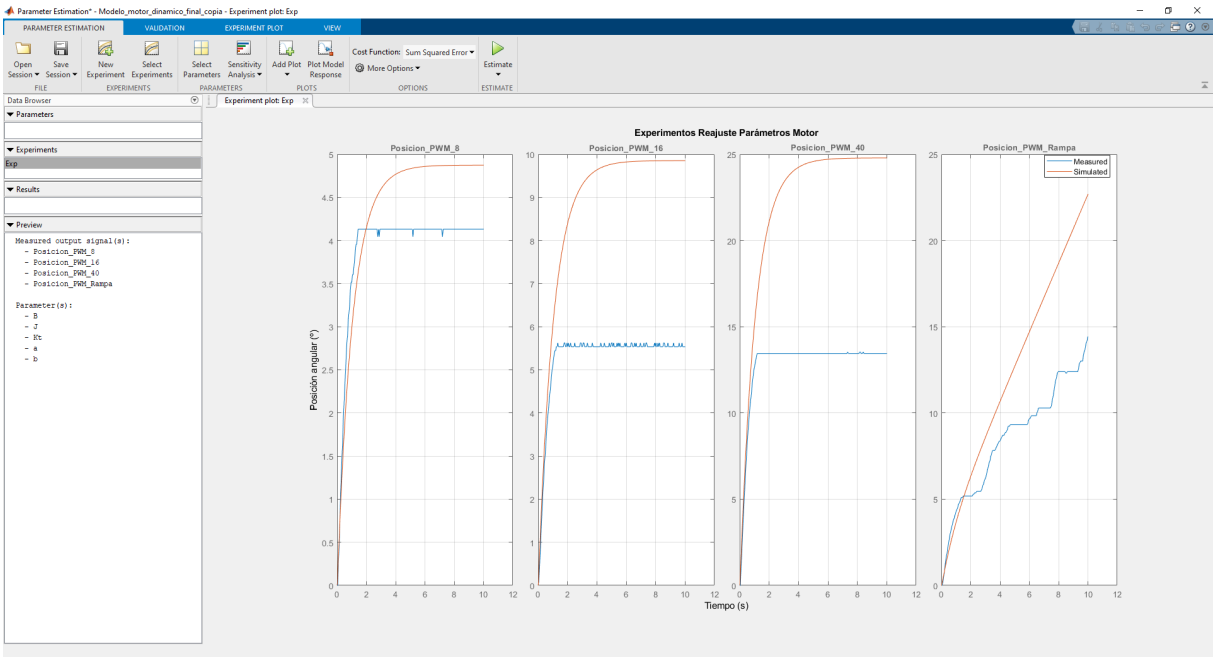


Figura 2.16: Comparativa de las respuestas entre modelo y motor para diferentes entradas antes del ajuste de los parámetros

Como se dispone de la respuesta del motor para diferentes entradas, se van a emplear todas conjuntamente para el reajuste de parámetros. De esta forma, aumenta la probabilidad de que el modelo mejore la precisión de su respuesta para cualquier posible entrada PWM. En la Figura 2.16 se observa la interfaz de la herramienta *Parameter Estimation* en la fase previa al reajuste de los parámetros. Respecto a la configuración que ofrece, no se ha modificado ninguna de las opciones, y se ha dejado como función de costes la que está seleccionada por defecto, la basada en la mínima suma del error cuadrático.

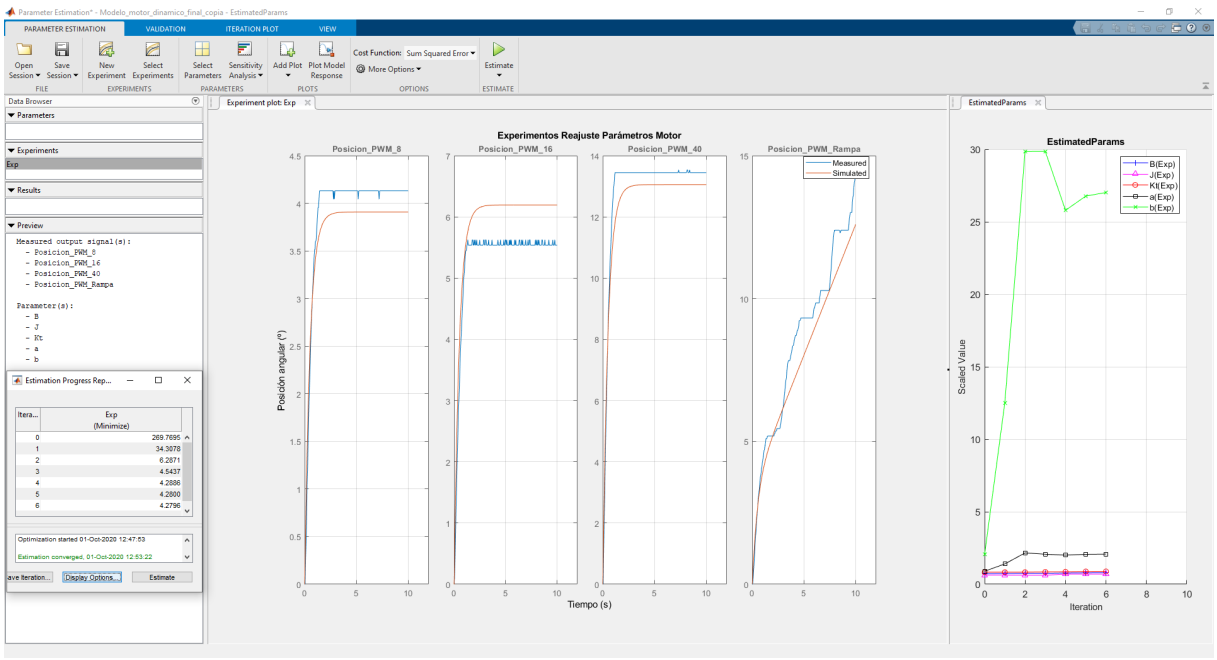
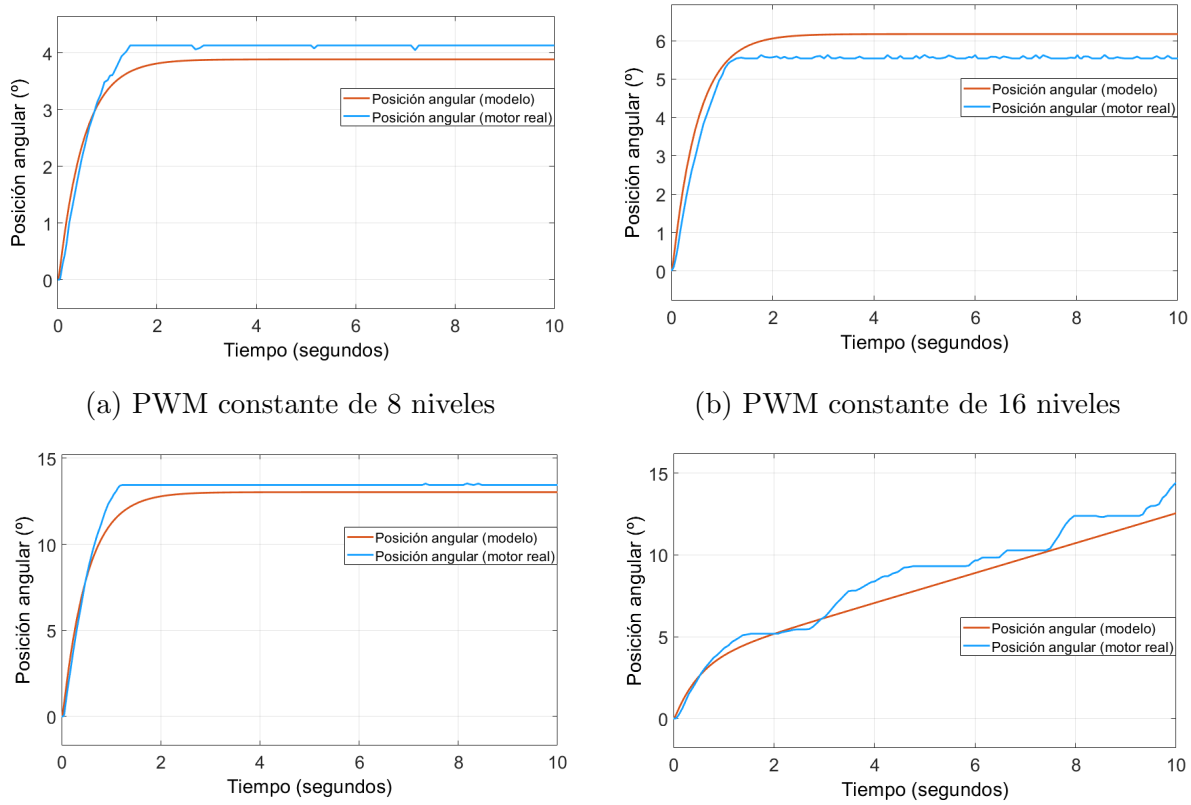


Figura 2.17: Comparativa de las respuestas entre modelo y motor para diferentes entradas después del ajuste de los parámetros (izquierda), y evolución de los valores de estos (derecha)

Tabla 2.3: Valores finales de los parámetros del motor

Parámetro	**Valor inicial**
Constante de par (K_t)	0.0069306 N/m
Momento de inercia del eje de carga (J)	0.01089778 kgm^2
Coeficiente de fricción del eje de carga (b)	0.01245892 Nms
Pendiente de la recta Posición angular - Par muelle (m)	0.01625
Ordenada de la recta Posición angular - Par muelle (n)	0.026

Tras 6 iteraciones, se aprecia como la respuesta del modelo ha mejorado para las cuatro entradas (Fig. 2.17), disminuyendo el error total entre las respuestas del modelo y del motor desde 269.7695 hasta 4.2796 unidades. Además, es posible observar cómo han evolucionado los parámetros modificables del motor a lo largo del proceso de reajuste. Los valores finales de estos se muestran en la Tabla 2.3. Cabe destacar que se ha establecido como límite inferior el valor 0 para los parámetros físicos del motor (K_t, J, yb), ya que debido a su naturaleza no pueden alcanzar valores negativos. Además, se ha restringido la modificación de estos tres parámetros a un $\pm 10\,\%$, puesto que se supone que no deben variar en exceso respecto a los valores iniciales. En cuanto a los parámetros m y n, los cuales definen la ecuación del par ejercido por el muelle en función de la posición angular del motor, estos se han dejado con total libertad para cambiar su valor, y de hecho han sufrido modificaciones considerables. Este importante cambio en sus valores podría deberse a posibles errores en la obtención de los datos del muelle, derivados de la dificultad de realizar la medición del incremento de longitud del mismo para diferentes voltajes de entrada.



(a) PWM constante de 8 niveles

(b) PWM constante de 16 niveles

(c) PWM constante de 40 niveles

(d) PWM de tipo rampa

Figura 2.18: Respuesta del modelo (en naranja) frente a la del motor (en azul) ante diferentes entradas, tras el reajuste

En la Figura 2.18 se muestran las respuestas del modelo ante las diferentes entradas estudiadas tras el reajuste de los parámetros del modelo. Se puede comprobar que todas se ajustan mejor a la respuesta del motor, verificando así que el calibrado del modelo del motor ha tenido éxito.

Tabla 2.4: Error relativo cometido en régimen permanente antes y después del reajuste

Entrada Voltaje PWM (niveles)	**Error relativo inicial (%)**	**Error relativo final(%)**
8	17.93	5.38
16	77.69	11.51
40	84.24	2.92

Para verificar cuánto ha mejorado la fidelidad del modelo dinámico del servomotor, se ha calculado el error relativo cometido en régimen permanente para cada una de las entradas de voltaje PWM constante, y se ha comparado el obtenido inicialmente con el logrado tras el reajuste de los parámetros. Los resultados se muestran en la Tabla 2.4, en la cual se puede comprobar que la mejora es más que notable: con los valores finales, el mayor error relativo es del 11.51 % (para un PWM de 16 niveles de amplitud) frente al error del 84.24 % obtenido de forma previa al reajuste (para un PWM de 40). El hecho de

que el error cometido no aumente en correlación con la amplitud del voltaje de entrada se debe a las no linealidades del motor, principalmente las debidas a la fricción estática del mismo.

2.4.4. Análisis con entrada PWM de tipo secuencia de impulsos

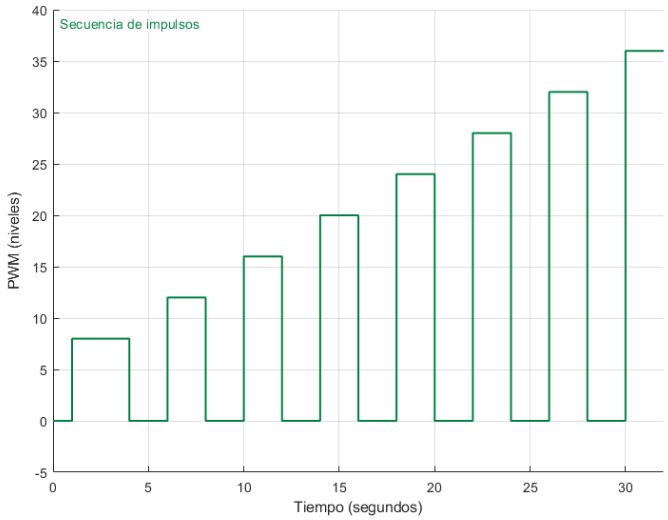


Figura 2.19: Entrada PWM de tipo secuencia de impulsos

Con el fin de probar el modelo con los nuevos valores ante una entrada de diferente naturaleza, se ha introducido al motor la señal de la Figura 2.19. Ésta consiste en una secuencia de impulsos, cuya amplitud va desde 8 niveles hasta 36. Después de cada impulso, el voltaje PWM es nulo durante 2 segundos. La secuencia total tiene una duración de 32 segundos.

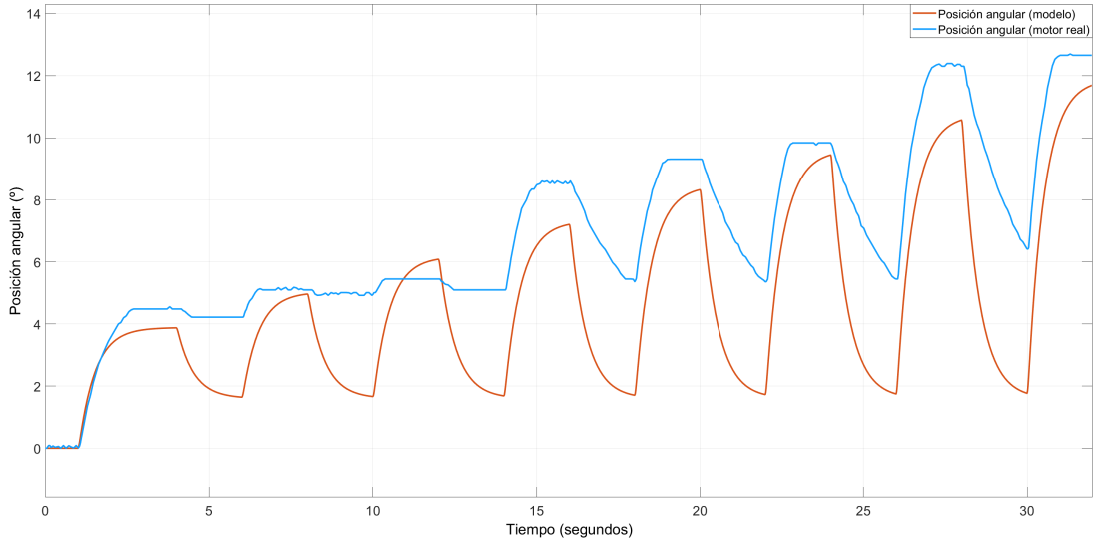


Figura 2.20: Respuesta en posición (grados) del motor real (en azul) frente a la respuesta obtenida con el modelo (en naranja), ante una entrada PWM de tipo secuencia de impulsos

El resultado de aplicar esta entrada al modelo y al motor se muestra en la Figura 2.20. En ella, se aprecia con claridad el efecto de la fricción estática sobre el motor: cuando el voltaje PWM es nulo, el motor no es capaz de recuperar su posición angular inicial, puesto que debe contrarrestar esta fricción. Sin embargo, en la respuesta del modelo se observa como el motor prácticamente recupera su posición angular después de cada impulso, como si el muelle ejerciera su par sobre el motor libre, sin fricción estática.

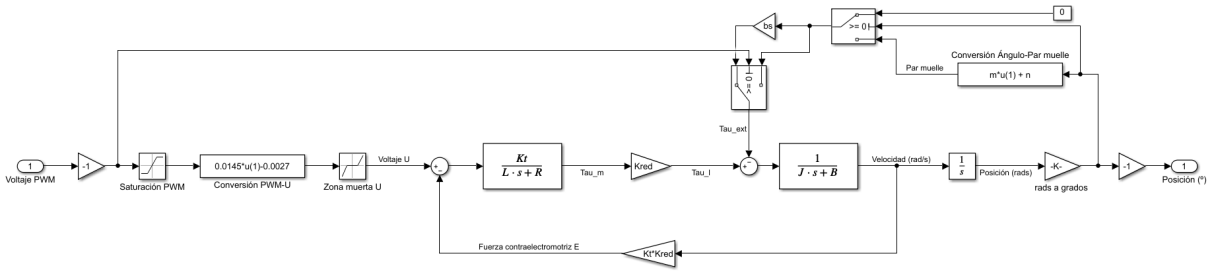


Figura 2.21: Modelo del servomotor en *Simulink* modificado para introducir la fricción estática del motor

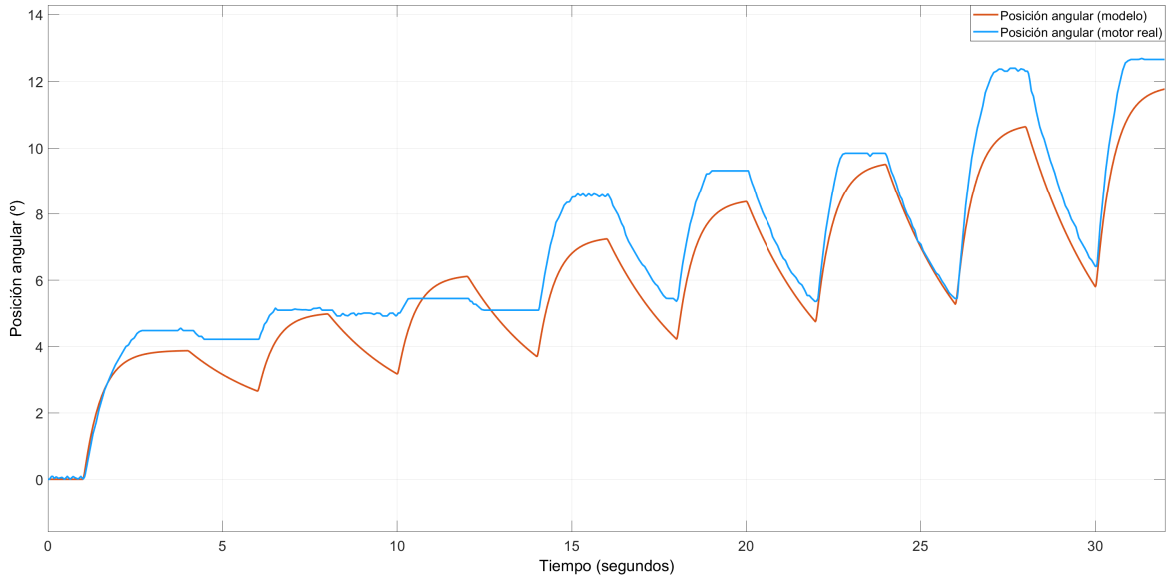


Figura 2.22: Respuesta en posición (grados) del motor real (en azul) frente a la respuesta obtenida con el modelo (en naranja), ante una entrada PWM de tipo secuencia de impulsos y con la fricción estática modelada

Para tratar de mejorar esta respuesta, se ha introducido una modificación en el modelo (Fig. 2.21), de tal forma que cuando el voltaje de entrada sea nulo, el par ejercido por el muelle no se aplique directamente sobre el modelo, sino que se multiplique previamente por una constante de reducción, denominada b_s. De tal forma, esta modificación actúa únicamente cuando el voltaje es nulo, disminuyendo el par ejercido por el muelle, ya que este debe contrarrestar la fricción estática del motor. Ésta se podría considerar una primera aproximación al modelado de una fricción estática, aunque no se va a profundizar más en este aspecto. Se ha probado a darle un valor de 0.2 a b_s, obteniendo con ello el resultado de la Figura 2.22.

2.5. Conclusión de la validación del modelo

Con los resultados obtenidos tras el reajuste de los parámetros del motor, se puede concluir que el modelo presenta una alta fiabilidad respecto a la respuesta del motor ante entradas de amplitud constante o de tipo rampa. Para entradas que alternen entre un valor PWM (negativo en este caso) y un valor nulo, sería necesario modelar la fricción estática del motor para obtener resultados fiables. Esto se propone como línea de trabajo futuro, siendo suficiente para este proyecto un modelo más simple del motor. Por lo tanto, se da por finalizado el modelado dinámico del servomotor.

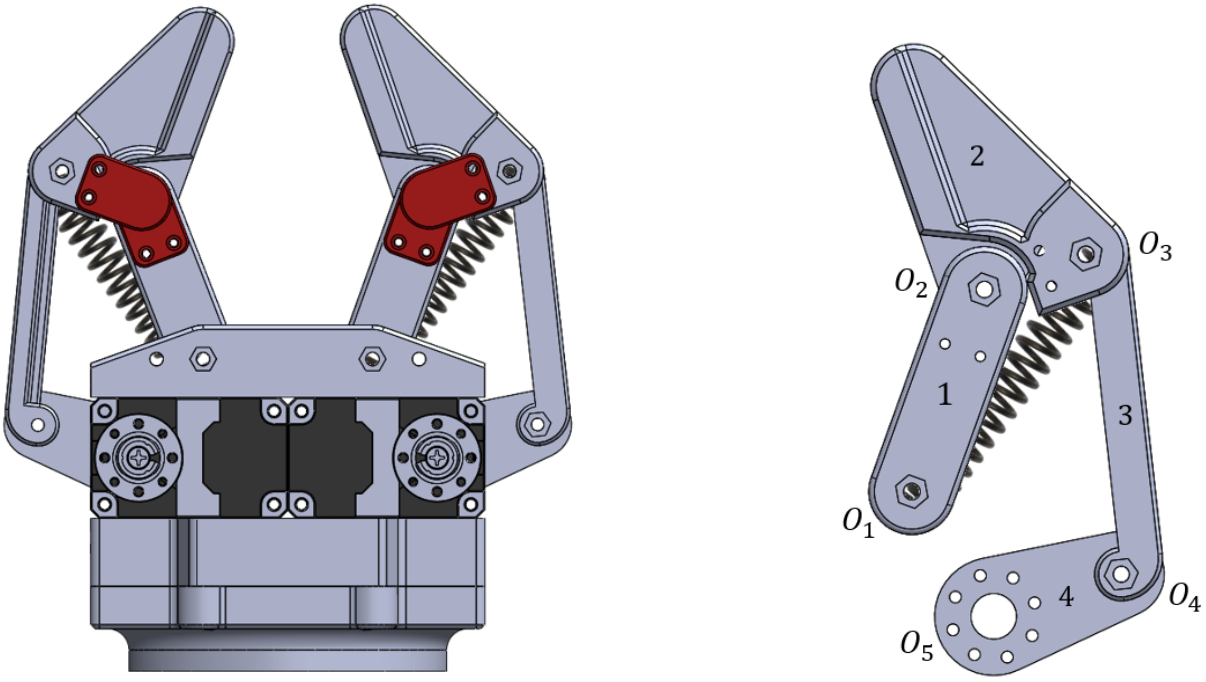
Sección 3

Modelado de la garra subactuada

3.1. Introducción

En esta sección se van a introducir primeramente el modelo cinemático desarrollado para uno de los dedos de la garra subactuada, obteniendo la expresión de la posición articular de todos sus eslabones en función de ángulos o longitudes conocidas. Posteriormente, la sección se centra en el desarrollo de un modelo dinámico del manipulador y su posterior implementación en *Simulink*.

El robot manipulador de este proyecto es una garra subactuada de dos dedos, compuesta por dos servomotores Dynamixel modelo *XM430-W210-R*, cada uno de los cuales proporciona un par de entrada y dos grados de libertad a cada dedo. Por su parte, estos presentan dos falanges que juntas conforman un manipulador plano de cadena cinemática cerrada, puesto que se puede llegar desde cualquier eslabón a cualquier otro por dos caminos. En cuanto al número de eslabones, se trata de un mecanismo de cinco barras, con un muelle que une las articulaciones 1 y 3, proporcionando a la garra su naturaleza de manipulador subactuado.



(a) Garra subactuada (b) Un dedo del manipulador

Figura 3.1: Modelo 3D del manipulador, con la arquitectura de un dedo en detalle (Fig. 3.1b)

En la Figura 3.1 se muestra la garra junto con uno de sus dedos ampliado. Como se puede observar, se trata de un manipulador complejo, que por su cadena cinemática cerrada no puede ser modelado dinámicamente utilizando métodos tradicionales tales como la formulación de Lagrange-Euler y las ecuaciones de Newton-Euler.

3.2. Modelo cinemático

El primer paso para desarrollar el modelo de este manipulador, es calcular su modelo cinemático, puesto que las posiciones articulares aportan información acerca de cómo es el objeto que está agarrando y con el que está interactuando el manipulador.

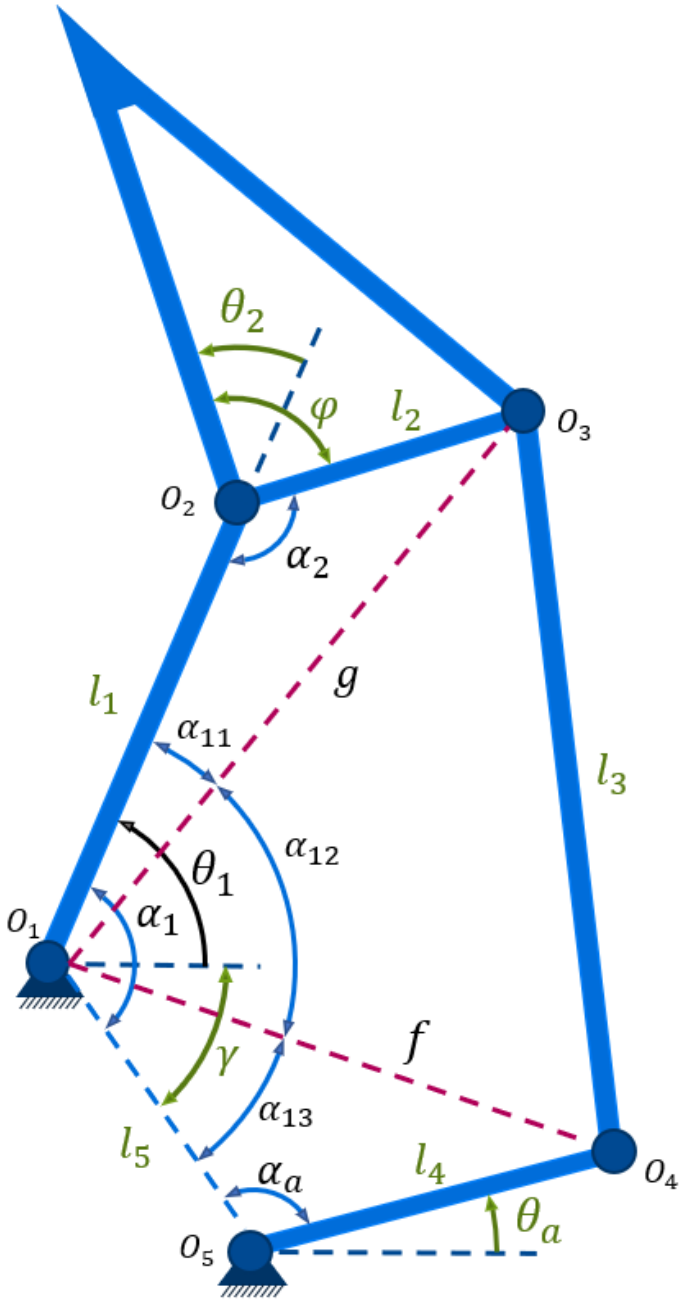


Figura 3.2: Mecanismo de cinco barras que constituye un dedo de la garra subactuada

Los parámetros que se conocen (en color verde en la Figura 3.2) y que permiten resolver el modelo cinemático de la garra son las longitudes de los eslabones, los ángulos invariantes ψ y γ, y los ángulos θ_2 y θ_a. De estos dos últimos, el primero se obtiene a partir de unos potenciómetros que miden el ángulo relativo entre las dos falanges del dedo, y el segundo se conoce directamente gracias al controlador del servo. Con estos datos, y calculando previamente ciertas distancias y ángulos auxiliares, se puede obtener

la expresión del ángulo θ_1 (en color negro en la Figura) y, por ende, el modelo cinemático del manipulador. Para ello, es necesario emplear diversas funciones trigonométricas y los teoremas del seno y el coseno.

El primer paso es obtener las expresiones de los ángulos auxiliares α_1, α_2 y α_a. Esto se puede hacer mediante simple observación, buscando encontrar expresiones que relacionen estos ángulos con otros conocidos o con el ángulo π.

$$\alpha_1 = \theta_1 + \gamma \tag{3.1}$$

$$\alpha_2 = \pi - \psi + \theta_2 \tag{3.2}$$

$$\alpha_a = \pi - \theta_a - \gamma \tag{3.3}$$

Posteriormente, se calculan las distancias auxiliares denotadas como f y g utilizando para ello el teorema del coseno.

$$f^2 = l_4^2 + l_5^2 - 2l_4l_5cos(\alpha_a) \Rightarrow f = \sqrt{l_4^2 + l_5^2 - 2l_4l_5cos(\alpha_a)} \tag{3.4}$$

Desarrollando el coseno del ángulo α_a (Ec. 3.3) y teniendo en cuenta la ecuación del coseno de una resta de ángulos, se puede simplificar la expresión y obtenerla directamente en función del ángulo variable θ_a.

$$cos(\alpha_a) = cos(\pi - (\theta_a + \gamma)) = cos(\pi)cos(\theta_a + \gamma) + sin(\pi)sin(\theta_a + \gamma) = -cos(\theta_a + \gamma) \tag{3.5}$$

Por lo tanto, la Ecuación 3.4 quedaría tal que:

$$f = \sqrt{l_4^2 + l_5^2 + 2l_4l_5cos(\theta_a + \gamma)} \tag{3.6}$$

De manera similar puede calcularse la distancia g, en esta ocasión a partir del ángulo α_2 o, simplificando, del ángulo θ_2.

$$g^2 = l_1^2 + l_2^2 + 2l_1l_2cos(\psi - \theta_2) \Rightarrow g = \sqrt{l_1^2 + l_2^2 + 2l_1l_2cos(\psi - \theta_2)} \tag{3.7}$$

Una vez se han obtenido las distancias y ángulos auxiliares anteriores, es posible obtener θ_1 en función de variables conocidas. Para ello, se parte de la Ecuación 3.1, teniendo en cuenta que este ángulo equivale a la suma de otros tres ángulos, denotados como α_{11}, α_{12} y α_{13}.

$$\alpha_1 = \theta_1 + \gamma = \alpha_{11} + \alpha_{12} + \alpha_{13} \Rightarrow \theta_1 = \alpha_{11} + \alpha_{12} + \alpha_{13} - \gamma \tag{3.8}$$

Por lo tanto, el siguiente paso es calcular estos tres ángulos, el primero y el tercero mediante el teorema del seno, y el segundo mediante el del coseno, haciendo uso de las distancias y ángulos ya conocidos.

$$\alpha_{11} : \frac{l_2}{sin(\alpha_{11})} = \frac{g}{sin(\psi - \theta_2)} \Rightarrow \alpha_{11} = asin(\frac{l_2}{g} sin(\psi - \theta_2)) \tag{3.9}$$

$$\alpha_{12} : l_3^2 = g^2 + f^2 - 2gfcos(\alpha_{12}) \Rightarrow \alpha_{12} = acos(\frac{g^2 + f^2 - l_3^2}{2gf}) \tag{3.10}$$

$$\alpha_{13} : \frac{l_4}{sin(\alpha_{13})} = \frac{f}{sin(\theta_a + \gamma)} \Rightarrow \alpha_{13} = asin(\frac{l_4}{f} sin(\theta_a + \gamma)) \tag{3.11}$$

Sustituyendo las Ecuaciones 3.9, 3.10 y 3.11 en la Ecuación 3.8, se obtiene la expresión final del ángulo θ_1, quedando completamente definido el modelo cinemático del manipulador robótico.

$$\theta_1 = asin(\frac{l_2}{g} sin(\psi - \theta_2)) + acos(\frac{g^2 + f^2 - l_3^2}{2gf}) + asin(\frac{l_4}{f} sin(\theta_a + \gamma)) - \gamma \tag{3.12}$$

3.3. Modelo estático

En la actualidad, numerosas investigaciones tratan de adaptar los métodos de modelado dinámico tradicionales (formulaciones de Lagrange-Euler y Newton-Euler) al desarrollo de modelos dinámicos para manipuladores de cadena cinemática cerrada. Sin embargo, la mayoría de ellos sufren de grandes cargas computacionales, consecuencias de combinar el uso de ecuaciones diferenciales y algebraicas para la obtención del modelo. Por este motivo, para este proyecto se ha estudiado el posible desarrollo del modelo mediante una de las técnicas alternativas que ofrece el estado del arte: un algoritmo recursivo basado en el principio de Gibbs–Appell ([9]). Los fundamentos de este desarrollo se encuentran en el Anexo C.

Sin embargo, en el manipulador de este proyecto el elemento que aporta la mayor parte de la dinámica es el actuador. Esto es consecuencia de diversas razones:

- El actuador es la parte que presenta mayor inercia en el sistema, ya que la relación entre el eje del motor y la salida del eje es de 212.6 a 1.
- Los eslabones que conforman los dedos tienen un tamaño reducido (el más largo es de 6cm), y al estar impresos en 3D su masa es muy pequeña (entre 3.2 y 13.6 gramos).
- Es necesario controlar la fuerza de agarre en todo momento, por lo tanto los dedos se mueven a muy bajas velocidades.

Por estos motivos, desarrollar un modelo dinámico tradicional que incorpore la dinámica completa de todos los elementos del manipulador es algo innecesario en este proyecto, ya que prácticamente toda la dinámica de la garra se encuentra en el actuador. De hecho, los propios errores de realizar un modelado dinámico tradicional van a ser

incluso mayores que los que se puedan conseguir sin tener en cuenta la dinámica de la garra. Por lo tanto, se ha decidido implementar un modelo estático-dinámico o modelo híbrido, donde por un lado se considera la dinámica del actuador (ya desarrollada en la Sección 2), y por otra se calcula la posición y la fuerza de agarre del manipulador mediante un modelo estático.

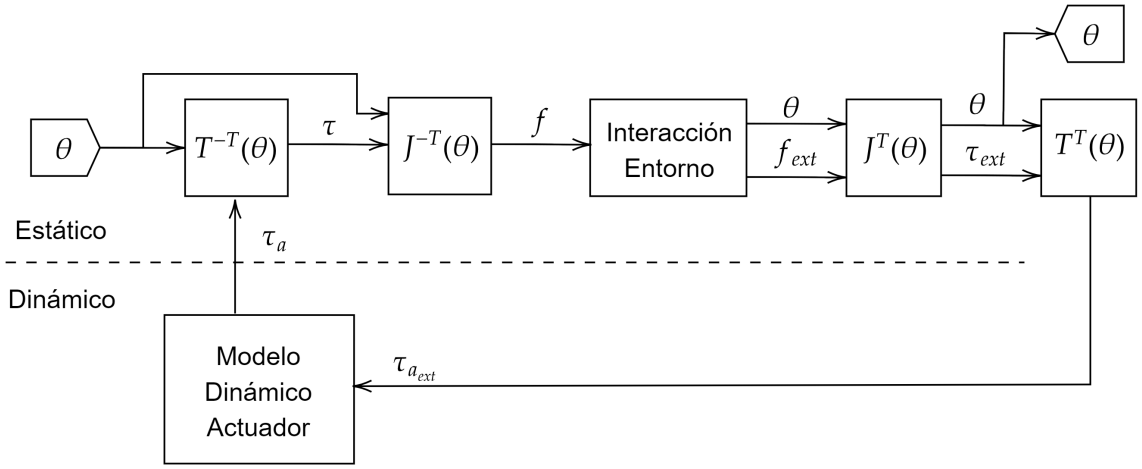


Figura 3.3: Esquema del modelo híbrido implementado

En la Figura 3.3 se observa el esquema a un nivel superior del modelo híbrido implementado. En él se incluyen tanto el modelo dinámico del motor, para obtener el par generado por este y su posición angular, como el modelo estático de la garra, para calcular en todo momento las posiciones de sus articulaciones θ_1 y θ_2. A partir de estos datos, se obtienen los pares que actúan sobre estas articulaciones mediante la matriz de transferencia del manipulador ($T(\theta)$), e introduciendo estos pares en el jacobiano ($J(\theta)$) se pueden calcular las fuerzas que está aplicando cada falange sobre un objeto determinado. Con el objetivo de simplificar el problema, se han considerado una serie de supuestos:

- El objeto a agarrar es una esfera de elasticidad y radio conocidos (Fig. 3.4). Además, el problema se convierte en puramente estático, estudiándose la deformación de la esfera únicamente desde este punto de vista, sin tener en cuenta su dinámica.
- La esfera se encuentra en el centro de la garra, en contacto (aunque sin estar aplicando fuerza) con ambas falanges en el momento inicial.
- El punto de contacto de la esfera con cada falange va a tener lugar en el centro de cada una, respectivamente. Estos puntos se van a denominar K_1 y K_2, siendo el subíndice 1 la falange proximal y el 2 la distal.

Teniendo en cuenta estas consideraciones, se ha implementado un modelo de interacción de las falanges con el entorno, de tal forma que a partir de las fuerzas que están aplicando, se puedan calcular las nuevas posiciones angulares de ambas falanges. Aplicando esta vez la matriz jacobiana inversa y la matriz de transferencia inversa, se obtiene el par que ejerce la esfera sobre el actuador, y este se introduce en el modelo dinámico del motor.

En los próximos subapartados se va a estudiar cada bloque por separado.

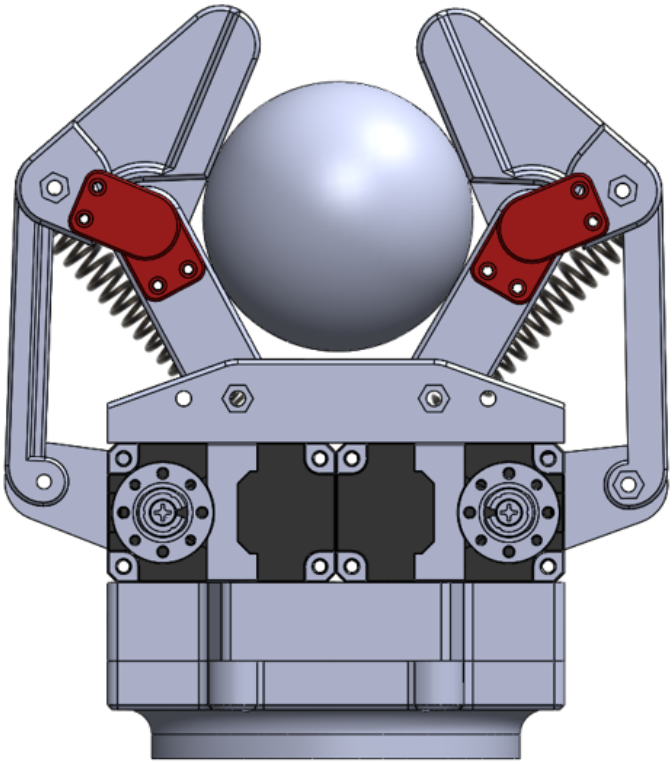


Figura 3.4: Modelo simplificado de un objeto determinado para poder simular la interacción física entre este y las falanges del manipulador mediante el modelo estático desarrollado

3.3.1. Matriz de transferencia

Es la matriz que relaciona el par generado por el actuador, (T_a), y por el muelle, (T_2). El primero se obtiene a la salida del modelo dinámico del motor, mientras que el segundo se calcula en función del primero, tal que:

$$T_2 = K_m(\theta_2 - \theta_{2_0})T_a \tag{3.13}$$

Donde K_m es la constante de elasticidad del muelle situado entre las articulaciones O_1 y O_3, y θ_{2_o} es la posición articular de la falange distal en el inicio del movimiento, siendo θ_2 la posición actual de dicha falange.

Estos dos pares, T_a y T_2, componen el vector t de la siguiente ecuación:

$$\tau = T^{-T}t \tag{3.14}$$

Dónde $\tau = [\tau_1\ \tau_2]^T$ es el vector que contiene los pares presentes en las articulaciones de las dos falanges, en θ_1 y θ_2. En cuanto a la matriz de transferencia (T), ésta se opera en su forma inversa y traspuesta, que es la siguiente:

$$T^{-T} = \begin{bmatrix} 1 & 0 \\ \frac{h}{h+l_1} & 1 \end{bmatrix} \tag{3.15}$$

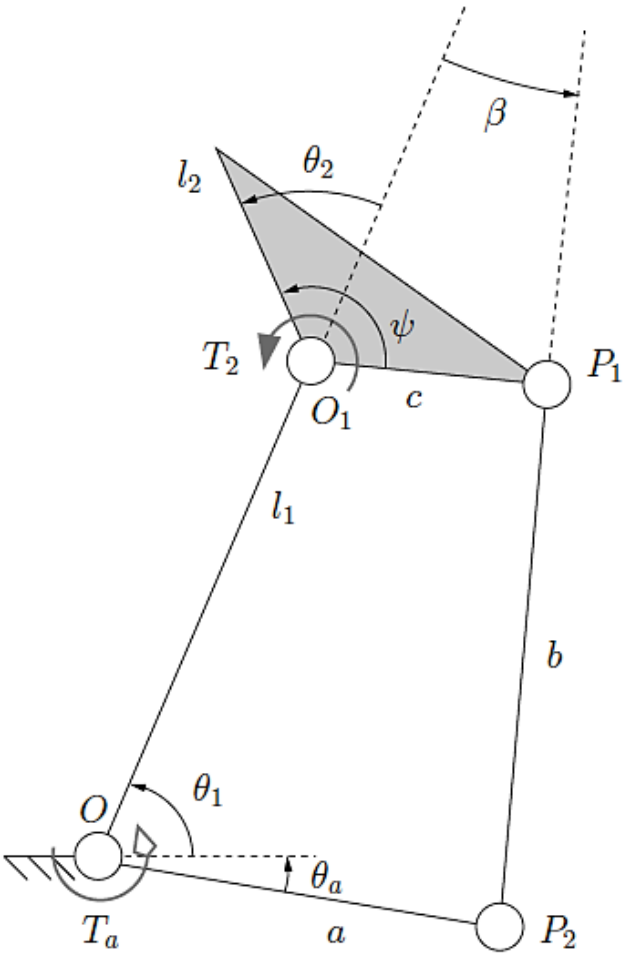


Figura 3.5: Manipulador de dos falanges y cuatro barras, modelo para el cual calcular la matriz de transferencia ([10])

El parámetro l_1 se refiere a la longitud de la falange proximal, mientras que la letra h es un parámetro empleado en la literatura consultada ([10]) para un manipulador de dos falanges con un mecanismo de cuatro barras (Fig. 3.5). Por lo tanto, para poder calcular la matriz de transferencia mediante este método, es necesario realizar una transformación previa para convertir el mecanismo de cinco barras de este proyecto en uno de cuatro.

Para ello, se analizan las diferencias entre ambos mecanismos, comprobando que el eslabón a del mecanismo de la Figura 3.5 es en realidad la barra imaginaria f del manipulador de este proyecto. Esta barra (Fig. 3.6) es la que conecta las articulaciones O_1 y O_4, y su cálculo se realizó en la Ecuación 3.6. Sabiendo esta similitud y viendo el mecanismo de cinco barras, se deduce que es posible trasladar el par generado por el actuador en la articulación O_5 ($T_a{}'$) al par presente en la articulación O_1 (T_a).

El primer paso para trasladar dicho par es calcular la fuerza f_a que lo provoca, de tal forma que:

$$T_a{}' = f_a l_4 \Rightarrow f_a = \frac{T_a{}'}{l_4} \tag{3.16}$$

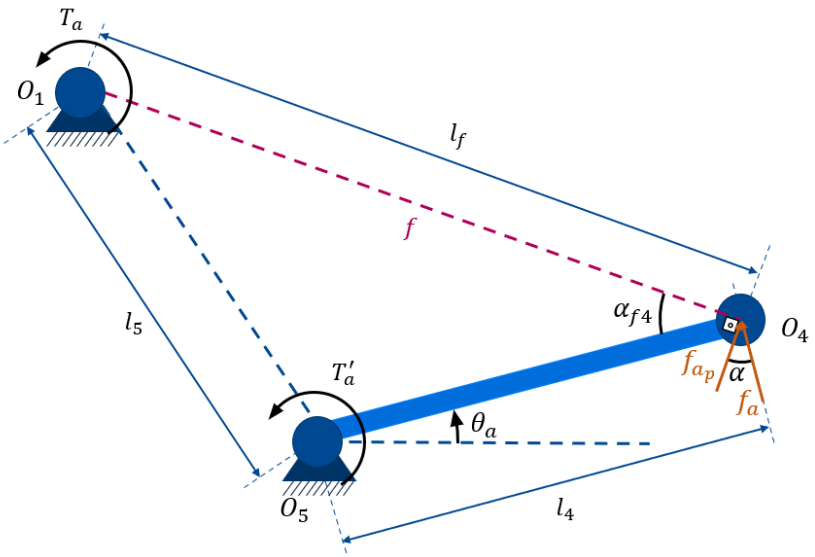


Figura 3.6: Zona inferior del manipulador de cinco barras

Posteriormente, se proyecta esta fuerza sobre la perpendicular del eslabón imaginario f. Utilizando el ángulo α y trigonometría básica, se tiene que:

$$cos(\alpha) = \frac{f_{a_p}}{f_a} \Rightarrow f_{a_p} = f_a cos(\alpha) \tag{3.17}$$

Siendo f_{a_p} la proyección de la fuerza f_a en la dirección perpendicular a la recta f. Respecto al ángulo α, este puede calcularse aplicando el teorema del coseno sobre el triángulo formado por las articulaciones O_1,O_4 y O_5. Lo primero es obtener el ángulo que forma la recta f (cuya longitud se denomina l_f) con el eslabón cuarto (el que conecta O_4 con O_5):

$$l_5^2 = l_f^2 + l_4^2 - 2 l_f l_4 cos(\alpha_{f4}) \Rightarrow \alpha_{f4} = acos(\frac{l_f^2 + l_4^2 - l_5^2}{2 l_f l_4}) \tag{3.18}$$

Sabiendo que la fuerza fa es perpendicular al eslabón cuarto, y la fuerza f_{a_p} lo es a la recta f, se puede determinar el ángulo α igualando ambos ángulos rectos, de tal forma que:

$$\frac{\pi}{2} - \alpha_{f4} = \frac{\pi}{2} - \alpha \Rightarrow \alpha = \alpha_{f4} \tag{3.19}$$

De esta manera el ángulo α, así como la fuerza proyectada f_{a_p} serían conocidos, haciendo posible obtener el par T_a de manera directa:

$$T_a = f_{a_p} l_f \tag{3.20}$$

De esta forma, se ha conseguido adaptar el mecanismo del manipulador de este proyecto al genérico de cuatro barras para el cual se puede aplicar la matriz de transferencia vista en la Ecuación 3.15. El parámetro h que aparece en ella se calcula tal que así:

$$h = l_2(cos(\theta_2 - \psi) - sen(\theta_2 - \psi)cot(\beta)) \tag{3.21}$$

Donde se puede sustituir la cotangente de β por su expresión:

$$cot(\beta) = \frac{l_2 sin(\theta_2 - \psi)\sqrt{4l_f^2 l_3^2 - N^2} + M(l_1 + l_2 cos(\theta_2 - \psi))}{-(l_1 + l_2 cos(\theta_2 - \psi))\sqrt{4l_f^2 l_3^2 - N} + M l_2 sin(\theta_2 - \psi)} \tag{3.22}$$

Siendo M y N dos variables auxiliares cuyas ecuaciones son:

$$M = -l_1(l_1 + 2l_2 cos(\theta_2 - \psi)) + l_f^2 - l_3^2 - l_2^2 \tag{3.23}$$

$$N = l_1(l_1 + 2l_2 cos(\theta_2 - \psi)) - l_f^2 - l_3^2 + l_2^2 \tag{3.24}$$

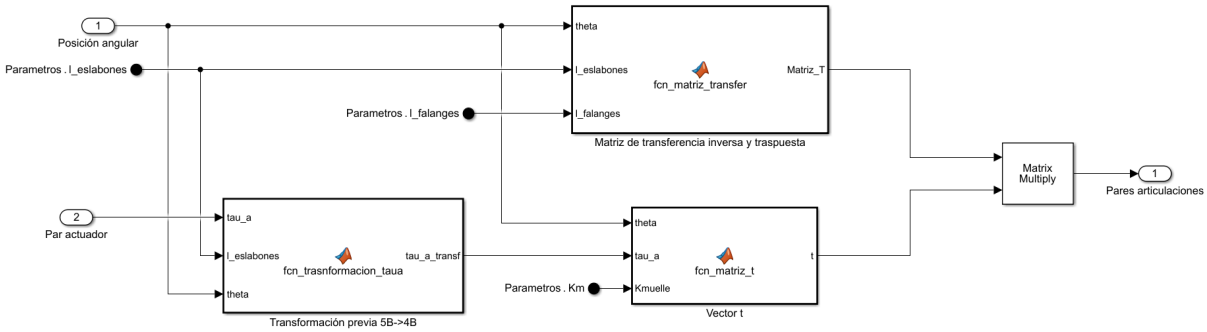


Figura 3.7: Bloque *Matriz de transferencia*

Resolviendo el valor del parámetro h para una determinada posición angular de la falange distal (θ_2), es posible calcular la matriz de transferencia y hallar los pares vistos en las articulaciones O_1 y O_2 a partir del par generado por el actuador y por el muelle. El modelo implementado en *Simulink* se observa en la Figura 3.7, y el código de cada bloque se encuentra en el Anexo A.2, así como los códigos de todas las funciones que se desarrollan en los siguientes apartados.

3.3.2. Cálculo del jacobiano

Tras hallar los pares que actúan en las articulaciones de ambas falanges, τ_1 y τ_2, el siguiente paso es obtener la fuerza de agarre o contacto que ejerce cada una de las falanges. Estas se escriben como f_1 y f_2, y componen el vector f en la siguiente ecuación:

$$f = J^{-T}T^{-T}t \Rightarrow f = J^{-T}\tau \tag{3.25}$$

Por lo tanto, lo único que falta por calcular para obtener las fuerzas es el jacobiano inverso traspuesto (J^{-T}). Este tiene como función traducir tanto el par en O_1 respecto al punto de contacto de la falange proximal con el objeto (K_1), como el par en O_2 respecto al punto de contacto de la falange distal con el mismo objeto (K_2). De esta forma, mediante

el jacobiano se pueden obtener las fuerzas cartesianas que ejerce el dedo del manipulador sobre el objeto con el que está en contacto.

Como el estudio es únicamente para los dos eslabones de la garra que están en contacto con un determinado objeto, el cálculo del jacobiano es sencillo, ya que a efectos de cálculo el manipulador se queda en uno de dos eslabones y 2 GDL planar ([10], [11]).

La expresión del jacobiano es la siguiente:

$$J^{-T} = \begin{bmatrix} \frac{1}{K_1} & -\frac{K_2+l_1 cos(\theta_2)}{K_1 K_2} \\ 0 & \frac{1}{K_2} \end{bmatrix} \tag{3.26}$$

Por lo tanto, calculando las fuerzas con la Ecuación 3.25 a partir de la expresión del jacobiano, de la matriz de transferencia y del vector de pares t, se obtiene que:

$$f = \begin{bmatrix} -\frac{l_1(-K_2+hcos(\theta_2))}{K_1 K_2(h+l_1)} T_a - \frac{K_2+l_1 cos(\theta_2))}{K_1 K_2} T_2 \\ \frac{h}{K_2(h+l_1)} T_a + \frac{1}{K_2} T_2 \end{bmatrix} \tag{3.27}$$

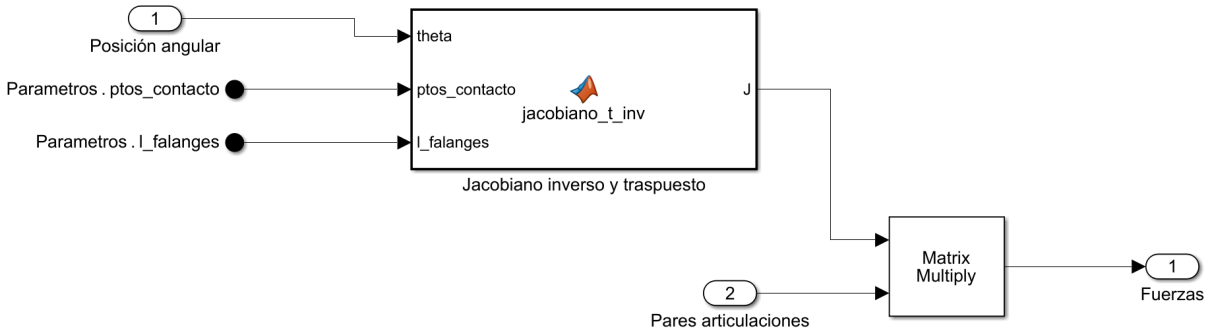


Figura 3.8: Bloque *Jacobiano*

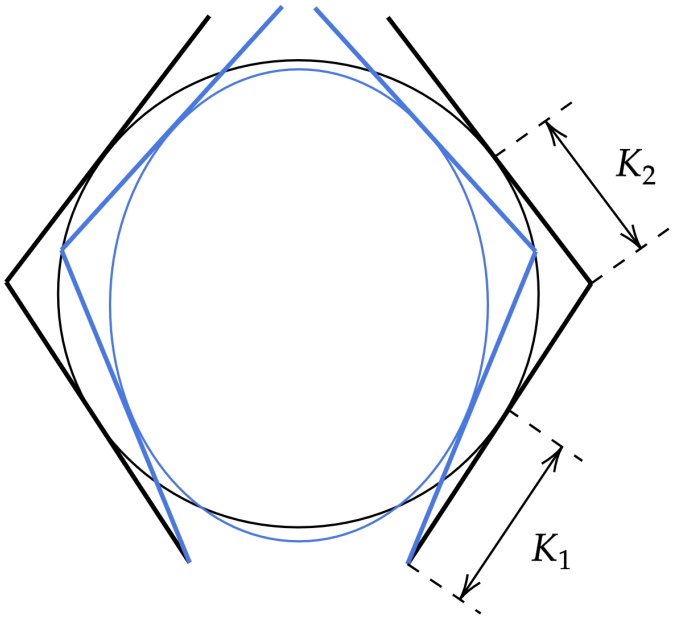
En la Figura 3.8 se observa el interior del bloque *Jacobiano*, que contiene la formulación desarrollada en este punto para la obtención de las fuerzas en las falanges a partir de los pares en las articulaciones.

3.3.3. Modelo de interacción con el entorno

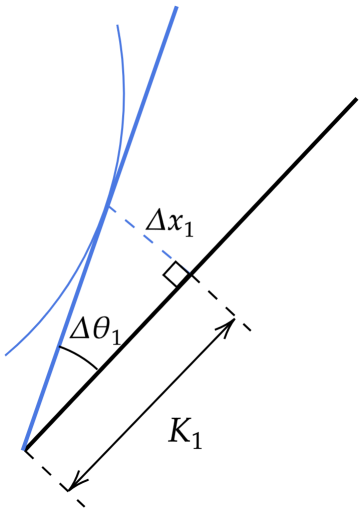
Como ya se ha comentado anteriormente, se ha supuesto una esfera con radio (r_e) y constante de elasticidad (K_e) conocidos, situada en el centro de la garra. Se considera únicamente la deformación estática de la esfera, y el objetivo de modelar la interacción de la garra con este objeto es calcular las nuevas posiciones angulares que presentan ambas falanges tras haberse deformado la esfera. Eso es posible debido a que sé qué fuerza se está aplicando con cada falange, y utilizando la constante de elasticidad de la esfera y la Ley de Hooke se evalúa cuanto se ha comprimido ésta. A partir de ahí, se obtienen los nuevos ángulos mediante relaciones trigonométricas sencillas.

De esta forma, el primer paso sería calcular las deformaciones sufridas por la esfera en los puntos de contacto de las falanges, de tal forma que:

$$f_i = K_e \Delta x_i \Rightarrow \Delta x_i = \frac{f_i}{K_e} \tag{3.28}$$



(a) Esquema de deformación de la esfera



(b) Detalle de la deformación provocada por la primera falange

Figura 3.9: Interacción de la garra con la esfera

Siendo $i = 1, 2$, dependiendo de si es la falange proximal o la distal, respectivamente. Una vez obtenidas las deformaciones sufridas por la esfera en los dos puntos de contacto (Fig. 3.9), los cuales se denominan K_1 y K_2, se obtienen las nuevas posiciones angulares, a través de las siguientes expresiones:

$$tg(\Delta\theta_i) = \frac{\Delta x_i}{K_i} \Rightarrow \Delta\theta_i = tg^{-1}(\frac{\Delta x_i}{K_i}) \tag{3.29}$$

Esta relación trigonométrica se puede observar con más detalle en la Figura 3.9b, para el caso de la articulación O_1. Para la articulación O_2 se desarrolla de forma análoga.

El último paso es sumar estos incrementos de posición a la posición angular actual de ambas falanges, tal que:

$$\theta_i = \theta_i + \Delta\theta_i \tag{3.30}$$

De esta forma, se consigue actualizar constantemente la posición angular de las falanges. Si se quisiera lograr un resultado más preciso, sería necesario tener en cuenta ciertas consideraciones:

- Inicialmente, la esfera puede no estar en contacto con las falanges, o incluso estarlo solo con la primera.

- El punto de contacto entre el objeto y las falanges no es siempre el mismo, por lo que sería necesario actualizarlo. Sin embargo, para pequeños desplazamientos estas variaciones de los puntos K_1 y K_2 pueden despreciarse en un principio.

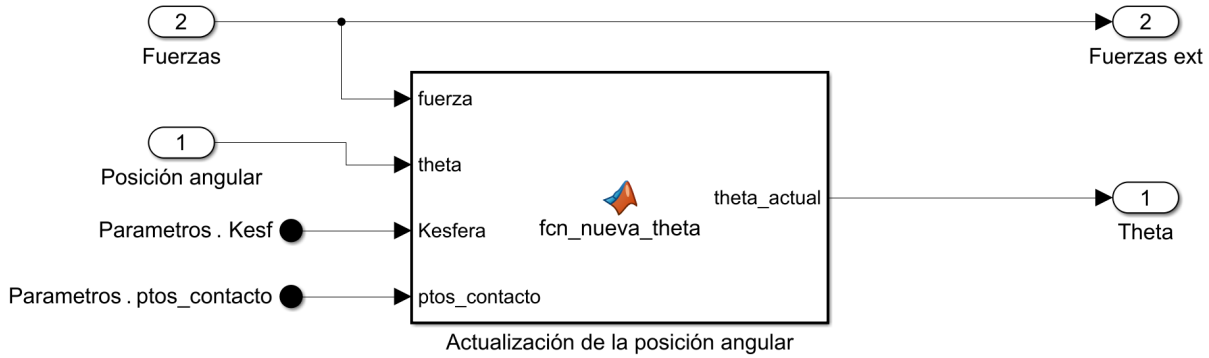


Figura 3.10: Bloque *Interacción con el entorno*

En lo que respecta al bloque en Simulink, este se muestra en la Figura 3.10. Como se puede observar, la fuerza que ejercen las falanges sobre la esfera es la misma que dicho objeto ejerce sobre las propias falanges, puesto que debe existir un equilibrio de fuerzas.

3.4. Modelo híbrido

Finalmente, se implementan todos los bloques vistos en esta Sección junto con el modelo dinámico del actuador para obtener el modelo híbrido de la Figura 3.11. Como se puede comprobar, este sigue la estructura vista anteriormente en la Figura 3.3, con el añadido de un bloque denominado *Posición angular*, que junta las posiciones angulares de ambas falanges con la del actuador en un solo vector, y del bloque *Parámetros del Manipulador*, el cual genera un vector que contiene la longitud de los eslabones, de las falanges, la ubicación de los puntos de contacto, los ángulos iniciales, y las constantes de elasticidad de la esfera y del muelle de la garra.

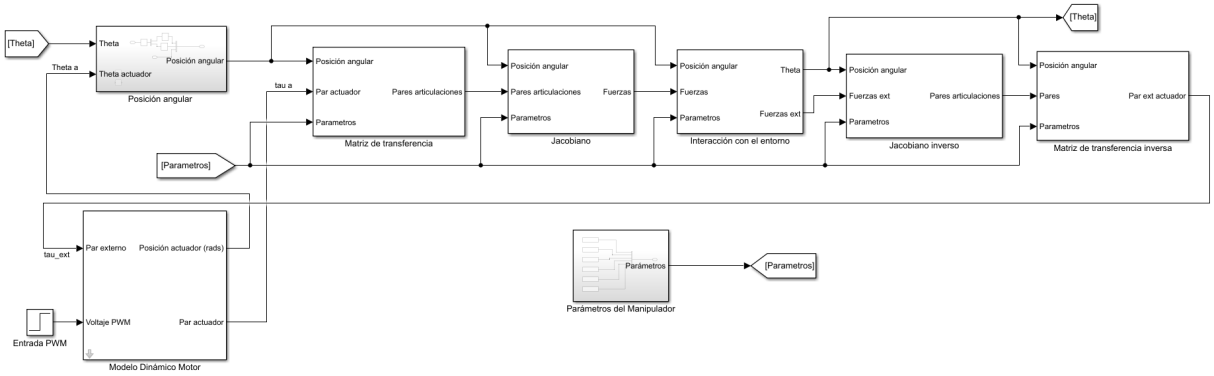


Figura 3.11: Modelo híbrido implementado en *Simulink*

Este modelo híbrido implementa todas las ecuaciones que se han visto hasta ahora, de tal forma que es capaz de determinar las fuerzas cartesianas producidas en el centro de

cada falange (f_1 y f_2) a partir de la entrada de voltaje del actuador. Para escribirlo todo en una sola ecuación, es necesario recurrir primero a la expresión matricial de las fuerzas (Ec. 3.27). Además, el torque producido por el muelle (T_2) se sustituye por su expresión (Ec. 3.13) que depende de T_a, con el objetivo de obtener las fuerzas en función del par del actuador. Sin embargo, a este par se le aplicó una transformación, para trasladarlo desde la articulación O_5 a la O_1. Por lo tanto, es necesario deshacer dicha transformación, convirtiendo el par T_a en $T_a{}'$ mediante la siguiente expresión:

$$T_a = \frac{T_a{}' cos(\frac{l_f^2+l_4^2-l_5^2}{2l_f l_4}) l_f}{l_4} \tag{3.31}$$

Esta se deriva de las ecuaciones de la Sección 3.3.1. Aplicando esta y el resto de modificaciones en la Ecuación 3.27, se obtiene que:

$$\begin{aligned} f_1 = \{ &- \frac{l_1(-K_2 + hcos(\theta_2))cos(\frac{l_f^2+l_4^2-l_5^2}{2l_f l_4}) l_f}{K_1 K_2 (h+l_1) l_4} - \\ &\frac{K_2 + l_1 cos(\theta_2)) K_m (\theta_2 - \theta_{2_0}) cos(\frac{l_f^2+l_4^2-l_5^2}{2l_f l_4}) l_f}{K_1 K_2 l_4} \} T_a{}' \end{aligned} \tag{3.32}$$

$$f_2 = \{ \frac{hcos(\frac{l_f^2+l_4^2-l_5^2}{2l_f l_4}) l_f}{K_2 (h+l_1) l_4} + \frac{K_m (\theta_2 - \theta_{2_0}) cos(\frac{l_f^2+l_4^2-l_5^2}{2l_f l_4}) l_f}{K_2 l_4} \} T_a{}' \tag{3.33}$$

En ambas ecuaciones, lo único necesario es expresar el par del actuador $T_a{}'$ como la diferencia entre el par en el eje de carga del motor (τ_l) y el par externo aplicado sobre el mismo (τ_{ext}). El par τ_l se puede escribir en función del voltaje de entrada mediante las ecuaciones obtenidas en el modelado dinámico del motor (Sección 2.2), de tal forma que:

$$\tau_l = N K_t \frac{u - N K_t \dot{\theta}_l}{R} \tag{3.34}$$

Por otra parte, en el caso del par externo se debe tener en cuenta la fricción estática modelada. Esto se va a realizar mediante el parámetro b_s, de manera que si el voltaje de entrada es nulo y por lo tanto están presentes los efectos de la fricción estática, el valor de b_s es un número positivo menor que la unidad (en los experimentos se le asignó un valor de 0.2). En cualquier otro caso en el que el voltaje sea distinto de 0, se desprecia la fricción estática, dándole un valor unitario a b_s. Esto se expresaría de la siguiente manera:

$$b_s = \begin{cases} 0{,}2 \ si \ u = 0 \\ 1 \ si \ u \neq 0 \end{cases} \tag{3.35}$$

Sustituyendo estas expresiones en las ecuaciones de las fuerzas (Ec. 3.32 y Ec. 3.33), se obtiene la expresión final del modelo dinámico que permite calcularlas para un determinado voltaje de entrada:

$$f_1 = \{-\frac{l_1(-K_2 + hcos(\theta_2))cos(\frac{l_f^2+l_4^2-l_5^2}{2l_f l_4})l_f}{K_1K_2(h+l_1)l_4} - \frac{K_2 + l_1cos(\theta_2))K_m(\theta_2 - \theta_{2_0})cos(\frac{l_f^2+l_4^2-l_5^2}{2l_f l_4})l_f}{K_1K_2l_4}\}(NK_t\frac{u - NK_t\dot{\theta}_l}{R} - b_s\tau_{ext}) \tag{3.36}$$

$$f_2 = \{\frac{hcos(\frac{l_f^2+l_4^2-l_5^2}{2l_f l_4})l_f}{K_2(h+l_1)l_4} + \frac{K_m(\theta_2 - \theta_{2_0})cos(\frac{l_f^2+l_4^2-l_5^2}{2l_f l_4})l_f}{K_2l_4}\}(NK_t\frac{u - NK_t\dot{\theta}_l}{R} - b_s\tau_{ext}) \tag{3.37}$$

Como conclusión al trabajo realizado, se ha creado un modelo híbrido que implementa la dinámica del actuador así como la estática del resto del manipulador, y se ha desarrollado un modelo simplificado que permite simular la interacción de las falanges del manipulador con el objeto a agarrar, actualizando constantemente en el modelo sus posiciones articulares. Además, es posible conocer las fuerzas que está aplicando la garra sobre dicho objeto sin necesidad de emplear sensores de fuerza, utilizando únicamente la información propioceptiva del servomotor y de las articulaciones pasivas.

Sección 4

Conclusiones y líneas de trabajo futuro

4.1. Conclusiones

En este proyecto se buscaba implementar un modelo dinámico de un manipulador basado en una garra subactuada de dos falanges, a partir de la integración del modelo del actuador y del modelo del propio dedo del manipulador. Por ello, las dos principales contribuciones realizadas en este proyecto se resumen en los siguientes puntos.

Programación de la lectura y envío de datos al actuador

Se ha modificado el *script* de *Matlab* para realizar la lectura de datos del servomotor Dynamixel, de forma que devuelva al usuario todos los valores que ofrece como realimentación este actuador: posición, velocidad, intensidad de corriente, ticks en tiempo real, temperatura y voltaje de entrada. Asimismo, se ha programado el envío de estos datos a *Simulink* en el formato de series de tiempo o *timeseries*. Por otro lado, se han incluido diferentes señales de entrada de voltaje PWM al motor para que el usuario escoja entre ellas, ofreciendo una gran versatilidad a este *script*.

Desarrollo y validación del modelo dinámico del servomotor

A partir de las ecuaciones que se derivan del modelo físico de un motor de corriente continua, se ha desarrollado un modelo dinámico del actuador mediante funciones de transferencia en *Simulink*. Para obtener los valores de los parámetros del servomotor en cuestión, se han utilizado inicialmente los obtenidos en la literatura consultada, posteriormente se ha realizado una serie de experimentos para estudiar el motor en unas condiciones genéricas, y después en unas más específicas, que incluyen estudiar el régimen de velocidades empleado en este proyecto y la necesidad de realimentar un par externo sobre el actuador. Finalmente, se ha validado el modelo realizando un reajuste de los valores hallados, con el fin de maximizar la fiabilidad del modelo.

Cabe destacar que para este apartado se ha diseñado un brazo para el servomotor necesario para realizar los experimentos con un muelle. Este se ha diseñado en *SolidWorks* y se ha impreso en 3D.

Desarrollo del modelo estático de un dedo

Se ha considerado contraproducente realizar un modelado dinámico tradicional de todos los componentes del manipulador, debido a que la mayor parte de la inercia se encuentra en el actuador. Por lo tanto, el modelo desarrollado del dedo es de naturaleza estática. Para realizarlo, se han tenido en cuenta una serie de consideraciones, estudiando un caso en concreto en el que el manipulador esté agarrando una esfera. A partir de estas condiciones de estudio, se ha desarrollado el modelo completo, incluyendo la matriz de transferencia entre el par del actuador y los pares en las articulaciones, el jacobiano del manipulador, y un modelo de interacción de la garra con la esfera.

Implementación del modelo híbrido

Se han implementado juntos el modelo dinámico del actuador con el modelo estático de la garra, con el objetivo de obtener, a partir de una determinada entrada PWM del modelo, la respuesta del manipulador ante una interacción puramente elástica con un objeto esférico. Esta respuesta incluye tanto los valores de fuerza que está ejerciendo el manipulador sobre dicho objeto, como las posiciones angulares que presentan ambas falanges y el actuador en todo momento.

4.2. Líneas de trabajo futuro

Habiendo cumplido todos los objetivos propuestos en el proyecto, se sugieren ciertas líneas de desarrollo para la continuación de este trabajo:

- Validar el modelo implementado mediante experimentos basados en la medición de fuerzas y de las posiciones angulares del manipulador.
- Modificar el modelo para incluir diferentes escenarios: que no haya contacto inicial entre la garra y el objeto, que éste tenga diferentes formas y tamaños, que los puntos de contacto se actualicen, etc.
- Implementar la dinámica del objeto en cuestión, ya que únicamente se ha estudiado una interacción elástica entre manipulador y objeto .

Bibliografía

[1] Joaquin Ballesteros y col. «Proprioceptive Estimation of Forces Using Underactuated Fingers for Robot-Initiated pHRI». En: *Sensors* 20.10 (2020), pág. 2863.

[2] Juan Gandarias y col. «Underactuated Gripper with Forearm Roll Estimation for Human Limbs Manipulation in Rescue Robotics». En: 2019, págs. 5937-5942. DOI: 10.1109/IROS40897.2019.8967953.

[3] Bryan Whitsell y Panagiotis Artemiadis. «Physical human–robot interaction (pHRI) in 6 DOF with asymmetric cooperation». En: *IEEE Access* 5 (2017), págs. 10834-10845.

[4] «Origins of the Servo-Motor [History]». En: *IEEE Industry Applications Magazine* 2.2 (1996), págs. 74-.

[5] Robotis. *ROBOTIS e-Manual*. en. URL: https://emanual.robotis.com/docs/en/dxl/x/xm430-w210/ (visitado 10-08-2020).

[6] Marcos R. O. A. Maximo, Carlos H. C. Ribeiro y Rubens Junqueira Magalhães Afonso. «MODELING OF A POSITION SERVO USED IN ROBOTICS APPLICATIONS». En: 2017.

[7] Tarso Sarzi Kraemer Sartori y André Luís da Silva. «MODELAGEM DE UM SERVOMOTOR PARA ACIONAMENTO DE UM SISTEMA DE GUIMBAIS». En: 2019. DOI: 10.29327/2cab2019.224935.

[8] Alenka Milovanovic, Miroslav Bjekić y Sanja Antic. «Permanent Magnet DC Motor Friction Measurement and Analysis of Friction's Impact». En: *International Review of Electrical Engineering* 6 (2011), págs. 2261-2269.

[9] A. M. Shafei y H. R. Shafei. «Dynamic modeling of planar closed-chain robotic manipulators in flight and impact phases». En: *Mechanism and Machine Theory* 126 (2018), págs. 141-154. ISSN: 0094-114X. DOI: https://doi.org/10.1016/j.mechmachtheory.2018.03.007. URL: http://www.sciencedirect.com/science/article/pii/S0094114X17314684.

[10] Lionel Birglen, Thierry Laliberté y Clément Gosselin. *Underactuated Robotic Hands*. Vol. 40. 2008. ISBN: 978-3-540-77458-7. DOI: 10.1007/978-3-540-77459-4.

[11] Bruno Belzile y Lionel Birglen. «Stiffness Analysis of Underactuated Fingers and Its Application to Proprioceptive Tactile Sensing». En: *IEEE/ASME Transactions on Mechatronics* 21 (2016), págs. 2672-2681. DOI: 10.1109/TMECH.2016.2589546.

Apéndice A

Código programado en *Matlab*

A.1. *Script* para lectura y envío de datos del actuador

Para ejecutar el siguiente código es necesario descargar el kit de desarrollo software *Dynamixel SDK* y seguir las instrucciones incluidas para el entorno de *Matlab* ([5]).

```
%% Para escoger diferentes tipos de entradas de voltaje PWM y leer la ...
   posición angular.
%%% Inicialización
clc;
clear;
lib_name = '';

if strcmp(computer, 'PCWIN')
  lib_name = 'dxl_x86_c';
elseif strcmp(computer, 'PCWIN64')
  lib_name = 'dxl_x64_c';
elseif strcmp(computer, 'GLNX86')
  lib_name = 'libdxl_x86_c';
elseif strcmp(computer, 'GLNXA64')
  lib_name = 'libdxl_x64_c';
elseif strcmp(computer, 'MACI64')
  lib_name = 'libdxl_mac_c';
end

%%% Carga de librerías
if ~libisloaded(lib_name)
    [notfound, warnings] = loadlibrary(lib_name, 'dynamixel_sdk.h', ...
       'addheader', 'port_handler.h', 'addheader', 'packet_handler.h');
end


%%% Direcciones de la tabla de control
ADDR_PRO_TORQUE_ENABLE       = 64;
ADDR_PRO_GOAL_PWM            = 100;
ADDR_PRO_REALTIME_TICK       = 120;
ADDR_PRO_PRESENT_POSITION    = 132;


%%% Versión de protocolo utilizada
PROTOCOL_VERSION            = 2.0;          % Es el usado por el modelo ...
   XM430-210W-R.

%%% Configuración
DXL_ID                      = 1;            % ID del motor (1 ó 2).
BAUDRATE                    = 56900;
DEVICENAME                  = 'COM3';

TORQUE_ENABLE               = 1;            % Variable para habilitar ...
   el torque del motor.
TORQUE_DISABLE              = 0;            % Variable para ...
   deshabilitar el torque del motor.
DXL_MINIMUM_POSITION_VALUE  = 0;            % Posición mínima del motor.
DXL_MAXIMUM_POSITION_VALUE  = 4095;         % Posición máxima del motor.
DXL_MOVING_STATUS_THRESHOLD = 10;           % Umbral de movimiento.

```

```
46 DXL_MINIMUM_PWM_VALUE  = -885;              % Valor mínimo de la señal PWM.
47 DXL_MAXIMUM_PWM_VALUE  = 885;               % Valor máximo de la señal PWM.
48
49 ESC_CHARACTER               = 'e';          % Comando para salir del ...
      bucle de lectura de datos.
50
51 COMM_SUCCESS                = 0;            % Resultado si la ...
      comunicación tiene éxito
52 COMM_TX_FAIL                = -1001;        % Indicador de fallo de ...
      comunicación Tx
53
54 % Initialize PortHandler Structs
55 % Set the port path
56 % Get methods and members of PortHandlerLinux or PortHandlerWindows
57 port_num = portHandler(DEVICENAME);
58
59 % Initialize PacketHandler Structs
60 packetHandler();
61 dxl_comm_result = COMM_TX_FAIL;           % Communication result
62 dxl_error = 0;                             % Dynamixel error
63 dxl_present_position = 0;
64
65
66 %%% Apertura del puerto
67 if (openPort(port_num))
68     fprintf('Succeeded to open the port!\n');
69 else
70     unloadlibrary(lib_name);
71     fprintf('Failed to open the port!\n');
72     input('Press any key to terminate...\n');
73     return;
74 end
75
76
77 %%% Baud rate = 59600 bps
78 if (setBaudRate(port_num, BAUDRATE))
79     fprintf('Succeeded to change the baudrate!\n');
80 else
81     unloadlibrary(lib_name);
82     fprintf('Failed to change the baudrate!\n');
83     input('Press any key to terminate...\n');
84     return;
85 end
86
87
88 %%% Habilitar el torque del motor
89 write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID, ...
      ADDR_PRO_TORQUE_ENABLE, TORQUE_ENABLE);
90 dxl_comm_result = getLastTxRxResult(port_num, PROTOCOL_VERSION);
91 dxl_error = getLastRxPacketError(port_num, PROTOCOL_VERSION);
92 if dxl_comm_result ~= COMM_SUCCESS
93     fprintf('%s\n', getTxRxResult(PROTOCOL_VERSION, dxl_comm_result));
94 elseif dxl_error ~= 0
95     fprintf('%s\n', getRxPacketError(PROTOCOL_VERSION, dxl_error));
96 else
97     fprintf('Dynamixel has been successfully connected \n');
98 end
99
```

```
100 %%% Inicialización de variables para el bucle.
101 i=1;
102 t_initial=now;
103 dxl_goal_pwm = 0;
104 initial_pos = 1020; %Posición inicial, hay que mirarla previamente en ...
       el programa R-Manager de Robotics.
105 opcion = 0; % 0: Entrada PWM constante; 1: Entrada rampa; 2: Entrada ...
       secuencia de impulsos
106
107
108 %%% Bucle de control del motor.
109 while 1
110     if input('Press any key to continue! (or input e to quit!)\n', 's') ...
           == ESC_CHARACTER
111         break;
112     end
113
114     %%% Control de tiempo para que el motor empiece siempre a moverse ...
           en el mismo instante.
115     while 1
116        initial_realtime_tick = read2ByteTxRx(port_num, ...
              PROTOCOL_VERSION, DXL_ID, ADDR_PRO_REALTIME_TICK);
117        fprintf('RealTIME: %03d\n', initial_realtime_tick);
118        if (initial_realtime_tick<50)
119            break;
120        end
121     end
122     dxl_realtime_tick=initial_realtime_tick;
123
124     switch opcion
125         case 0
126             while (dxl_realtime_tick<=10000) % Tiempo en milisegundos
127                 dxl_realtime_tick=read2ByteTxRx(port_num, ...
                       PROTOCOL_VERSION, DXL_ID, ADDR_PRO_REALTIME_TICK);
128                 dxl_goal_pwm = -40;
129                 write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID, ...
                       ADDR_PRO_GOAL_PWM, typecast(int32(dxl_goal_pwm), ...
                       'uint32'));
130
131                 dxl_comm_result = getLastTxRxResult(port_num, ...
                       PROTOCOL_VERSION);
132                 dxl_error = getLastRxPacketError(port_num, ...
                       PROTOCOL_VERSION);
133                 if dxl_comm_result ~= COMM_SUCCESS
134                     fprintf('%s\n', getTxRxResult(PROTOCOL_VERSION, ...
                           dxl_comm_result));
135                 elseif dxl_error ~= 0
136                     fprintf('%s\n', getRxPacketError(PROTOCOL_VERSION, ...
                           dxl_error));
137                 end
138
139                 % Lectura de la posición
140                 dxl_present_position = read4ByteTxRx(port_num, ...
                       PROTOCOL_VERSION, DXL_ID, ADDR_PRO_PRESENT_POSITION);
141                 dxl_comm_result = getLastTxRxResult(port_num, ...
                       PROTOCOL_VERSION);
142                 dxl_error = getLastRxPacketError(port_num, ...
                       PROTOCOL_VERSION);
```

```
143                 if dxl_comm_result ~= COMM_SUCCESS
144                     fprintf('%s\n', getTxRxResult(PROTOCOL_VERSION, ...
                            dxl_comm_result));
145                 elseif dxl_error ~= 0
146                     fprintf('%s\n', getRxPacketError(PROTOCOL_VERSION, ...
                            dxl_error));
147                 end
148
149                 position(i,1)=(initial_pos-dxl_present_position)*0.00153398078; ...
                        %rads
150                 time(i,1)=datetime('now');
151
152                 fprintf('[ID:%03d] i:%03d GoalPWM:%03d   ...
                        PresPos:%03d\n', DXL_ID, i, dxl_goal_pwm,  ...
                        position(i,1));
153                 i=i+1;
154             end
155
156         case 1
157              while (dxl_realtime_tick<=10000)  % Tiempo en milisegundos
158                 dxl_realtime_tick=read2ByteTxRx(port_num, ...
                        PROTOCOL_VERSION, DXL_ID, ADDR_PRO_REALTIME_TICK);
159                 dxl_goal_pwm = -0.0032*dxl_realtime_tick-8;
160                 write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID, ...
                        ADDR_PRO_GOAL_PWM, typecast(int32(dxl_goal_pwm), ...
                        'uint32'));
161
162                 dxl_comm_result = getLastTxRxResult(port_num, ...
                        PROTOCOL_VERSION);
163                 dxl_error = getLastRxPacketError(port_num, ...
                        PROTOCOL_VERSION);
164                 if dxl_comm_result ~= COMM_SUCCESS
165                     fprintf('%s\n', getTxRxResult(PROTOCOL_VERSION, ...
                            dxl_comm_result));
166                 elseif dxl_error ~= 0
167                     fprintf('%s\n', getRxPacketError(PROTOCOL_VERSION, ...
                            dxl_error));
168                 end
169
170                 % Lectura de la posición
171                 dxl_present_position = read4ByteTxRx(port_num, ...
                        PROTOCOL_VERSION, DXL_ID, ADDR_PRO_PRESENT_POSITION);
172                 dxl_comm_result = getLastTxRxResult(port_num, ...
                        PROTOCOL_VERSION);
173                 dxl_error = getLastRxPacketError(port_num, ...
                        PROTOCOL_VERSION);
174                 if dxl_comm_result ~= COMM_SUCCESS
175                     fprintf('%s\n', getTxRxResult(PROTOCOL_VERSION, ...
                            dxl_comm_result));
176                 elseif dxl_error ~= 0
177                     fprintf('%s\n', getRxPacketError(PROTOCOL_VERSION, ...
                            dxl_error));
178                 end
179
180                 position(i,1)=(initial_pos-dxl_present_position)*0.00153398078; ...
                        %rads
181                 time(i,1)=datetime('now');
182
```

```
183                 fprintf('[ID:%03d] i:%03d GoalPWM:%03d   ...
                        PresPos:%03d\n', DXL_ID, i, dxl_goal_pwm,  ...
                        position(i,1));
184                 i=i+1;
185
186             end
187         case 2
188             while (dxl_realtime_tick<=32000)  % Tiempo en milisegundos
189                 dxl_realtime_tick=read2ByteTxRx(port_num, ...
                        PROTOCOL_VERSION, DXL_ID, ADDR_PRO_REALTIME_TICK);
190
191                 if((abs(dxl_realtime_tick-initial_realtime_tick)<=1000)|| ...
                        (abs(dxl_realtime_tick-initial_realtime_tick)>4000 ...
                        && ...
                        abs(dxl_realtime_tick-initial_realtime_tick)<=6000)|| ...
                        (abs(dxl_realtime_tick-initial_realtime_tick)>8000 ...
                        && ...
                        abs(dxl_realtime_tick-initial_realtime_tick)<=10000)|| ...
                        (abs(dxl_realtime_tick-initial_realtime_tick)>12000 ...
                        && ...
                        abs(dxl_realtime_tick-initial_realtime_tick)<=14000)|| ...
                        (abs(dxl_realtime_tick-initial_realtime_tick)>16000 ...
                        && ...
                        abs(dxl_realtime_tick-initial_realtime_tick)<=18000)|| ...
                        (abs(dxl_realtime_tick-initial_realtime_tick)>20000 ...
                        && ...
                        abs(dxl_realtime_tick-initial_realtime_tick)<=22000)|| ...
                        (abs(dxl_realtime_tick-initial_realtime_tick)>24000 ...
                        && ...
                        abs(dxl_realtime_tick-initial_realtime_tick)<=26000)|| ...
                        (abs(dxl_realtime_tick-initial_realtime_tick)>28000 ...
                        && abs(dxl_realtime_tick-initial_realtime_tick)<=30000))
192                     dxl_goal_pwm = 0;
193                 elseif ...
                        (abs(dxl_realtime_tick-initial_realtime_tick)>1000 ...
                        && abs(dxl_realtime_tick-initial_realtime_tick)<=4000)
194                     dxl_goal_pwm = -8;
195                 elseif ...
                        (abs(dxl_realtime_tick-initial_realtime_tick)>6000 ...
                        && abs(dxl_realtime_tick-initial_realtime_tick)<=8000)
196                     dxl_goal_pwm = -12;
197                 elseif ...
                        (abs(dxl_realtime_tick-initial_realtime_tick)>10000 ...
                        && abs(dxl_realtime_tick-initial_realtime_tick)<=12000)
198                     dxl_goal_pwm = -16;
199                 elseif ...
                        (abs(dxl_realtime_tick-initial_realtime_tick)>14000 ...
                        && abs(dxl_realtime_tick-initial_realtime_tick)<=16000)
200                     dxl_goal_pwm = -20;
201                 elseif ...
                        (abs(dxl_realtime_tick-initial_realtime_tick)>18000 ...
                        && abs(dxl_realtime_tick-initial_realtime_tick)<=20000)
202                     dxl_goal_pwm = -24;
203                 elseif ...
                        (abs(dxl_realtime_tick-initial_realtime_tick)>22000 ...
                        && abs(dxl_realtime_tick-initial_realtime_tick)<=24000)
204                     dxl_goal_pwm = -28;
```

```
205                 elseif ...
                        (abs(dxl_realtime_tick-initial_realtime_tick)>26000 ...
                        && abs(dxl_realtime_tick-initial_realtime_tick)<=28000)
206                     dxl_goal_pwm = -32;
207                 elseif ...
                        (abs(dxl_realtime_tick-initial_realtime_tick)>30000 ...
                        && abs(dxl_realtime_tick-initial_realtime_tick)<=32000)
208                     dxl_goal_pwm = -36;
209                 end
210
211                 % Escritura de la entrada PWM
212                 write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID, ...
                        ADDR_PRO_GOAL_PWM, typecast(int32(dxl_goal_pwm), ...
                        'uint32'));
213
214                 dxl_comm_result = getLastTxRxResult(port_num, ...
                        PROTOCOL_VERSION);
215                 dxl_error = getLastRxPacketError(port_num, ...
                        PROTOCOL_VERSION);
216                 if dxl_comm_result ~= COMM_SUCCESS
217                     fprintf('%s\n', getTxRxResult(PROTOCOL_VERSION, ...
                            dxl_comm_result));
218                 elseif dxl_error ~= 0
219                     fprintf('%s\n', getRxPacketError(PROTOCOL_VERSION, ...
                            dxl_error));
220                 end
221
222                 % Lectura de la posición
223                 dxl_present_position = read4ByteTxRx(port_num, ...
                        PROTOCOL_VERSION, DXL_ID, ADDR_PRO_PRESENT_POSITION);
224                 dxl_comm_result = getLastTxRxResult(port_num, ...
                        PROTOCOL_VERSION);
225                 dxl_error = getLastRxPacketError(port_num, ...
                        PROTOCOL_VERSION);
226                 if dxl_comm_result ~= COMM_SUCCESS
227                     fprintf('%s\n', getTxRxResult(PROTOCOL_VERSION, ...
                            dxl_comm_result));
228                 elseif dxl_error ~= 0
229                     fprintf('%s\n', getRxPacketError(PROTOCOL_VERSION, ...
                            dxl_error));
230                 end
231
232                 position(i,1)=(initial_pos-dxl_present_position)*0.00153398078; ...
                        %rads
233                 time(i,1)=datetime('now');
234
235                 fprintf('[ID:%03d] i:%03d GoalPWM:%03d   ...
                        PresPos:%03d\n', DXL_ID, i, dxl_goal_pwm,  ...
                        position(i,1));
236                 i=i+1;
237             end
238     end
239 end
240
241
242 %%% Detener el motor
243 dxl_goal_pwm = 0;
244
```

```
245 %%% Deshabilitar el torque del motor
246 write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID, ...
        ADDR_PRO_TORQUE_ENABLE, TORQUE_DISABLE);
247 dxl_comm_result = getLastTxRxResult(port_num, PROTOCOL_VERSION);
248 dxl_error = getLastRxPacketError(port_num, PROTOCOL_VERSION);
249 if dxl_comm_result ~= COMM_SUCCESS
250     fprintf('%s\n', getTxRxResult(PROTOCOL_VERSION, dxl_comm_result));
251 elseif dxl_error ~= 0
252     fprintf('%s\n', getRxPacketError(PROTOCOL_VERSION, dxl_error));
253 end
254
255 %%% Cierre del puerto COM
256 closePort(port_num);
257
258 %%% Cierre de las librerías utilizadas
259 unloadlibrary(lib_name);
260
261 close all;
262
263
264 %%%% Envío de datos a Simulink
265 %%% Obtención de un vector de posiciones en el tiempo.
266 time.Format = 'HH:mm:ss.SSS';
267
268 d2s    = 24*3600;        % Convertir de días a segundos.
269 tspan  = zeros(length(time(:,1)),1);
270 for ii = 1:length(time)
271     tspan(ii,1) = d2s*(datenum(time(ii,:))-datenum(time(1,:)));
272 end
273 end_time=tspan(length(time))+0.0320;
274 tspan=[tspan; end_time];
275
276
277 %%% Guardar los datos tomados en vectores de tiempo.
278 initial_pos=0;
279 position=[initial_pos; position];
280 ts_position = timeseries(position,tspan);
281
282
283 %% Para leer todos los datos posibles del motor para una determinada ...
        entrada PWM constante
284
285 %%% Inicialización
286 clc;
287 clear;
288 lib_name = '';
289
290 if strcmp(computer, 'PCWIN')
291   lib_name = 'dxl_x86_c';
292 elseif strcmp(computer, 'PCWIN64')
293   lib_name = 'dxl_x64_c';
294 elseif strcmp(computer, 'GLNX86')
295   lib_name = 'libdxl_x86_c';
296 elseif strcmp(computer, 'GLNXA64')
297   lib_name = 'libdxl_x64_c';
298 elseif strcmp(computer, 'MACI64')
299   lib_name = 'libdxl_mac_c';
300 end
```

```
301
302 %%% Carga de librerías
303 if ~libisloaded(lib_name)
304     [notfound, warnings] = loadlibrary(lib_name, 'dynamixel_sdk.h', ...
        'addheader', 'port_handler.h', 'addheader', 'packet_handler.h');
305 end
306
307
308 %%% Direcciones de la tabla de control
309 ADDR_PRO_TORQUE_ENABLE       = 64;
310 ADDR_PRO_GOAL_PWM            = 100;
311 ADDR_PRO_GOAL_POSITION       = 116;
312 ADDR_PRO_REALTIME_TICK       = 120;
313 ADDR_PRO_PRESENT_CURRENT     = 126;
314 ADDR_PRO_PRESENT_VELOCITY    = 128;
315 ADDR_PRO_PRESENT_POSITION    = 132;
316 ADDR_PRO_PRESENT_VOLTAGE     = 144;
317 ADDR_PRO_PRESENT_TEMP        = 146;
318
319 %%% Versión de protocolo utilizada
320 PROTOCOL_VERSION             = 2.0;          % Es el usado por el modelo ...
        XM430-210W-R.
321
322 %%% Configuración
323 DXL_ID                       = 1;            % ID del motor (1 ó 2).
324 BAUDRATE                     = 56900;
325 DEVICENAME                   = 'COM3';
326
327 TORQUE_ENABLE                = 1;            % Variable para habilitar ...
        el torque del motor.
328 TORQUE_DISABLE               = 0;            % Variable para ...
        deshabilitar el torque del motor.
329 DXL_MINIMUM_POSITION_VALUE   = 0;            % Posición mínima del motor.
330 DXL_MAXIMUM_POSITION_VALUE   = 4095;         % Posición máxima del motor.
331 DXL_MOVING_STATUS_THRESHOLD = 10;            % Umbral de movimiento.
332
333 DXL_MINIMUM_PWM_VALUE   = -885;              % Valor mínimo de la señal PWM.
334 DXL_MAXIMUM_PWM_VALUE   = 885;               % Valor máximo de la señal PWM.
335
336 ESC_CHARACTER                = 'e';          % Comando para salir del ...
        bucle de lectura de datos.
337
338 COMM_SUCCESS                 = 0;            % Resultado si la ...
        comunicación tiene éxito
339 COMM_TX_FAIL                 = -1001;        % Indicador de fallo de ...
        comunicación Tx
340
341 % Initialize PortHandler Structs
342 % Set the port path
343 % Get methods and members of PortHandlerLinux or PortHandlerWindows
344 port_num = portHandler(DEVICENAME);
345
346 % Initialize PacketHandler Structs
347 packetHandler();
348
349
350 dxl_comm_result = COMM_TX_FAIL;           % Communication result
351 dxl_error = 0;                             % Dynamixel error
```

```
352 dxl_present_position = 0;
353 dxl_present_velocity = 0;
354 dxl_present_current=0;
355 dxl_present_voltage=0;
356 dxl_present_temp=0;
357
358 %%% Apertura del puerto
359
360 if (openPort(port_num))
361     fprintf('Succeeded to open the port!\n');
362 else
363     unloadlibrary(lib_name);
364     fprintf('Failed to open the port!\n');
365     input('Press any key to terminate...\n');
366     return;
367 end
368
369 %%% Baud rate = 59600 bps
370 if (setBaudRate(port_num, BAUDRATE))
371     fprintf('Succeeded to change the baudrate!\n');
372 else
373     unloadlibrary(lib_name);
374     fprintf('Failed to change the baudrate!\n');
375     input('Press any key to terminate...\n');
376     return;
377 end
378
379 %%% Habilitar el torque del motor
380 write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID, ...
       ADDR_PRO_TORQUE_ENABLE, TORQUE_ENABLE);
381 dxl_comm_result = getLastTxRxResult(port_num, PROTOCOL_VERSION);
382 dxl_error = getLastRxPacketError(port_num, PROTOCOL_VERSION);
383 if dxl_comm_result ~= COMM_SUCCESS
384     fprintf('%s\n', getTxRxResult(PROTOCOL_VERSION, dxl_comm_result));
385 elseif dxl_error ~= 0
386     fprintf('%s\n', getRxPacketError(PROTOCOL_VERSION, dxl_error));
387 else
388     fprintf('Dynamixel has been successfully connected \n');
389 end
390
391 %%% Inicialización de variables para el bucle.
392 i=1;
393 t_initial=now;
394 dxl_goal_pwm = 0;
395 initial_pos = 1020;
396
397 %%% Bucle de control del motor.
398 while 1
399     if input('Press any key to continue! (or input e to quit!)\n', 's') ...
          == ESC_CHARACTER
400         break;
401     end
402
403     %%% Control de tiempo para que el motor empiece siempre a moverse ...
          en el mismo instante.
404     while 1
405         initial_realtime_tick = read2ByteTxRx(port_num, ...
              PROTOCOL_VERSION, DXL_ID, ADDR_PRO_REALTIME_TICK);
```

```
406         fprintf('RealTIME: %03d\n', initial_realtime_tick);
407         if (initial_realtime_tick<50)
408             break;
409         end
410     end
411     dxl_realtime_tick=initial_realtime_tick;
412
413
414     while (dxl_realtime_tick<=10000)  % Tiempo en milisegundos.
415         dxl_realtime_tick=read2ByteTxRx(port_num, PROTOCOL_VERSION, ...
                DXL_ID, ADDR_PRO_REALTIME_TICK);
416
417         %% Escritura de la entrada PWM
418         dxl_goal_pwm = -40;
419         write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID, ...
                ADDR_PRO_GOAL_PWM, typecast(int32(dxl_goal_pwm), 'uint32'));
420
421         dxl_comm_result = getLastTxRxResult(port_num, PROTOCOL_VERSION);
422         dxl_error = getLastRxPacketError(port_num, PROTOCOL_VERSION);
423         if dxl_comm_result ~= COMM_SUCCESS
424             fprintf('%s\n', getTxRxResult(PROTOCOL_VERSION, ...
                    dxl_comm_result));
425         elseif dxl_error ~= 0
426             fprintf('%s\n', getRxPacketError(PROTOCOL_VERSION, dxl_error));
427         end
428
429         % Lectura de datos
430         dxl_present_position = read4ByteTxRx(port_num, ...
                PROTOCOL_VERSION, DXL_ID, ADDR_PRO_PRESENT_POSITION);
431         dxl_present_velocity = read4ByteTxRx(port_num, ...
                PROTOCOL_VERSION, DXL_ID, ADDR_PRO_PRESENT_VELOCITY);
432         dxl_present_voltage = read2ByteTxRx(port_num, PROTOCOL_VERSION, ...
                DXL_ID, ADDR_PRO_PRESENT_VOLTAGE);
433         dxl_present_temp = read1ByteTxRx(port_num, PROTOCOL_VERSION, ...
                DXL_ID, ADDR_PRO_PRESENT_TEMP);
434         dxl_present_current = read2ByteTxRx(port_num, PROTOCOL_VERSION, ...
                DXL_ID, ADDR_PRO_PRESENT_CURRENT);
435
436         dxl_comm_result = getLastTxRxResult(port_num, PROTOCOL_VERSION);
437         dxl_error = getLastRxPacketError(port_num, PROTOCOL_VERSION);
438         if dxl_comm_result ~= COMM_SUCCESS
439             fprintf('%s\n', getTxRxResult(PROTOCOL_VERSION, ...
                    dxl_comm_result));
440         elseif dxl_error ~= 0
441             fprintf('%s\n', getRxPacketError(PROTOCOL_VERSION, dxl_error));
442         end
443
444         % Tratamiento de los datos
445         position(i,1)=(initial_pos-dxl_present_position)*0.00153398078; ...
                % rads
446         time(i,1)=datetime('now');
447         velocity(i,1)=dxl_present_velocity*0.0239809;  % rad/s
448         current(i,1)=dxl_present_current*2.69;  % mA
449         voltage(i,1)=dxl_present_voltage*0.1;   % V
450         temp(i,1)=dxl_present_temp;   % grados Celsius
451
452         fprintf('[ID: %03d] i: %03d GoalPWM: %03d  PresVel: %03d  ...
                PresPos: %03d  PresCurr: %03d  PresVolt: %03d  ...
```

```
            PresTemp:%03d\n', DXL_ID, i, dxl_goal_pwm, velocity(i,1),  ...
            position(i,1), current(i,1), voltage(i,1), temp(i,1));
        i=i+1;
    end
end
%%% Detener el motor
dxl_goal_pwm = 0;

%%% Deshabilitar el torque del motor
write1ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID, ...
   ADDR_PRO_TORQUE_ENABLE, TORQUE_DISABLE);
dxl_comm_result = getLastTxRxResult(port_num, PROTOCOL_VERSION);
dxl_error = getLastRxPacketError(port_num, PROTOCOL_VERSION);
if dxl_comm_result ~= COMM_SUCCESS
    fprintf('%s\n', getTxRxResult(PROTOCOL_VERSION, dxl_comm_result));
elseif dxl_error ~= 0
    fprintf('%s\n', getRxPacketError(PROTOCOL_VERSION, dxl_error));
end

%%% Cierre del puerto COM
closePort(port_num);

%%% Cierre de las librerías utilizadas
unloadlibrary(lib_name);

close all;

%%% Envío de datos a Simulink
%%% Obtención de un vector de posiciones en el tiempo.
time.Format = 'HH:mm:ss.SSS';

d2s    = 24*3600;       % Convertir de días a segundos.
tspan  = zeros(length(time(:,1)),1);
for ii = 1:length(time)
    tspan(ii,1) = d2s*(datenum(time(ii,:))-datenum(time(1,:)));
end
end_time=tspan(length(time))+0.0320;
tspan=[tspan; end_time];


%%% Guardar los datos tomados en vectores de tiempo.
initial_pos=0;
position=[initial_pos; position];
velocity=[0; velocity];
current=[0; current];
voltage=[0; voltage];
temp=[temp; temp(i-1)];

ts_position = timeseries(position,tspan);
ts_velocity = timeseries(velocity,tspan);
ts_current = timeseries(current,tspan);
ts_voltage = timeseries(voltage,tspan);
ts_temp = timeseries(temp,tspan);
```

A.2. Código de las funciones empleadas en el modelo híbrido del manipulador

A.2.1. Matriz de transferencia

Matriz de transferencia inversa y traspuesta

```
function Matriz_T = fcn_matriz_transfer(theta,l_eslabones,l_falanges)

gamma=56*pi/180;
psi=90*pi/180;

f=sqrt(l_eslabones(4)^2+l_eslabones(5)^2+2*l_eslabones(4)* ...
    l_eslabones(5)*cos(theta(3)+gamma));
M=-l_falanges(1)*(l_falanges(1)+2*l_eslabones(2)*cos(theta(2)-psi))+ ...
    f^2-l_eslabones(3)^2-l_eslabones(2)^2;
N=l_falanges(1)*(l_falanges(1)+2*l_eslabones(2)*cos(theta(2)-psi))- ...
    f^2-l_eslabones(3)^2+l_eslabones(2)^2;

cot_beta=(l_eslabones(2)*sin(theta(2)-psi)*sqrt(4*(f^2)* ...
    l_eslabones(3)^2-N^2)+M*(l_falanges(1)+l_eslabones(2)* ...
                  cos(theta(2)-psi)))/-((l_falanges(1)+l_eslabones(2)* ...
                       cos(theta(2)-psi))*sqrt(4*(f^2)*l_eslabones(3)^2-N)+
M*l_eslabones(2)*sin(theta(2)-psi));

h=l_eslabones(2)*(cos(theta(2)-psi)-sin(theta(2)-psi)*cot_beta);

Matriz_T=[1, -((h)/(h+l_falanges(1))); 0, 1];
Matriz_T=(inv(Matriz_T))';
```

Transformación previa 5B-4B

```
function tau_a_transf = fcn_trasnformacion_taua(tau_a,l_eslabones,theta)

gamma=56*pi/180;
f=sqrt(l_eslabones(4)^2+l_eslabones(5)^2+2*l_eslabones(4)* ...
    l_eslabones(5)*cos(theta(3)+gamma));
alpha=acos((f^2+l_eslabones(4)^2-l_eslabones(5)^2)/(2*f*l_eslabones(4)));

tau_a_transf=(tau_a/l_eslabones(4))*cos(alpha)*f;
```

Vector t

```
function t = fcn_matriz_t(theta,tau_a, Kmuelle)
t2=Kmuelle*theta(2)*tau_a;
t=[tau_a,t2]';
```

A.2.2. Jacobiano

Jacobiano inverso y traspuesto

```
function J = jacobiano_t_inv(theta,ptos_contacto, l_falanges)
J=[1/ptos_contacto(1), -(ptos_contacto(2)+l_falanges(1)*cos(theta(2)))/ ...
    (ptos_contacto(1)*ptos_contacto(2)); 0, 1/ptos_contacto(2)]
```

A.2.3. Interacción con el entorno

Actualización de la posición angular

```
function theta_actual = fcn_nueva_theta(fuerza,theta,Kesfera,ptos_contacto)

theta_actual=theta;
delta_x=[0,0];
delta_theta=[0,0];
delta_x(1)=fuerza(1)/Kesfera;
delta_x(2)=fuerza(2)/Kesfera;
delta_theta(1)=atan(delta_x(1)/ptos_contacto(1));
delta_theta(2)=atan(delta_x(2)/ptos_contacto(2));

theta_actual(1)=theta(1)+delta_theta(1);
theta_actual(2)=theta(2)+delta_theta(2);
theta_actual=[theta_actual(1) theta_actual(2) theta(3)];
```

Apéndice B

Cálculo de la constante de elasticidad del muelle

Para obtener la constante de elasticidad del muelle empleado en los experimentos, se ha recurrido al método estático. Este consiste en fijar uno de los extremos del muelle, mientras que del otro se cuelgan diferentes masas. Para cada una de ellas se mide el alargamiento producido en el muelle (Tabla B.1). Con estas mediciones se puede dibujar una gráfica que muestre el alargamiento producido en función de la fuerza que ejerce una determinada masa sobre el muelle. Según la Ley de Hooke, la pendiente de la interpolación lineal de dicha gráfica (Fig. B.1) es la constante de elasticidad del muelle. Por lo tanto, se puede afirmar que el valor de K_m para el muelle en cuestión es de 163.02 N/m.

Tabla B.1: Mediciones para el cálculo de la constante de elasticidad del muelle (K_m)

Masa (Kg)	**Fuerza (N)**	**Estiramiento del muelle (m)**
0.035	0.343	0.0015
0.06	0.588	0.0035
0.09	0.882	0.0055
0.11	1.078	0.0065
0.124	1.2152	0.0075
0.13	1.274	0.0085
0.16	1.568	0.0105
0.195	1.911	0.0115
0.235	2.303	0.0143
0.24	2.352	0.0145
0.28	2.744	0.0165
0.315	3.087	0.0185

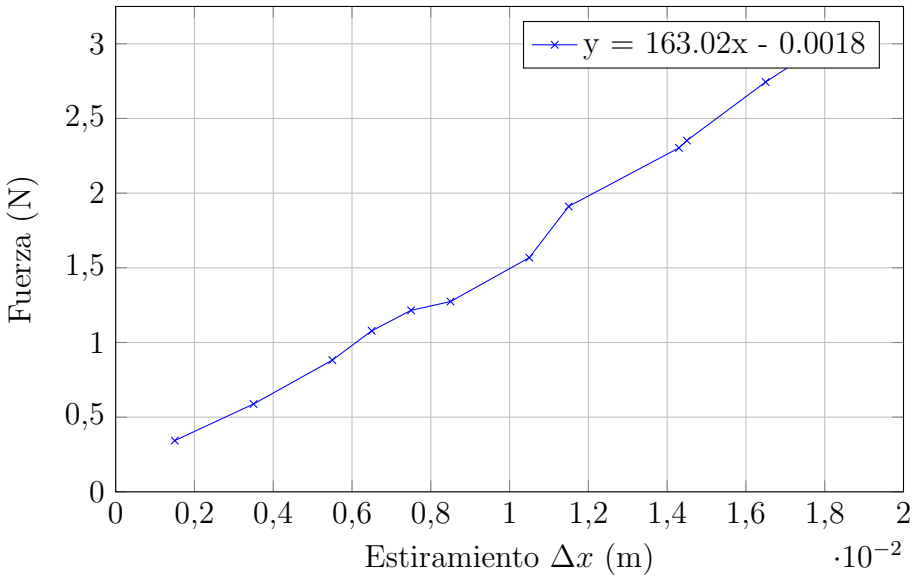


Figura B.1: Relación Estiramiento-Fuerza

Apéndice C

Validación y reajuste de parámetros para un modelado dinámico tradicional del actuador

En este Anexo se muestra paso a paso cómo validar y refinar los parámetros de un modelo dinámico tradicional para un motor DC. Para ello, se van a utilizar diferentes entradas PWM de tipo escalón, y se va a obtener la velocidad angular en el eje de carga del motor para cada una de ellas.

Tabla C.1: Valores originales de los parámetros del motor

Parámetro	**Valor**
Resistencia de armadura (R)	4.6 Ω
Constante de par (K_t)	0.007145 Nm/A
Momento de inercia del eje de carga (J)	0.009182685 kgm^2
Coeficiente de fricción del eje de carga (b)	0.010592355 Nms
Rendimiento (η)	76.97 %

Inicialmente, se van a emplear los valores de los parámetros hallados en la literatura consultada ([7]) para el mismo modelo de motor que el empleado en este proyecto. Dichos valores se muestran en la Tabla C.1.

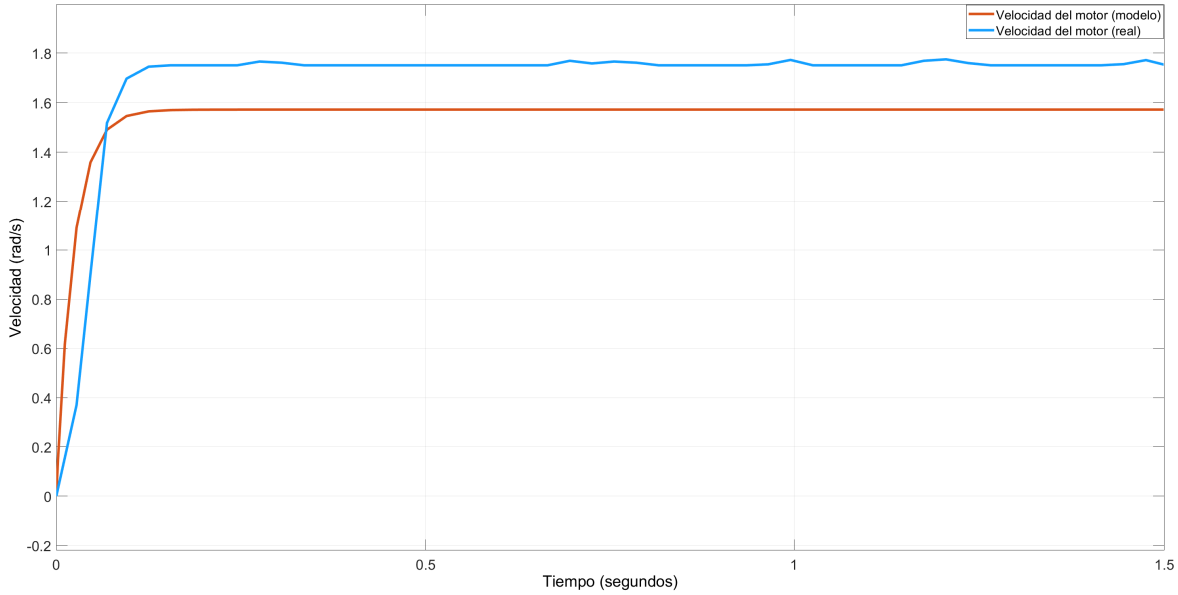


Figura C.1: Respuesta en velocidad (rad/s) del motor real (en azul) frente a la respuesta obtenida con el modelo (en naranja) ante una entrada PWM de 200 niveles

En las Figuras C.1, C.2 y C.3 se muestra la evolución de la velocidad angular del motor para una entrada PWM de 200, 500 y 700 niveles de amplitud, respectivamente. Además, en cada gráfica se compara la respuesta real obtenida con la teórica aportada por el modelo para el mismo voltaje de entrada. En los tres casos, la respuesta real y la teórica comparten una forma parecida, aunque el modelo presenta un error en régimen permanente que disminuye cuanto mayor es la amplitud del voltaje de entrada (Tab. C.2). De esta forma, se puede comprobar que el error relativo máximo se produce en el primer caso (PWM de 200 niveles), con un valor del 10.22 %.

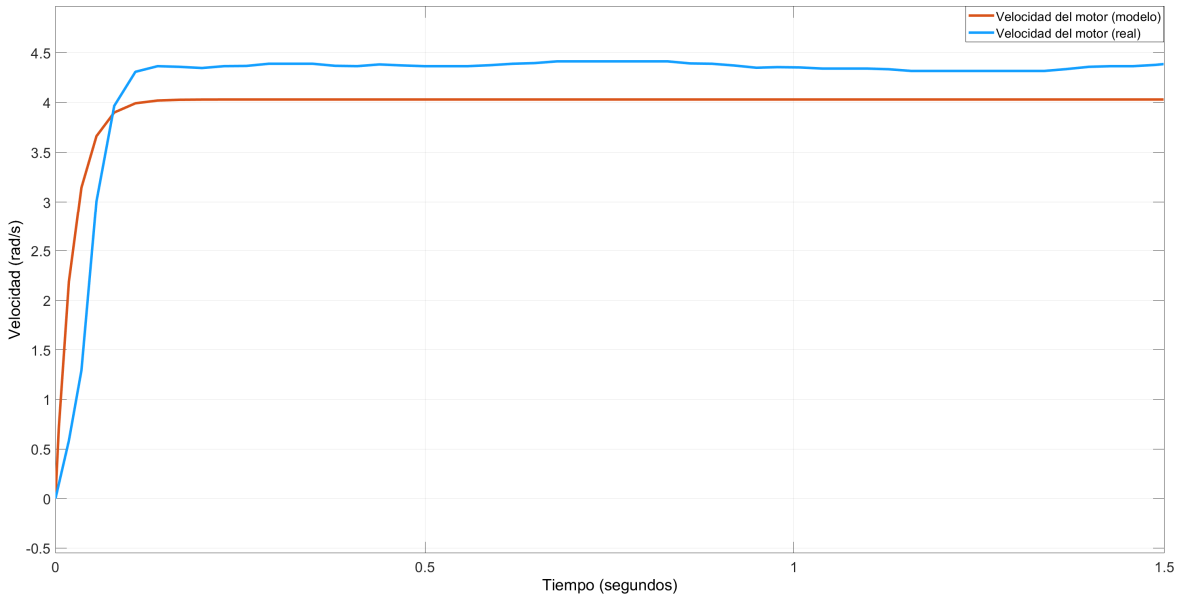


Figura C.2: Respuesta en velocidad (rad/s) del motor real (en azul) frente a la respuesta obtenida con el modelo (en naranja) ante una entrada PWM de 500 niveles

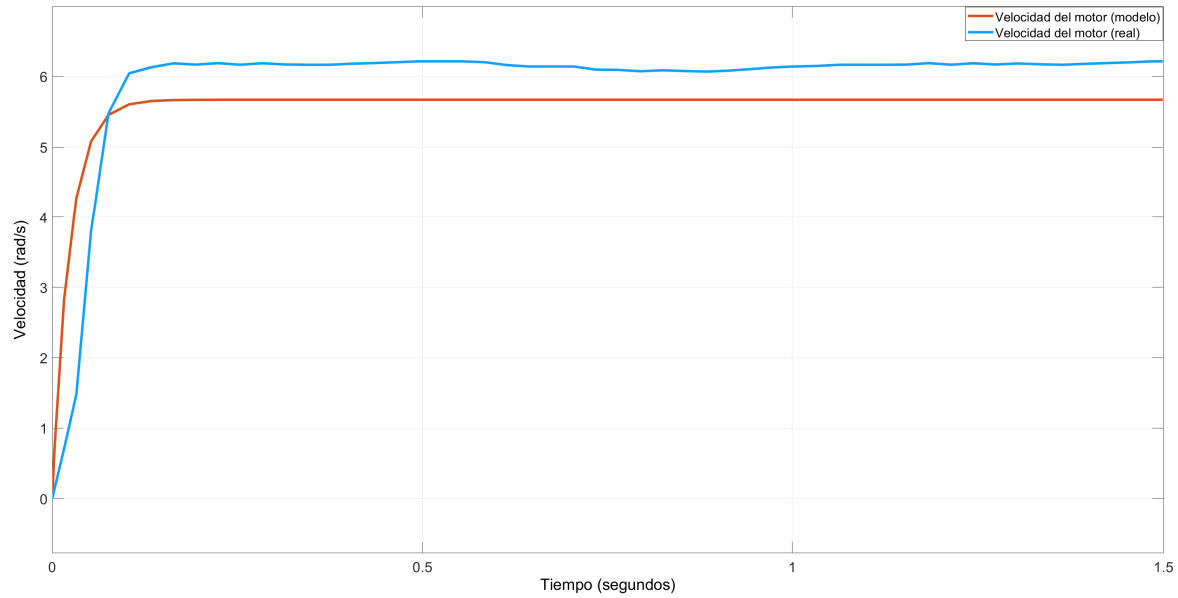


Figura C.3: Respuesta en velocidad (rad/s) del motor real (en azul) frente a la respuesta obtenida con el modelo (en naranja) ante una entrada PWM de 700 niveles

Tabla C.2: Error de la respuesta del modelo antes del reajuste de parámetros

PWM (niveles)	**Error absoluto (rad/s)**	**Error relativo (%)**
200	0.179	10.22
500	0.3584	8.17
700	0.47	7.65

A partir de estos resultados, se utiliza la herramienta *Parameter Estimation* de *Simulink* para realizar un afinamiento de los parámetros con el fin de disminuir el error entre la respuesta del modelo y la del motor real.

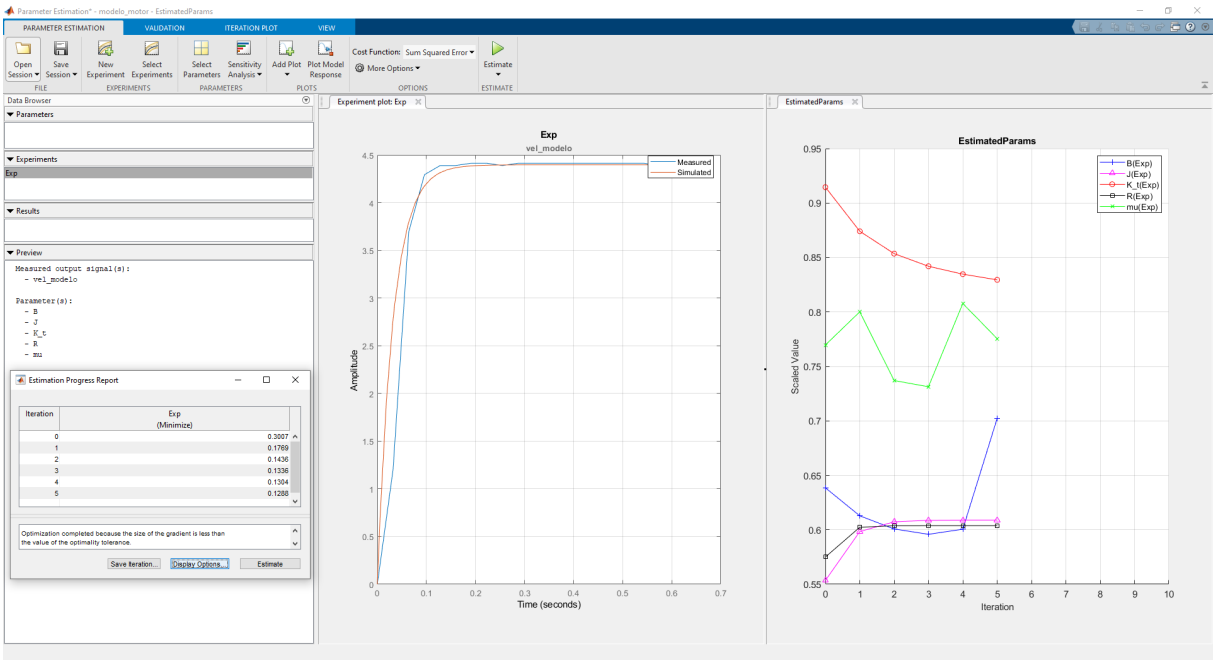


Figura C.4: Estimación de parámetros obtenida con la función de costes basada en la mínima suma del error cuadrático

En las Figuras C.4 y C.5 se muestran los resultados obtenidos tras calibrar el modelo ejecutando esta herramienta con el objetivo de minimizar la suma del error cuadrático y del error absoluto, respectivamente. En ambos casos, el ajuste del modelo a la respuesta real es satisfactorio. Por otra parte, en la parte derecha de estas figuras se observa la evolución de los parámetros a lo largo de las iteraciones, con sus valores escalados en la gráfica.

Con el objetivo de determinar qué función de costes ha aportado un mejor afinamiento del modelo, se han comparado las respuestas del modelo al aplicar ambas funciones de coste, y con sus valores en régimen permanente se ha elaborado la Tabla C.3. En ella aparecen calculados, para dicho régimen, tanto el error absoluto como el relativo cometidos en cada caso. De esta forma, se puede afirmar que el método de estimación que aporta un mejor calibrado del modelo es el basado en la búsqueda de la mínima suma del error absoluto y, por este motivo, se mantiene la estimación obtenida con esta función de costes. Las respuestas del modelo reajustado con esta función de costes ante una entrada PWM de 200, 500 y 700 niveles se muestran en las Figuras C.6, C.7 y C.8, respectivamente,

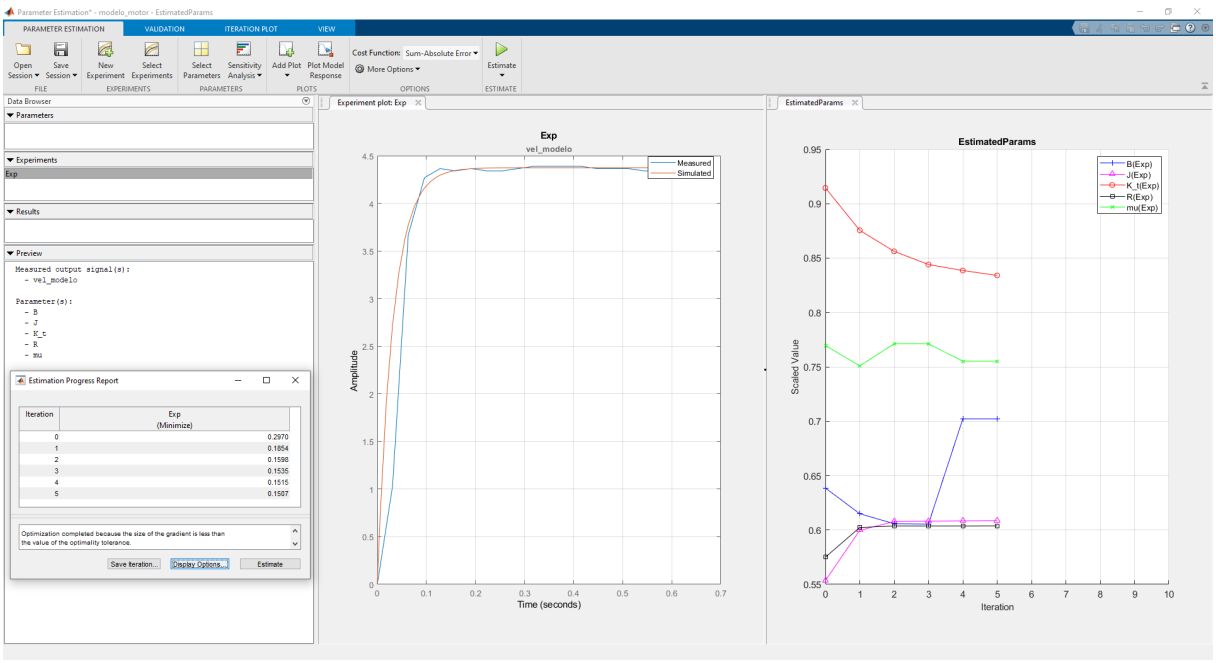


Figura C.5: Estimación de parámetros obtenida con la función de costes basada en la mínima suma del error absoluto

Tabla C.3: Error, después del reajuste, de la respuesta del modelo

Función de costes	**PWM (niveles)**	**Error absoluto (rad/s)**	**Error relativo (%)**
Mínima suma del error cuadrático	200	0.03629	2.07
	500	0.0552	1.27
	700	0.06877	1.12
Mínima suma del error absoluto	200	0.03201	1.83
	500	0.0148	0.34
	700	0.01333	0.22

y en todas se puede comprobar cómo el error de la respuesta del modelo ha disminuido considerablemente, mejorando así la fiabilidad del modelado.

Tabla C.4: Valores reajustados de los parámetros del motor

Parámetro	**Valor**
Resistencia de armadura (R)	4.6 Ω
Constante de par (K_t)	0.0065 Nm/A
Momento de inercia del eje de carga (J)	0.00990708 kgm^2
Coeficiente de fricción del eje de carga (b)	0.01142741 Nms
Rendimiento (η)	75.52 %

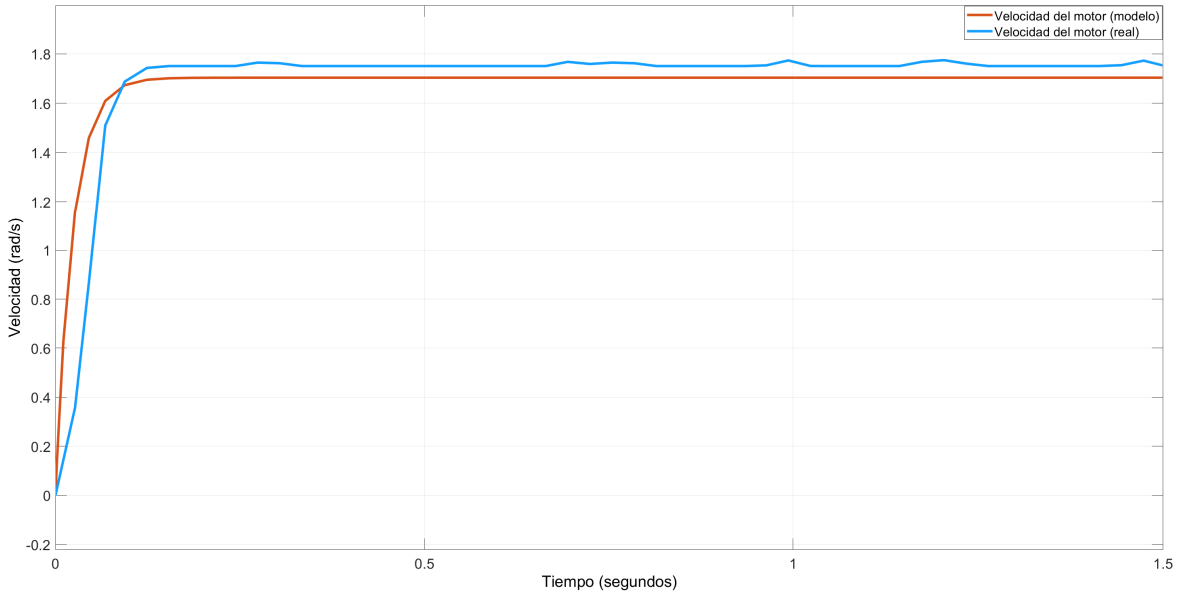


Figura C.6: Respuesta en velocidad (rad/s) del motor real (en azul) frente a la respuesta obtenida con el modelo (en naranja) ante una entrada PWM de 200 niveles, tras el reajuste

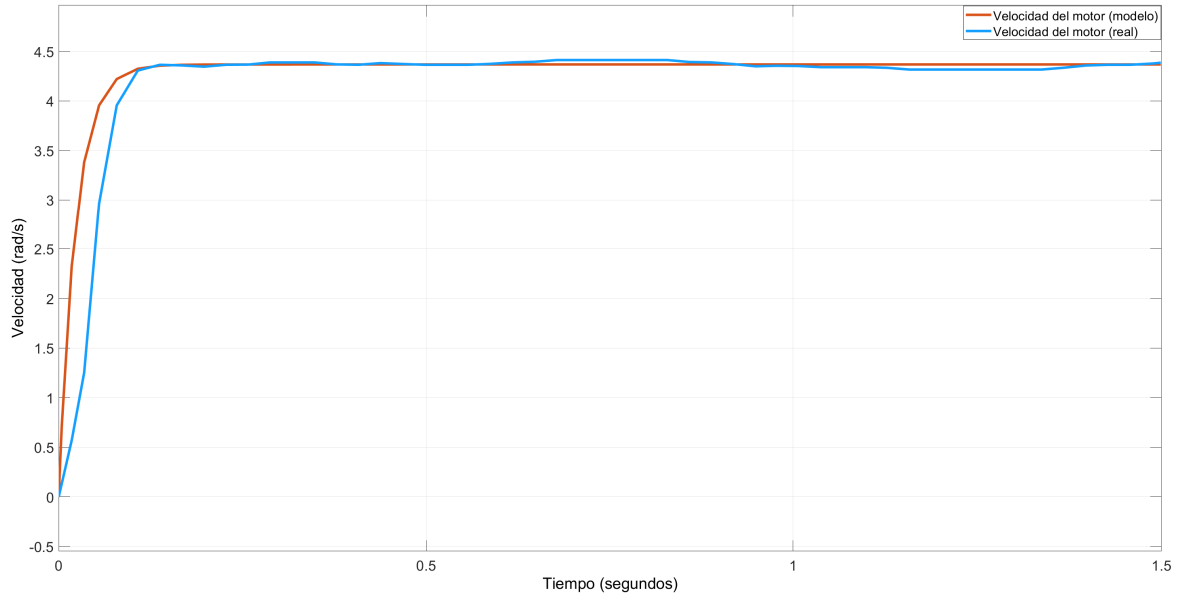


Figura C.7: Respuesta en velocidad (rad/s) del motor real (en azul) frente a la respuesta obtenida con el modelo (en naranja) ante una entrada PWM de 500 niveles, tras el reajuste

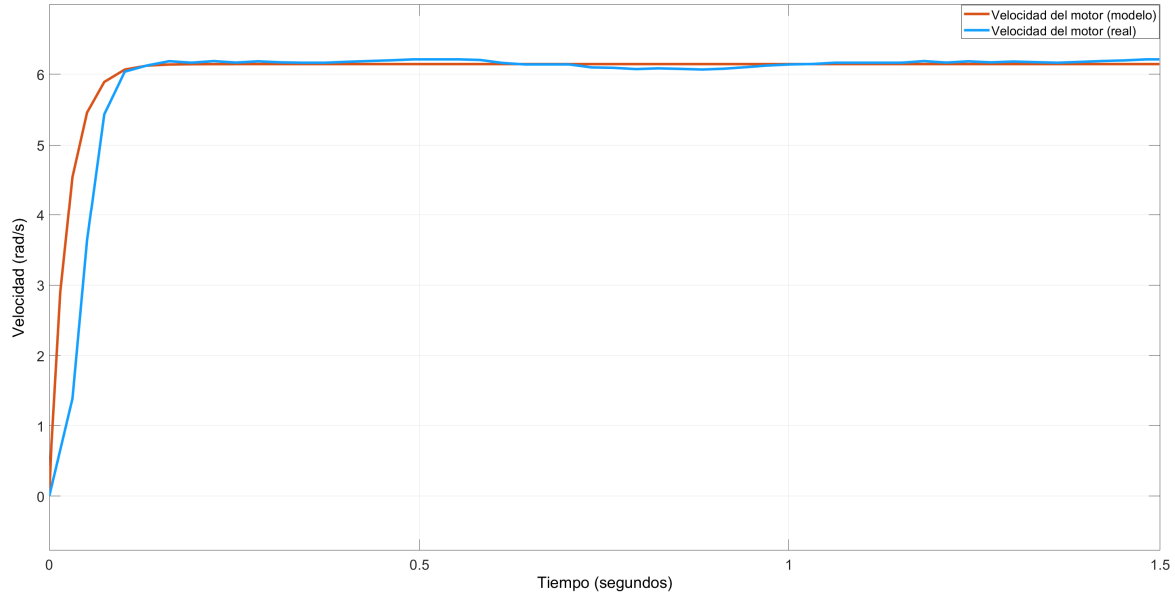


Figura C.8: Respuesta en velocidad (rad/s) del motor real (en azul) frente a la respuesta obtenida con el modelo (en naranja) ante una entrada PWM de 700 niveles, tras el reajuste

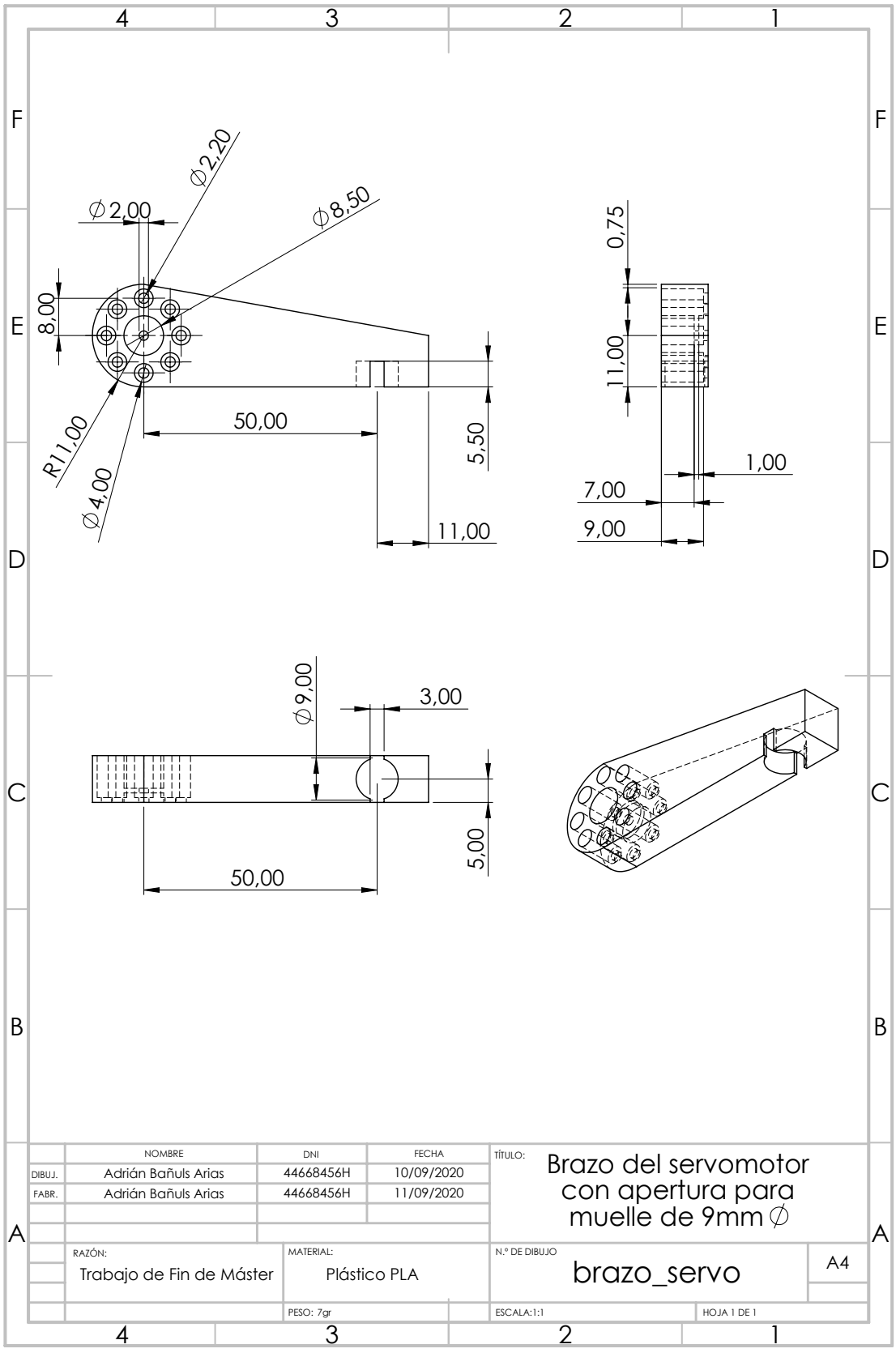
En la Tabla C.4 se observan los nuevos valores de los parámetros del motor. Cabe destacar que se introdujeron ciertas restricciones lógicas en el software para la estimación de los nuevos valores. Estas fueron las siguientes:

- Todos los parámetros son de naturaleza física, por lo que su valor debe ser siempre mayor o igual que 0.
- Se ha supuesto que al tratarse del mismo modelo de motor, la variación de los parámetros no puede ser mayor de un $\pm 10\,\%$ respecto a su valor inicial. Esto se ha hecho para todos los parámetros salvo para el rendimiento del motor, cuyo valor se ha restringido a un rango de $\pm 5\,\%$, ya que tratándose del mismo modelo de motor se asume que el rendimiento debe ser muy parecido.

Con los nuevos valores de los parámetros del motor, se puede concluir que se ha desarrollado un modelo preciso del mismo, que permitirá obtener el torque de salida directamente a partir del voltaje PWM de entrada, e introducir este torque en el modelo dinámico de la garra subactuada para obtener las fuerzas externas realizadas.

Apéndice D

Plano del brazo del servomotor


4
3
2
1
F
E
D
C
B
A
∅2,20
∅2,00
∅8,50
8,00
R11,00
∅4,00
50,00
5,50
11,00
0,75
11,00
1,00
7,00
9,00
∅9,00
3,00
5,00
50,00
NOMBRE
DNI
FECHA
TÍTULO:
DIBUJ.
Adrián Bañuls Arias
44668456H
10/09/2020
FABR.
Adrián Bañuls Arias
44668456H
11/09/2020
Brazo del servomotor con apertura para muelle de 9mm ∅
RAZÓN:
Trabajo de Fin de Máster
MATERIAL:
Plástico PLA
N.º DE DIBUJO
brazo_servo
A4
PESO: 7gr
ESCALA:1:1
HOJA 1 DE 1

Apéndice E

Fundamentos del modelo dinámico del manipulador basado en el principio de Gibbs–Appell

En la actualidad, numerosas investigaciones tratan de adaptar los métodos de modelado dinámico tradicionales (formulaciones de Lagrange-Euler y Newton-Euler) al desarrollo de modelos dinámicos para manipuladores de cadena cinemática cerrada. Sin embargo, la mayoría de ellos sufren de grandes cargas computacionales, consecuencias de combinar el uso de ecuaciones diferenciales y algebraicas para la obtención del modelo. Por este motivo, en este trabajo se ha estudiado el posible desarrollo del modelo mediante una de las técnicas alternativas que ofrece el estado del arte: un algoritmo recursivo basado en el principio de Gibbs–Appell ([9]). Aunque finalmente se ha descartado su implementación, se incluye aquí el estudio realizado y el código programado en *Matlab* para el cálculo de la matriz de inercias del manipulador, como un inicio del desarrollo de dicho modelo.

E.1. Desarrollo matemático

Ecuación del modelo dinámico del manipulador de cadena cinemática abierta:

$$I_F^O \ddot{\theta}_F^O = R_F^O \tag{E.1}$$

Se separa la última articulación y se incluyen dos fuerzas de restricción, f_{X_1} y f_{X_2}, obteniendo la expresión del modelo dinámico para una cadena cinemática cerrada:

$$I_F^O \ddot{\theta}_F^O = R_F^O + {}^{n+1}\mathrm{j}_{1,:}^T f_{X_1} + {}^{n+1}\mathrm{j}_{2,:}^T f_{X_2} \tag{E.2}$$

Donde sus términos se expresan como:

$$\ddot{\theta}_F^O = \begin{bmatrix} \ddot{q}_1 & \ddot{q}_2 & \ddot{q}_3 & \ddot{q}_4 & \ddot{q}_5 & \ddot{X}_1 & \ddot{X}_2 \end{bmatrix}^T \tag{E.3a}$$

$$R_F^O = \begin{bmatrix} R_{q_1} & R_{q_2} & R_{q_3} & \ddot{q}_4 & \ddot{q}_5 & \ddot{X}_1 & \ddot{X}_2 \end{bmatrix}^T \tag{E.3b}$$

$${}^{i}\mathrm{j}_{j,k} = \frac{\partial({}^{ref}\mathrm{r}_{O_i}^T \cdot {}^{ref}\mathrm{X}_j)}{\partial \theta_k} \tag{E.3c}$$

Esta expresión evoluciona hasta:

$$I_F^C \ddot{\theta}_F^C = R_F^C \tag{E.4}$$

Donde la matriz de inercia I_F^C tiene la siguiente forma:

$$I_F^C = \begin{bmatrix} a+b+c & a+b+c & a+b+c & a+b+c & a+b & d+e & d+e & -{}^6\mathrm{j}_{1,1} & -{}^6\mathrm{j}_{2,1} \\ a+b+c & a+b+c & a+b+c & a+b+c & a+b & d+e & d+e & -{}^6\mathrm{j}_{1,2} & -{}^6\mathrm{j}_{2,2} \\ a+b+c & a+b+c & a+b+c & a+b+c & a+b & d+e & d+e & -{}^6\mathrm{j}_{1,3} & -{}^6\mathrm{j}_{2,3} \\ a+b+c & a+b+c & a+b+c & a+b+c & a+b & d+e & d+e & -{}^6\mathrm{j}_{1,4} & -{}^6\mathrm{j}_{2,4} \\ a+b & a+b & a+b & a+b & a & d & d & -{}^6\mathrm{j}_{1,5} & -{}^6\mathrm{j}_{2,5} \\ d+e & d+e & d+e & d+e & d & M_{tot} & 0 & 0 & -{}^6\mathrm{j}_{2,6} \\ d+e & d+e & d+e & d+e & d & 0 & M_{tot} & -{}^6\mathrm{j}_{1,7} & 0 \\ {}^6\mathrm{j}_{1,1} & {}^6\mathrm{j}_{1,2} & {}^6\mathrm{j}_{1,3} & {}^6\mathrm{j}_{1,4} & {}^6\mathrm{j}_{1,5} & 0 & {}^6\mathrm{j}_{1,7} & 0 & 0 \\ {}^6\mathrm{j}_{2,1} & {}^6\mathrm{j}_{2,2} & {}^6\mathrm{j}_{2,3} & {}^6\mathrm{j}_{2,4} & {}^6\mathrm{j}_{2,5} & {}^6\mathrm{j}_{2,6} & 0 & 0 & 0 \end{bmatrix} \tag{E.5a}$$

$$R_F^C = \begin{bmatrix} f+g & f+g & f+g & f+g & g & h & h & -{}^6\dot{\mathrm{j}}_{1,:} \cdot \dot{\theta}_F^O & -{}^6\dot{\mathrm{j}}_{2,:} \cdot \dot{\theta}_F^O \end{bmatrix}^T \tag{E.5b}$$

Elementos de la matriz de inercias y de la matriz del resto de términos dinámicos:

$$a = {}^j\vec{\mathrm{x}}_{j,3}^T \cdot {}^j\sigma_k \, {}^k\vec{\mathrm{x}}_{k,3} \tag{E.6a}$$

$$b = -{}^j\vec{\mathrm{x}}_{j,3}^T \cdot {}^j\psi_k \, {}^k\vec{\mathrm{x}}_{k,3} \tag{E.6b}$$

$$c = -{}^j\vec{\mathrm{x}}_{j,3}^T \cdot {}^jU_k \, {}^k\vec{\mathrm{x}}_{k,3} \tag{E.6c}$$

$$d = {}^j\vec{\mathrm{x}}_{j,3}^T \cdot {}^j\xi_{ref} \, {}^{ref}\vec{\mathrm{X}}_{k-n} \tag{E.6d}$$

$$e = {}^j\vec{\mathrm{x}}_{j,3}^T \cdot {}^j\gamma_{ref} \, {}^{ref}\vec{\mathrm{X}}_{k-n} \tag{E.6e}$$

$$f = \sum_{i=j+1}^{n} -\frac{\partial {}^i\vec{\mathrm{r}}_{O_i}^T}{\partial \ddot{q}_j} \cdot {}^i\vec{\mathrm{S}}_i \tag{E.6f}$$

$$g = \sum_{i=j}^{n} -\frac{\partial \, {}^i\dot{\vec{\omega}}_i^T}{\partial \ddot{q}_j} \cdot {}^i\vec{\mathrm{T}}_i \tag{E.6g}$$

$$h = \sum_{i=-1}^{n} -\frac{\partial \, {}^i\ddot{\vec{\mathrm{r}}}_{O_i}^T}{\partial \ddot{X}_j} \cdot {}^i\vec{\mathrm{S}}_i \tag{E.6h}$$

Expresión de las variables empleadas para la simplificación de las ecuaciones anteriores:

$${}^j\sigma_k = \sum_{i=max(j,k)}^{n} \left[{}^j\mathrm{R}_i \cdot (B_{2i} + B_{3i}) \cdot {}^i R_k \right] \tag{E.7a}$$

$${}^j\psi_k = \sum_{i=max(j+1,k)}^{n} \left[{}^j\tilde{\mathrm{r}}_{O_i/O_j} \cdot {}^j\mathrm{R}_i B_{1i} \cdot {}^i R_k \right] \tag{E.7b}$$

$${}^jU_k = \sum_{t=k}^{n-1} \left[({}^j\xi_{t^+} + {}^j\gamma_t) \cdot {}^j\tilde{\mathrm{r}}_{O_{t+1}/O_t} \cdot {}^t R_k \right] \tag{E.7c}$$

$${}^j\xi_{t^+} = \sum_{i=max(j,t+1)}^{n} \left[{}^j\mathrm{R}_i \cdot B_{1i} \cdot {}^i R_t \right] \tag{E.7d}$$

$${}^j\gamma_t = \sum_{i=max(j+1,t+1)}^{n} \left[{}^j\tilde{\mathrm{r}}_{O_i/O_j} \cdot B_{0i} \cdot {}^j R_t \right] \tag{E.7e}$$

$$ {}^{j}\xi_{ref} = \sum_{i=j}^{n} \left[{}^{j}\mathrm{R}_i \cdot B_{1i} \cdot \ {}^{i}R_{ref} \right] \tag{E.7f} $$

$$ {}^{j}\gamma_{ref} = \sum_{i=j+1}^{n} \left[{}^{j}\tilde{\mathrm{r}}_{O_i/O_j} \cdot B_{0i} \cdot \ {}^{j}R_{ref} \right] \tag{E.7g} $$

$$ {}^{i}\vec{\mathrm{S}}_i = B_{0i} \ {}^{i}\ddot{\vec{\mathrm{r}}}_{O_{v,i}} - B_{1i} \ {}^{i}\dot{\vec{\omega}}_{v,i} - {}^{i}\tilde{\omega}_i \ B_{1i} \ {}^{i}\vec{\omega}_i \tag{E.7h} $$

$$ {}^{i}\vec{\mathrm{T}}_i = B_{1i} \ {}^{i}\ddot{\vec{\mathrm{r}}}_{O_{v,i}} + (B_{2i} + B_{3i}) \ {}^{i}\dot{\vec{\omega}}_{v,i} + {}^{i}\tilde{\omega}_i \ (B_{2i} + B_{3i}) \ {}^{i}\vec{\omega}_i \tag{E.7i} $$

Expresión de la variable M_{tot}, la cual representa la masa total del manipulador:

$$ M_{tot} = \sum_{i=1}^{n} B_{0i} \tag{E.8} $$

Expresiones de las variables B_{ji}:

$$ B_{0i} = \int_0^{l_i} \mu_i \ \partial\eta_i \tag{E.9a} $$

$$ B_{1i} = \int_0^{l_i} \mu_i \ {}^{i}\tilde{\mathrm{r}}_{Q/O_i} \partial\eta_i \tag{E.9b} $$

$$ B_{2i} = \int_0^{l_i} \mu_i \ {}^{i}\tilde{\mathrm{r}}^{T}_{Q/O_i} \ {}^{i}\tilde{\mathrm{r}}_{Q/O_i} \ \partial\eta_i \tag{E.9c} $$

$$ B_{3i} = \int_0^{l_i} J_i \ \partial\eta_i \tag{E.9d} $$

Dónde ${}^{i}\tilde{\mathrm{r}}_{Q/O_i}$ es una matriz antisimétrica relacionada con el vector ${}^{i}\vec{\mathrm{r}}_{Q/O_i}$, el cual se expresa cómo:

$$ {}^{i}\vec{\mathrm{r}}_{Q/O_i} = \eta \ {}^{i}\vec{\mathrm{x}}_{i,1} \tag{E.10} $$

Siendo η la distancia entre los puntos O_i y Q, y el vector ${}^{i}\vec{\mathrm{x}}_{i,1} = \{1 \ 0 \ 0\}^T$.

Vamos ahora con los términos relativos a la cinemática del manipulador que aparecen en las Ecuaciones E.7h y E.7i.

$$ {}^{i}\ddot{\vec{\mathrm{r}}}_{O_i} = {}^{i}\ddot{\vec{\mathrm{r}}}_{O_{s,i}} + {}^{i}\ddot{\vec{\mathrm{r}}}_{O_{v,i}} \tag{E.11a} $$

$$ {}^{i}\dot{\vec{\omega}}_i = {}^{i}\dot{\vec{\omega}}_{s,i} + {}^{i}\dot{\vec{\omega}}_{v,i} \tag{E.11b} $$

Y cada uno de estos términos se obtiene de la siguiente manera:

$$ {}^{i}\ddot{\vec{\mathrm{r}}}_{O_{s,i}} = \sum_{j=1}^{3} \left[{}^{i}R_{ref}\ {}^{ref}\vec{X}_j \ddot{X}_j \right] + \sum_{k=-1}^{i-1} \left[{}^{i}R_k ({}^{k}\dot{\vec{\omega}}_{s,k} \times {}^{k}\vec{\mathrm{r}}_{O_{k+1}/O_k}) \right] \tag{E.12a} $$

$$ {}^{i}\ddot{\vec{\mathrm{r}}}_{O_{v,i}} = {}^{i}R_{ref}\ {}^{ref}\vec{X}_2 g + \sum_{k=-1}^{i-1} \left[{}^{i}R_k ({}^{k}\dot{\vec{\omega}}_{v,k} \times {}^{k}\vec{\mathrm{r}}_{O_{k+1}/O_k} + {}^{k}\vec{\omega}_k \times ({}^{k}\vec{\omega}_k \times {}^{k}\vec{\mathrm{r}}_{O_{k+1}/O_k})) \right] \tag{E.12b} $$

$$ {}^{i}\dot{\vec{\omega}}_{s,i} = \sum_{k=-1}^{i} \left[{}^{i}R_k\ {}^{k}\vec{x}_{k,3}\ \ddot{\theta}_k \right] \tag{E.12c} $$

$$ {}^{i}\dot{\vec{\omega}}_{v,i} = \sum_{k=-1}^{i-1} \left[{}^{i}R_k\ {}^{k}\vec{\omega}_k \times {}^{i}R_{k+1}\ {}^{k+1}\vec{x}_{k+1,3}\ \dot{\theta}_{k+1} \right] \tag{E.12d} $$

Donde ${}^{k}\vec{\mathrm{r}}_{O_{k+1}/O_k} = l_k\ {}^{k}\vec{x}_{k,1}$, ${}^{ref}\vec{X}_1 = \{1\ 0\ 0\}^T$, ${}^{ref}\vec{X}_2 = \{0\ 1\ 0\}^T$, ${}^{ref}\vec{X}_3 = \{0\ 0\ 1\}^T$, y ${}^{k}\vec{x}_{k,3} = \{0\ 0\ 1\}^T$.

Llegados a este punto, ya tendríamos todos los elementos necesarios para realizar el cálculo del modelo dinámico de un manipulador de cadena cerrada.

E.2. Código en *Matlab* para el cálculo de I_F^C

```
%% Inicialización
clear all;
close all;

syms q2i qa;

g=9.8;  % Gravedad

% Ejes de cada articulación
xj = [1 0 0; 0 1 0; 0 0 1];

% Masas de los eslabones
m = [9.05 13.6 5.06 3.2]*10^-3;  % Kg
Mtot = sum(m);  % Masa total de todos los eslabones

% Longitudes de los eslabones
l = [40 20 60 25 27.8]*10^-3;  % m

% Ángulos conocidos
psi = 90*pi/180;  % rad
gamma = 56*pi/180;  % rad

% Distancias auxiliares
f=sqrt(l(4)^2+l(5)^2+2*l(4)*l(5)*cos(qa+gamma));
g=sqrt(l(1)^2+l(2)^2+2*l(1)*l(2)*cos(psi-q2i));
```

```
26
27 % Ángulo qi
28 q1i= asin((l(4)/f)*sin(qa+gamma))+asin((l(2)/g)*sin(psi-q2i))  ...
      +acos((f^2+g^2-l(3)^2)/(2*f*g))-gamma;
29
30 % Ángulos auxiliares
31 alpha1=q1i+gamma;
32 alpha2=pi-psi+q2i;
33 alphaa=pi-qa-gamma;
34 alpha3=asin((l(1)*sin(alpha2)/g))+acos((g^2+l(3)^2-f^2)/(2*g*l(3)));
35 alpha4=acos((f^2+l(4)^2-l(5)^2)/(2*f*l(4)))+acos((l(3)^2+f^2-g^2)/(2*l(3)*f));
36
37 syms q1 q2 q3 q4 q5
38 theta=[q1,q2,q3,q4,q5,0,0]; % Los dos últimos valores indican la ...
      posición del sistema de la articulación O1 con respecto al sistema ...
      de referencia inercial
39
40 %% Posiciones relativas de cada sistema de referencia respecto al ...
      sistema de referencia inercial
41 Rref_O5x=l(1)*cos(q1)+l(2)*cos(q2)+l(3)*cos(q3)+l(4)*cos(q4);
42 Rref_O5y=l(1)*sin(q1)+l(2)*sin(q2)+l(3)*sin(q3)+l(4)*sin(q4);
43 Rref_O5=[Rref_O5x Rref_O5y 0];
44
45
46 %% Cálculo de variable B
47 B0 = [9.05 13.6 5.06 3.2]*10^-3; % Kg
48 B11= [0 0 0; 0 0 -(m(1)*l(1))/2; 0 (m(1)*l(1))/2 0]; % Kg*m
49 B12= [0 0 0; 0 0 -(m(2)*l(2))/2; 0 (m(2)*l(2))/2 0]; % Kg*m
50 B13= [0 0 0; 0 0 -(m(3)*l(3))/2; 0 (m(3)*l(3))/2 0]; % Kg*m
51 B14= [0 0 0; 0 0 -(m(4)*l(4))/2; 0 (m(4)*l(4))/2 0]; % Kg*m
52
53 B1=B11;
54 B1(:,:,2)=B12;
55 B1(:,:,3)=B13;
56 B1(:,:,4)=B14;
57
58 B21= [0 0 0; 0 (m(1)*(l(1)^2))/3 0; 0 0 (m(1)*(l(1)^2))/3]; % Kg*m^2
59 B22= [0 0 0; 0 (m(2)*(l(2)^2))/3 0; 0 0 (m(2)*(l(2)^2))/3]; % Kg*m^2
60 B23= [0 0 0; 0 (m(3)*(l(3)^2))/3 0; 0 0 (m(3)*(l(3)^2))/3]; % Kg*m^2
61 B24= [0 0 0; 0 (m(4)*(l(4)^2))/3 0; 0 0 (m(4)*(l(4)^2))/3]; % Kg*m^2
62
63 B2=B21;
64 B2(:,:,2)=B22;
65 B2(:,:,3)=B23;
66 B2(:,:,4)=B24;
67
68 B31= [625.77 106.81 0; 106.81 5892.47 0; 0 0 5647.41]*10^-9; % Kg*m^2
69 B32= [7282.20 -38.87 0; -38.87 1929.72 0; 0 0 8442.66]*10^-9; % Kg*m^2
70 B33= [625.77 106.81 0; 106.81 5892.47 0; 0 0 5647.41]*10^-9; % Kg*m^2
71 B34= [134.10 425.72 89; 425.72 5620.37 8.29; 89 8.29 5695.45]*10^-9; % ...
      Kg*m^2
72
73 B3=B31;
74 B3(:,:,2)=B32;
75 B3(:,:,3)=B33;
76 B3(:,:,4)=B34;
77
78
```

```
79
80 %% Cálculo de las matrices de rotación
81 % Rotaciones que se producen en el sentido contrario de las agujas
82 % del reloj ==> ángulos positivos.
83
84 R01=[cos(q1) -sin(q1) 0; sin(q1) cos(q1) 0; 0 0 1];
85 R12=[cos(-q2) -sin(-q2) 0; sin(-q2) cos(-q2) 0; 0 0 1];
86
87 R02=R01*R12;
88 R23=[cos(-q3) -sin(-q3) 0; sin(-q3) cos(-q3) 0; 0 0 1];
89
90 R03=R02*R23;
91 R13=R12*R23;
92 R34=[cos(-q4) -sin(-q4) 0; sin(-q4) cos(-q4) 0; 0 0 1];
93
94 R04=R03*R34;
95 R14=R13*R34;
96 R24=R23*R34;
97 R45=[cos(-q5) -sin(-q5) 0; sin(-q5) cos(-q5) 0; 0 0 1];
98
99 R05=R04*R45;
100 R15=R14*R45;
101 R25=R24*R45;
102 R35=R34*R45;
103
104 R10=inv(R01);
105 R20=inv(R02);
106 R30=inv(R03);
107 R40=inv(R04);
108 R50=inv(R05);
109 R21=inv(R12);
110 R32=inv(R23);
111 R31=inv(R13);
112 R43=inv(R34);
113 R41=inv(R14);
114 R42=inv(R24);
115 R54=inv(R45);
116 R51=inv(R15);
117 R52=inv(R25);
118 R53=inv(R35);
119
120 R0=sym([1 0 0; 0 1 0; 0 0 1]);
121 R0(:,:,2)=R01;
122 R0(:,:,3)=R02;
123 R0(:,:,4)=R03;
124 R0(:,:,5)=R04;
125 R0(:,:,6)=R05;
126
127
128 R1=R10;
129 R1(:,:,2)=sym([1 0 0; 0 1 0; 0 0 1]);
130 R1(:,:,3)=R12;
131 R1(:,:,4)=R13;
132 R1(:,:,5)=R14;
133 R1(:,:,6)=R15;
134
135 R2=R20;
136 R2(:,:,2)=R21;
```

```
137 R2(:,:,3)=sym([1 0 0; 0 1 0; 0 0 1]);
138 R2(:,:,4)=R23;
139 R2(:,:,5)=R24;
140 R2(:,:,6)=R25;
141
142 R3=R30;
143 R3(:,:,2)=R31;
144 R3(:,:,3)=R32;
145 R3(:,:,4)=sym([1 0 0; 0 1 0; 0 0 1]);
146 R3(:,:,5)=R34;
147 R3(:,:,6)=R35;
148
149 R4=R40;
150 R4(:,:,2)=R41;
151 R4(:,:,3)=R42;
152 R4(:,:,4)=R43;
153 R4(:,:,5)=sym([1 0 0; 0 1 0; 0 0 1]);
154 R4(:,:,6)=R45;
155
156 R5=R50;
157 R5(:,:,2)=R51;
158 R5(:,:,3)=R52;
159 R5(:,:,4)=R53;
160 R5(:,:,5)=R54;
161 R5(:,:,6)=sym([1 0 0; 0 1 0; 0 0 1]);
162
163 R=R0;
164 R(:,:,:,2)=R1;
165 R(:,:,:,3)=R2;
166 R(:,:,:,4)=R3;
167 R(:,:,:,5)=R4;
168 R(:,:,:,6)=R5;  % Notación: Rjk=R(:,:,k,j)
169
170
171 %% Cálculo de las distancias entre las articulaciones
172 % Las distancias no directas se han obtenido mediante el teorema del ...
       coseno.
173 OiOj=sym(zeros(5,5));
174 OiOj(1,1)=0;
175 OiOj(1,2)=l(1);
176 OiOj(1,3)=g;
177 OiOj(1,4)=f;
178 OiOj(1,5)=l(5);
179
180 OiOj(2,1)=OiOj(1,2);
181 OiOj(2,2)=0;
182 OiOj(2,3)=l(2);
183 OiOj(2,4)=sqrt(l(2)^2+l(3)^2-2*l(2)*l(3)*cos(alpha3));
184 OiOj(2,5)=sqrt(l(1)^2+l(5)^2-2*l(1)*l(5)*cos(alpha1));
185
186 OiOj(3,1)=OiOj(1,3);
187 OiOj(3,2)=OiOj(2,3);
188 OiOj(3,4)=l(3);
189 OiOj(3,5)=sqrt(l(3)^2+l(4)^2-2*l(3)*l(4)*cos(alpha4));
190
191 OiOj(4,1)=OiOj(1,4);
192 OiOj(4,2)=OiOj(2,4);
193 OiOj(4,3)=OiOj(3,4);
```

```
194 OiOj(4,5)=l(4);
195
196 OiOj(5,1)=OiOj(1,5);
197 OiOj(5,2)=OiOj(2,5);
198 OiOj(5,3)=OiOj(3,5);
199 OiOj(5,4)=OiOj(4,5);
200
201
202
203
204 %% Bucles para el cálculo de la matrix de inercias Icf
205 n=4;
206 Icf=sym(zeros(n+4,n+4));
207
208 for j=1:n-1
209    for k=j:n-1
210       A=xj(3,:)*fcn_sigma(j,k,n,B2,B3,R)*xj(3,:)';
211       B=-xj(3,:)*fcn_psi(j,k,n,OiOj,B1,R)*xj(3,:)';
212       C=-xj(3,:)*fcn_U(j,k,n,OiOj,B0,B1,R)*xj(3,:)';
213       Icf(j,k) = A+B+C;
214       Icf(k,j) = Icf(j,k);
215    end
216 end
217
218 for j=1:n
219    for k=n
220       if j~=n
221           A=xj(3,:)*fcn_sigma(j,k,n,B2,B3,R)*xj(3,:)';
222           B=-xj(3,:)*fcn_psi(j,k,n,OiOj,B1,R)*xj(3,:)';
223           Icf(j,k) = A+B;
224           Icf(k,j) = Icf(j,k);
225       else
226           Icf(j,k) = 1;
227       end
228    end
229 end
230
231 for j=1:n
232    for k=n+1:n+2
233       if j~=n
234           D=xj(3,:)*fcn_xi(j,n,B1,R)*xj(k-n,:)';
235           E=xj(3,:)*fcn_gamma(j,n,OiOj,B0,R)*xj(k-n,:)';
236           Icf(j,k) = D+E;
237           Icf(k,j) = Icf(j,k);
238       else
239           Icf(j,k) = D;
240           Icf(k,j) = Icf(j,k);
241       end
242    end
243 end
244
245 for j=n+1:n+2
246    for k=n+1:n+2
247       if j==k
248           Icf(j,k) = Mtot;
249       else
250           Icf(j,k) = 0;
251       end
```

```
252     end
253 end
254 
255 for j=n+3:n+4
256     for k=1:n+2
257         if (j~=n+3 && k~=n+1)|| (j~=n+4 && k~=n+2)
258             Icf(j,k) = fcn_jacob(j-n-2,k,xj,Rref_O5,theta);
259             Icf(k,j) = -Icf(j,k);
260         else
261             Icf(j,k) = 0;
262             Icf(k,j) = Icf(j,k);
263         end
264     end
265 end
266 
267 for j=n+3:n+4
268     for k=n+3:n+4
269         Icf(j,k) = 0;
270         Icf(k,j) = Icf(j,k);
271     end
272 end
```

E.2.1. Código *fcn_sigma*

```
1 function sigma = fcn_sigma(j,k,n,B2,B3,R)
2     sigma=zeros(3,3);
3     for i = max(j,k):n
4         sum=R(:,:,i+1,j+1)*(B2(:,:,i)+B3(:,:,i))*R(:,:,k+1,i+1);
5         sigma=sigma+sum;
6     end
7 end
```

E.2.2. Código *fcn_psi*

```
1 function psi_k = fcn_psi(j,k,n,OiOj,B1,R)
2     psi_k=zeros(3,3);
3     for i = max(j+1,k):n
4         sum=OiOj(i,j)*R(:,:,i+1,j+1)*B1(:,:,i)*R(:,:,k+1,i+1);
5         psi_k=psi_k+sum;
6     end
7 end
```

E.2.3. Código *fcn_U*

```
1 function U = fcn_U(j,k,n,OiOj,B0,B1,R)
2     xi=zeros(3,3);
3     gamma_t=zeros(3,3);
4     U=zeros(3,3);
5     for t = k:n-1
6 
7         for i = max(j,t+1):n
8             sum_xi=R(:,:,i+1,j+1)*OiOj(i,j)*B1(:,:,i)*R(:,:,t+1,i+1);
9             xi=xi+sum_xi;
10        end
11 
```

```
12          for i = max(j+1,t+1):n
13              sum_gamma=OiOj(i,j)*B0(i)*R(:,:,t+1,j+1);
14              gamma_t=gamma_t+sum_gamma;
15          end
16
17          sum=(xi+gamma_t)*OiOj(t,t+1)*R(:,:,k+1,t+1);
18          U=U+sum;
19      end
20  end
```

E.2.4. Código *fcn_xi*

```
1  function xi = fcn_xi(j,n,B1,R)
2      xi=zeros(3,3);
3      for i = j:n
4          sum=R(:,:,i+1,j+1)*B1(:,:,i)*R(:,:,1,i+1);
5          xi=xi+sum;
6      end
7  end
```

E.2.5. Código *fcn_gamma*

```
1  function gamma = fcn_gamma(j,n,OiOj,B0,R)
2      gamma=zeros(3,3);
3      for i = j+1:n
4          sum=OiOj(i,j)*B0(i)*R(:,:,1,j+1);
5          gamma=gamma+sum;
6      end
7  end
```

E.2.6. Código *fcn_jacob*

```
1  function jacob = fcn_jacob(j,k,xj,Rref_O5,theta)
2      jacob=0;
3      jacob=diff(Rref_O5*xj(j,:)',theta(k));
4  end
```