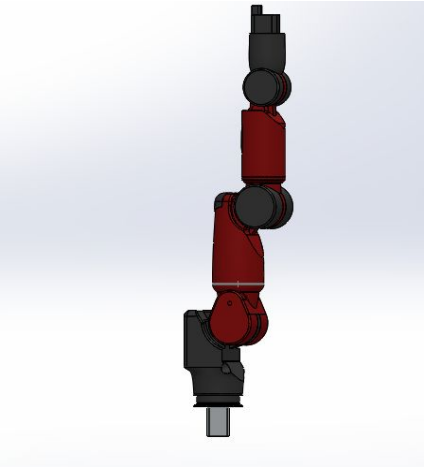
ITESO

Universidad Jesuita de Guadalajara

Materia: Sistemas de Control Automático

Profesor: Luis E. González

Título: Manipulador serial de siete grados de libertad.



Nombre del Alumno	Expediente	Correo Electrónico
Jorge Loya Herrera	ie703043	ie703043@iteso.mx
Alan Yahir Duran Merchant	ie704239	ie704239@iteso.mx

Fecha: 5 de diciembre del 2018

Índice:

Planteamiento del problema.

Nuestro objetivo es controlar la posición del efector final de un manipulador serial de siete grados de libertad mediante la manipulación de la posición angular de cada una de sus articulaciones, que a su vez están definidas por las velocidades angulares ejercidas por los motores. De manera que necesitamos implementar un sistema que a partir de una posición deseada calcule los siete voltajes necesarios para hacer que los motores se muevan a una velocidad tal que se cumpla el objetivo de control.

Variables controladas:
Posiciones (articulaciones de robot Baxter).

Variables manipuladas:
Voltaje (Motor).
Velocidad angular (Motor).

El modelo realizado está basado en las especificaciones del robot industrial Baxter, construido por Rethink Robots, y por motivos prácticos el desarrollo de este proyecto estará limitado únicamente a la obtención del modelo matemático, diseño del controlador y simulación de su comportamiento mediante un modelo realizado en SolidWorks y la herramienta SimScape de MATLAB que permite importar este modelo a Simulink y visualizarlo.

Link	θ	d (m)	a (m)	α (rad)	m (kg)
1	θ_1	0.2703	0.069	$-\pi/2$	5.70044
2	θ_2	0	0	$\pi/2$	3.22698
3	θ_3	0.3644	0.069	$-\pi/2$	4.31272
4	θ_4	0	0	$\pi/2$	2.07206
5	θ_5	0.3743	0.01	$-\pi/2$	2.24665
6	θ_6	0	0	$\pi/2$	1.60979
7	θ_7	0.2295	0	0	0.54218

Figura 1. Tabla de parámetros para brazo Baxter

Link	I_{xx}	I_{yy}	I_{zz}
1	0.0470910226	0.035959884	0.0376697645
2	0.027885975	0.020787492	0.0117520941
3	0.0266173355	0.012480083	0.0284435520
4	0.0131822787	0.009268520	0.0071158268
5	0.0166774282	0.003746311	0.0167545726
6	0.0070053791	0.005527552	0.0038760715
7	0.0008162135	0.0008735012	0.0005494148
Link	I_{xy}	I_{yz}	I_{xz}
1	-0.0061487003	-0.0007808689	0.0001278755
2	-0.0001882199	0.0020767576	-0.00030096397
3	-0.0039218988	-0.001083893	0.0002927063
4	-0.0001966341	0.000745949	0.0003603617
5	-0.0001865762	0.0006473235	0.0001840370
6	0.0001534806	-0.0002111503	-0.0004438478
7	0.000128440	0.0001057726	0.00018969891

Figura 2. Inercia para cada brazo

Link	$\bar{x}$	$\bar{y}$	$\bar{z}$
1	-0.05117	0.07908	0.00086
2	0.00269	-0.00529	0.06845
3	-0.07176	0.08149	0.00132
4	0.00159	-0.01117	0.02618
5	-0.01168	0.13111	0.0046
6	0.00697	0.006	0.06048
7	0.005137	0.0009572	-0.06682

Figura 3. Centros de masas para robot Baxter.

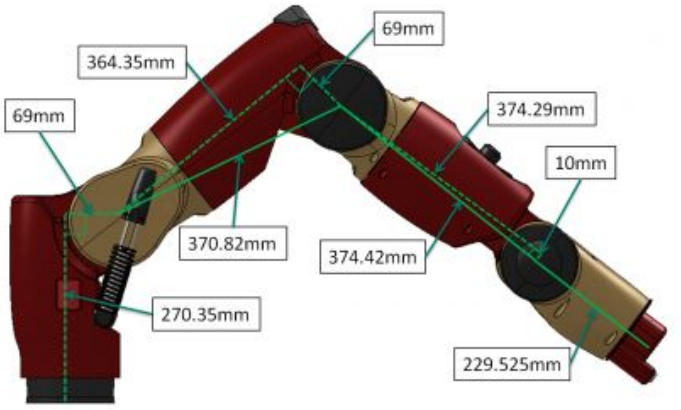


Figura 4. Brazo Baxter con sus respectivas medidas.

Todos estos parámetros son asignados con ayuda de la herramienta diseñada por Peter Corke para MATLAB.

Marco teórico

Para la representación matemática de la geometría que involucra a este tipo de robots, es importante conocer algunos conceptos primero, a continuación se muestran los más relevantes:

- Manipulador serial: están compuestos por una serie de barras rígidas unidas por articulaciones de un grado de libertad que presentan un movimiento rotacional o prismático. Estas barras y articulaciones en conjunto forman una cadena cinemática abierta.

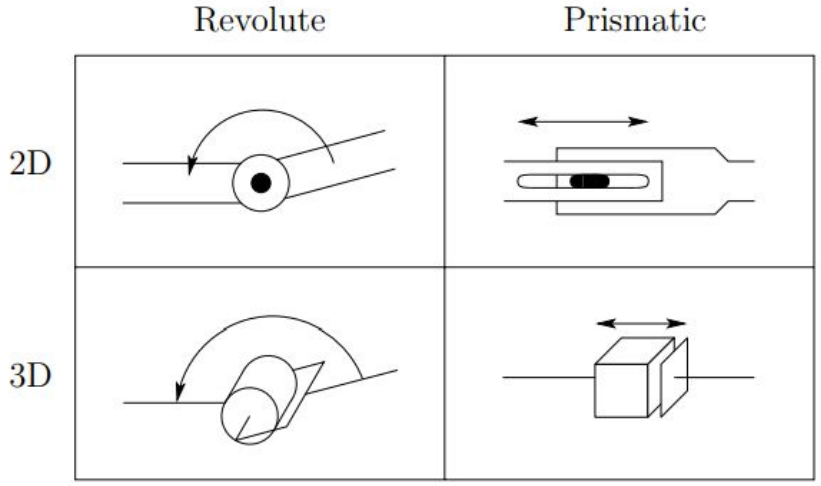


Figura 5. Representación de los tipos de articulaciones.

- Matriz de rotación: es una forma de representar la orientación relativa de un cuerpo rígido respecto a otro de referencia. Como el robot que estamos utilizando se mueve en tres dimensiones, nuestra matriz queda de la siguiente forma:

$$R_1^0 = \begin{bmatrix} x_1 \cdot x_0 & y_1 \cdot x_0 & z_1 \cdot x_0 \\ x_1 \cdot y_0 & y_1 \cdot y_0 & z_1 \cdot y_0 \\ x_1 \cdot z_0 & y_1 \cdot z_0 & z_1 \cdot z_0 \end{bmatrix}$$

Figura 6. Matriz de rotación en tres dimensiones

Dado un plano fijo *o0x0y0z0*, un plano actual *o1x1y1z1* y una matriz de rotación $R\,{}^{0}_{1}$ que las relaciona y si un tercer plano *o2x2y2z2* se puede obtener relativo al plano actual mediante una rotación $R\,{}^{1}_{2}$, entonces la rotación respecto al plano fijo se puede obtener de la siguiente manera:

$$R\,{}^{0}_{2} = R\,{}^{0}_{1}\ R\,{}^{1}_{2}$$

- Matriz de transformación homogénea: una matriz de dimensión 4x4 que representa la transformación de un vector de coordenadas homogéneas de un sistema de coordenadas a otro.

$$H = \left[\begin{array}{c|c} R_{3\times3} & d_{3\times1} \\ \hline f_{1\times3} & s_{1\times1} \end{array}\right] = \left[\begin{array}{c|c} Rotation & Translation \\ \hline perspective & scale\ factor \end{array}\right]$$

Figura 7. Forma general de una matriz de transformación homogénea.

Esta matriz está compuesta por cuatro submatrices de distinto tamaño: una submatriz R de 3x3 que corresponde a una matriz de rotación; una submatriz p de 3x1 que corresponde al vector de traslación; una submatriz f de 1x3 que representa una transformación de perspectiva, y una submatriz s de 1x1 que representa un escalado global. En robótica generalmente sólo interesará conocer el valor de R3x3 y de p3x1 , considerándose las componentes f1x3 nulas y la de w1x1 la unidad,

$$T = \begin{bmatrix} R_{3\times3} & p_{3\times1} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} Rotación & Traslación \\ 0 & 1 \end{bmatrix}$$

Figura 8. Matriz de transformación homogénea utilizada para nuestro proyecto

De la misma manera que con las matrices de rotación, dada una transformación homogénea $H\,{}^{0}_{1}$ relacionando a dos planos y si un segundo movimiento rígido es aplicado al plano actual, entonces:

$$H\,{}^{0}_{2} = H\,{}^{0}_{1} H$$

Mientras que, si el segundo movimiento rígido es realizado relativo al plano fijo, entonces:

$$H\,{}^{0}_{2} = H H\,{}^{0}_{1}$$

- Cinemática directa: se refiere al cálculo necesario para determinar la posición y orientación del efector final dadas las posiciones angulares de las articulaciones del robot.
- Cinemática diferencial: De manera similar, esta cinemática busca entre otras cosas encontrar la relación existente entre las velocidades articulares y la velocidad del efector final, y por extensión de cualquier otro punto asociado al mismo.

Descripción del Modelo Matemático

Modelado del robot

Para la representación de nuestro robot en el espacio nos basamos en la convención de Denavit-Hartenberg, el cual además de simplificar nuestro modelo a uno donde solamente son necesarios cuatro parámetros donde uno de ellos es variable, nos permite realizar un procedimiento sistemático con el cual poder resolver el problema de la cinemática directa.

Dichos parámetros son:

- ai: Longitud de la barra rígida
- di: Desplazamiento lineal de la barra rígida
- αi: Ángulo de separación del eje Z(i-1) al eje Zi, respecto al eje Xi
- θi: Posición angular de la articulación, como en nuestro robot solo tenemos articulaciones con movimiento revoluto, estas son nuestras siete variables y las demás son parámetros fijos.

En esta convención, la matriz de transformación homogénea para cada plano queda de la siguiente forma:

$$ {}^{i-1}\mathbf{A}_i = \left(\begin{array}{cccc|c} \cos\theta_i & -\cos\alpha_i \sin\theta_i & \sin\alpha_i \sin\theta_i & a_i \cos\theta_i \\ \sin\theta_i & \cos\alpha_i \cos\theta_i & -\sin\alpha_i \cos\theta_i & a_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ \hline 0 & 0 & 0 & 1 \end{array} \right) $$

Figura 9. Matriz de transformación homogenea

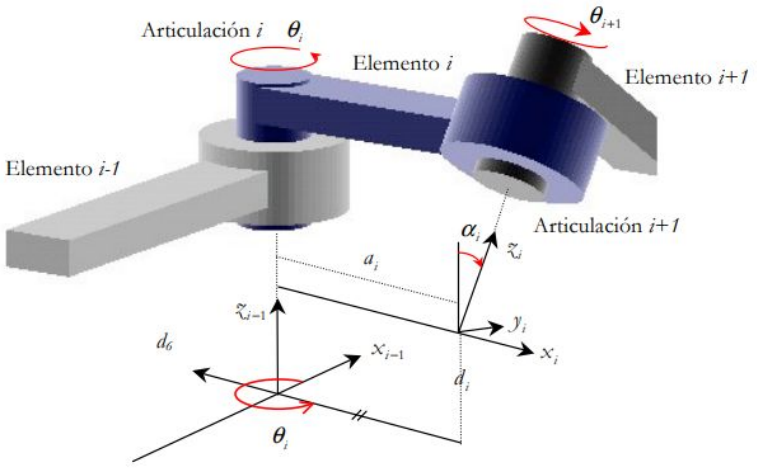


Figura 10. Asignación de parámetros cinemáticos según la metodología de Denavit-Hartenberg

Usando las propiedades de la matriz de transformación homogénea, se puede resolver el problema de la cinemática directa mediante un método iterativo donde la matriz de transformación homogénea del plano de referencia se puede obtener mediante la multiplicación consecutiva de las matrices:

$$ {}^{0}T_i = {}^{0}T_1 \, {}^{1}T_2 \ldots {}^{i-1}T_i. $$

La posición del efector final se puede obtener de la submatriz *p* de T_0^7

Para resolver la cinemática diferencial y el cálculo de las velocidades obtenemos el jacobiano, el cual se divide en velocidad lineal y velocidad angular, teniendo una dimensión de 6xn, donde n es el número de grados de libertad del robot:

$$J = \begin{bmatrix} J_v \\ J_\omega \end{bmatrix}$$

Para esta etapa del proyecto no fue necesario utilizar Jw, por lo que decidimos ignorar esta submatriz para quedarnos solamente con Jv:

$J_v = [J_{v1} \ \ldots J_{v7}]$

La cual está conformada por una matriz de 3xn

Para calcular cada Jv:

$$J_{v_i} = z_{i-1} \times (o_n - o_{i-1})$$

Donde z_i es una matriz de 3x1 y O_n es una matriz de 3x1,

$J_{v1} = Z_0 x (O_7 - O_0)$
$J_{v2} = Z_1 x (O_7 - O_1)$
$J_{v3} = Z_2 x (O_7 - O_2)$
$J_{v4} = Z_3 x (O_7 - O_3)$
$J_{v5} = Z_4 x (O_7 - O_4)$
$J_{v6} = Z_5 x (O_7 - O_5)$
$J_{v7} = Z_6 x (O_7 - O_6)$

Siendo Z la última columna de la matriz de rotación y Z0 = [0;0;1]

<u>Modelado de los motores</u>

Los motores están basados en una caracterización que habíamos realizado previamente para las prácticas del curso, por lo que simplemente basamos nuestro espacio de estados de acuerdo a esos valores.

$$\dot{x}(t)=\begin{bmatrix} -\frac{R_a}{L_a} & -\frac{K_b}{L_a} \\ \frac{K_T}{J_m} & -\frac{B_m}{J_m} \end{bmatrix} x(t)+\begin{bmatrix} \frac{1}{L_a} \\ 0 \end{bmatrix} u(t)$$

$$y(t)=\begin{bmatrix} 0 & 1 \end{bmatrix} x(t)$$

$$x(t)=\begin{bmatrix} i_a(t) \\ \omega(t) \end{bmatrix}, \quad u(t)=V_a(t)$$

Figura 11. Espacio de estados del motor

Pero este modelo no consideraba la carga que debía soportar cada motor, por lo que intentamos agregar esta parte al modelo como si fuera una perturbación.

Para calcular el par de carga decidimos utilizar el algoritmo iterativo de Newton-Euler, el cual consiste en empezar desde la primera barra (conectada al cuerpo) hasta la última barra, en cada barra se calculan las velocidades y aceleraciones de diversos puntos de ella misma. Posteriormente se hace una segunda iteración comenzando desde nuestra última barra hasta la primera, donde se calcula la fuerza y el torque a partir de los parámetros obtenidos.

$a_{c,i}$ = the acceleration of the center of mass of link i

$a_{e,i}$ = the acceleration of the end of link i (i.e., joint $i+1$)

ω_i = the angular velocity of frame i w.r.t. frame 0

α_i = the angular acceleration of frame i w.r.t. frame 0

g_i = the acceleration due to gravity (expressed in frame i)

f_i = the force exerted by link $i-1$ on link i

τ_i = the torque exerted by link $i-1$ on link i

R_i^{i+1} = the rotation matrix from frame $i+1$ to frame i

m_i = the mass of link i

I_i = the inertia matrix of link i about a frame parallel to frame i whose origin is at the center of mass of link i

$r_{i,ci}$ = the vector from joint i to the center of mass of link i

$r_{i+1,ci}$ = the vector from joint $i+1$ to the center of mass of link i

$r_{i,i+1}$ = the vector from joint i to joint $i+1$

Figura 12. Parámetros a calcular

Primera iteración, desde la primera hasta la última barra, se calcula mediante las siguientes ecuaciones en el siguiente orden:

$$\omega_i = (R_{i-1}^i)^T \omega_{i-1} + b_i \dot{q}_i \quad b_i = (R_0^i)^T z_{i-1}$$

$$\alpha_i = (R_{i-1}^i)^T \alpha_{i-1} + b_i \ddot{q}_i + \omega_i \times b_i \dot{q}_i$$

$$a_{e,i} = (R_{i-1}^i)^T a_{e,i-1} + \dot{\omega}_i \times r_{i,i+1} + \omega_i \times (\omega_i \times r_{i,i+1})$$

$$a_{c,i} = (R_{i-1}^i)^T a_{e,i-1} + \dot{\omega}_i \times r_{i,ci} + \omega_i \times (\omega_i \times r_{i,ci})$$

Utilizando las siguientes condiciones iniciales:

$$\omega_0 = 0, \alpha_0 = 0, a_{c,0} = 0, a_{e,0} = 0$$

$$f_{n+1} = 0, \qquad \tau_{n+1} = 0$$

Para la segunda iteración, se calculan los parámetros faltantes para así obtener el torque resultante

$$f_i = R_i^{i+1} f_{i+1} + m_i a_{c,i} - m_i g_i$$

$$\tau_i = R_i^{i+1} \tau_{i+1} - f_i \times r_{i,ci} + (R_i^{i+1} f_{i+1}) \times r_{i+1,ci} + \alpha_i + \omega_i \times (I_i \omega_i)$$

Descripción del Proceso de Diseño del Controlador:

Controlador por retro de error:
Una vez calculado nuestro jacobiano, podemos comenzar a diseñar nuestro controlador.

Sabemos que nuestro sistema recibirá de entrada una referencia en ‘x’, ‘y’ y ‘z’
$y_{ref} = [x; y; z]$

Sabemos que la velocidad es equivalente al jacobiano de la derivada de la posición angular.
$p' = J_v(q')$

Nosotros queremos controlar las velocidades de los motores de tal manera que nuestro sistema converja en la posición deseada por lo que nuestros estados serán:

$x_1 = q$
$x_2 = p$

$u = q'$

Entonces:
$x_1' = q' = u$
$x_2' = J_v(x_1) * u$
$y = x_2$

Una vez tenemos los estados, debemos considerar que nuestro sistema seguirá referencias no constantes por lo tanto el controlador a utilizar será referencia por retro de error.
$e' = y_{ref}' - y'$
$e' = y_{ref}' - J_v(x_1) * u$
$u = J_v^{-1}(y_{ref}' - k_e)$

Diseño de controlador PID.

A partir de la salida del primer controlador, obtenemos un voltaje de referencia que nuestros motores deben de seguir. Como resultaría impráctico utilizar sensores de corriente para los motores y no implementamos un observador, PID fue nuestra mejor opción ya que este controlador no necesita conocer los estados del sistema.

Cuando intentamos hacer que compensara los pares de carga, la salida del sistema requerida tenía que ser demasiado grande, de manera que no nos fue posible simular ni sintonizar adecuadamente las ganancias requeridas.

Una vez retiramos del sistema la carga de los motores, simulamos un sistema ideal donde prácticamente no se necesitaba ningún valor significativo de voltaje o corriente, por lo que el sistema se podía sintonizar sin mucho problema utilizando solo ganancias proporcionales.

Implementación del Controlador Diseñado:

Para implementar el controlador por retro de error:

```
function dX = simulink_jacobiano(x, yref,dYref,t)

Ke = -5;
```

Figura 13. Ganancia para controlador por retro de error

Primeramente, declaramos una ganancia Ke, esta se obtuvo variando hasta obtener el mejor resultado para nuestra salida.

```
%Parametros del sistema
d = [0.2703 0 0.3644 0 0.3743 0 0.2295]; %metros
a = [0.069 0 0.069 0 0.01 0 0]; %metros
alpha = [-pi/2 pi/2 -pi/2 pi/2 -pi/2 pi/2 0]; %radianes
```

Figura 14. Parámetros del sistema

Se declaran los parámetros de nuestro sistema.

```
T = D_H(x(1:7),a,alpha,d,0,7);
J = jacobiano(T);
Jv = J(4:6,:);
```

Figura 15. Cálculo del Jacobiano

Aplicamos la Transformación homogénea, posteriormente el jacobiano y de ahí obtenemos el Jv.

```
e = yref - T{1,7}(1:3,4);
U = pinv(Jv)*(dYref - Ke*e);
%ODE's
dX = [U;Jv*U];
```

Figura 16**.** Implementación de controlador por retro de error.

Implementamos el controlador por retro de error.

Ahora ingresamos las referencia de 2 senoidales en x y z, y una constante en y.

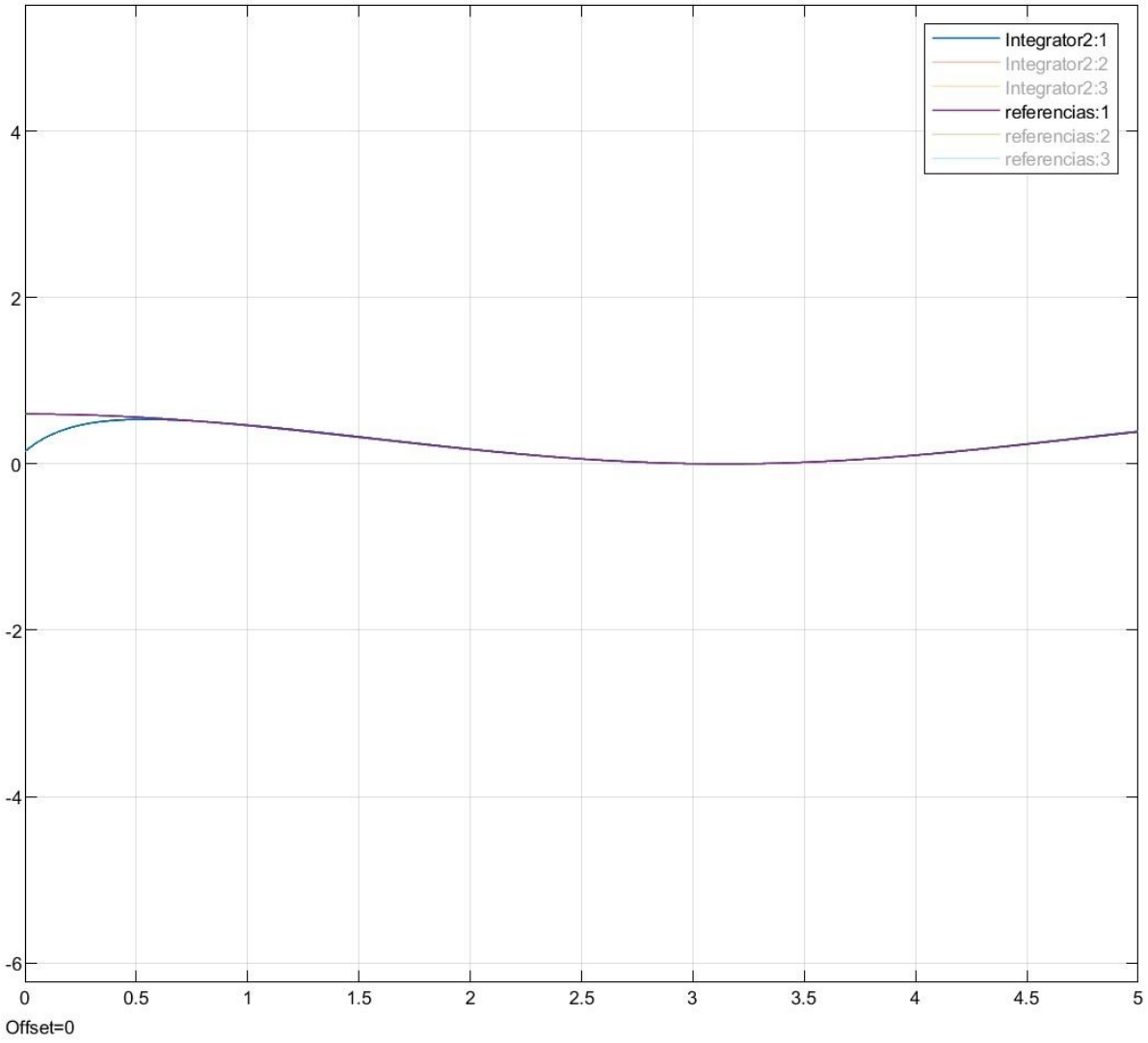


Figura 17. Posición x del efector final respecto a la referencia x

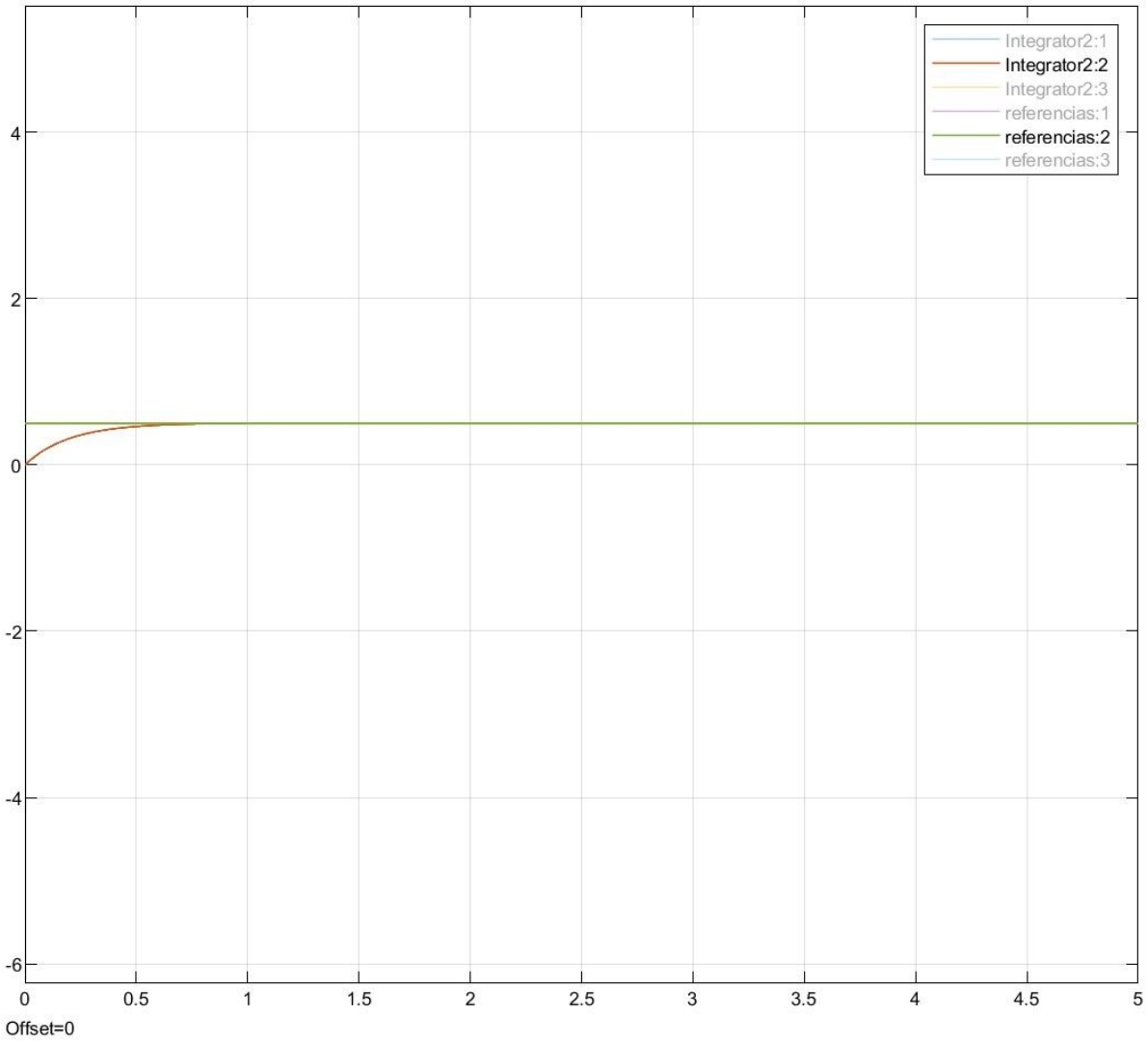


Figura 18. Posición y del efector final respecto a la referencia y

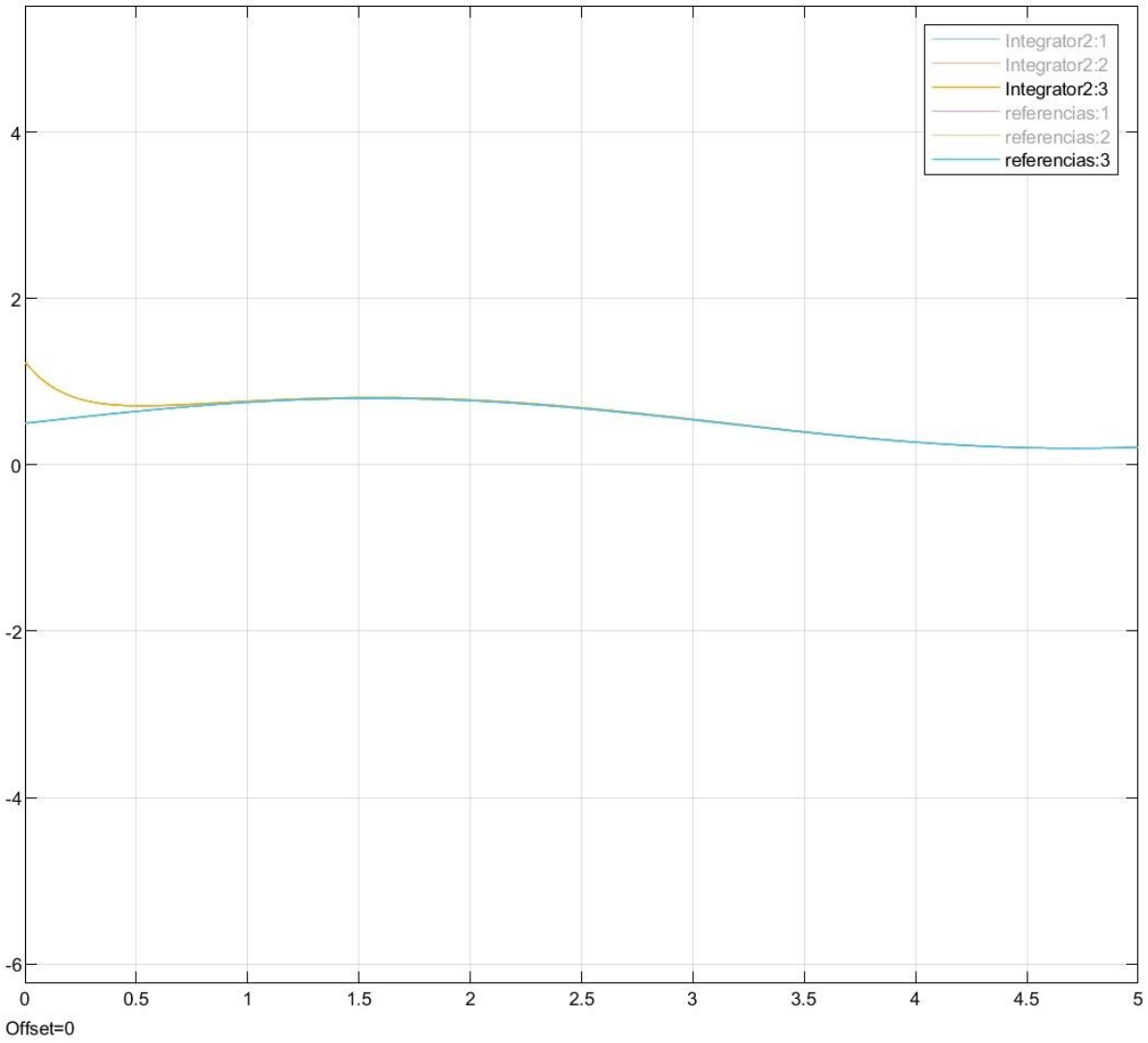


Figura 19. Posición z del efector final respecto a la referencia z

Como podemos ver se logra seguir la referencia.

Para implementar el control PID nos basamos en los datos calculados en la práctica 3:

$R_a = 11.24$

$L_a = 2.712e-3$

$K_b = 14.87e-3$

$K_t = K_b$

$B_m = 2.01e-6$

$J_m = 2779.56e-9$

Originalmente se consideró el PID y el modelo en un solo bloque, sin embargo cuando se calculó el par de carga que soportará cada motor, fue necesario separar en 2 bloques, uno el PID y el otro con los estados del motor:

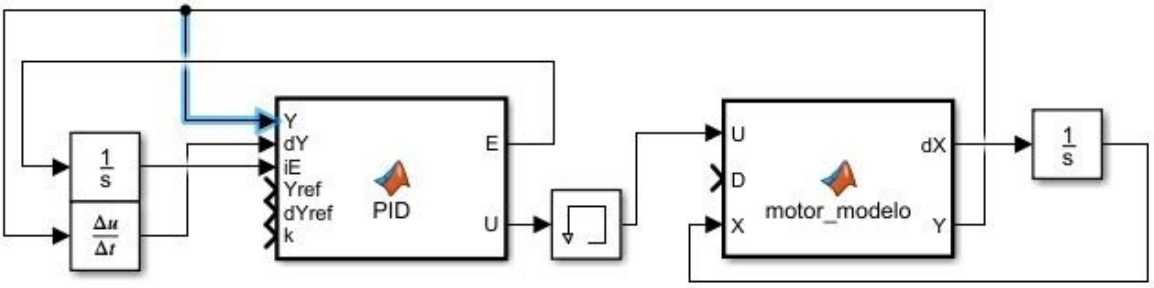


Figura 20. Bloques que se encargan de simular el PID y el modelado del motor

```
function [dX,Y]= motor_modelo(U,D,X)

%MATRICES DEL MOTOR
Ra = 11.24;
La = 2.713e-3;
Kb = 14.87e-3;
Kt = Kb;
bm = 2.01e-6;
Jm = 2779.56e-9;

A = [-Ra/La -Kb/La;Kt/Jm -bm/Jm];
B = [1/La;0];
C = [0 1];

%CONTROLADOR PID
dX = [A*X + B*U];
Y = [C*X + D*U];
```

Figura 21. Implementación de matlab para el modelo del motor.

```
function [E,U] = PID(Y,dY,iE,Yref,dYref,k)

E = Yref - Y;
dE = dYref - dY;

U = k(1)*E + k(2)*iE + k(3)*dE;
```

Figura 22. Implementación del PID para nuestro motor.

GUI.

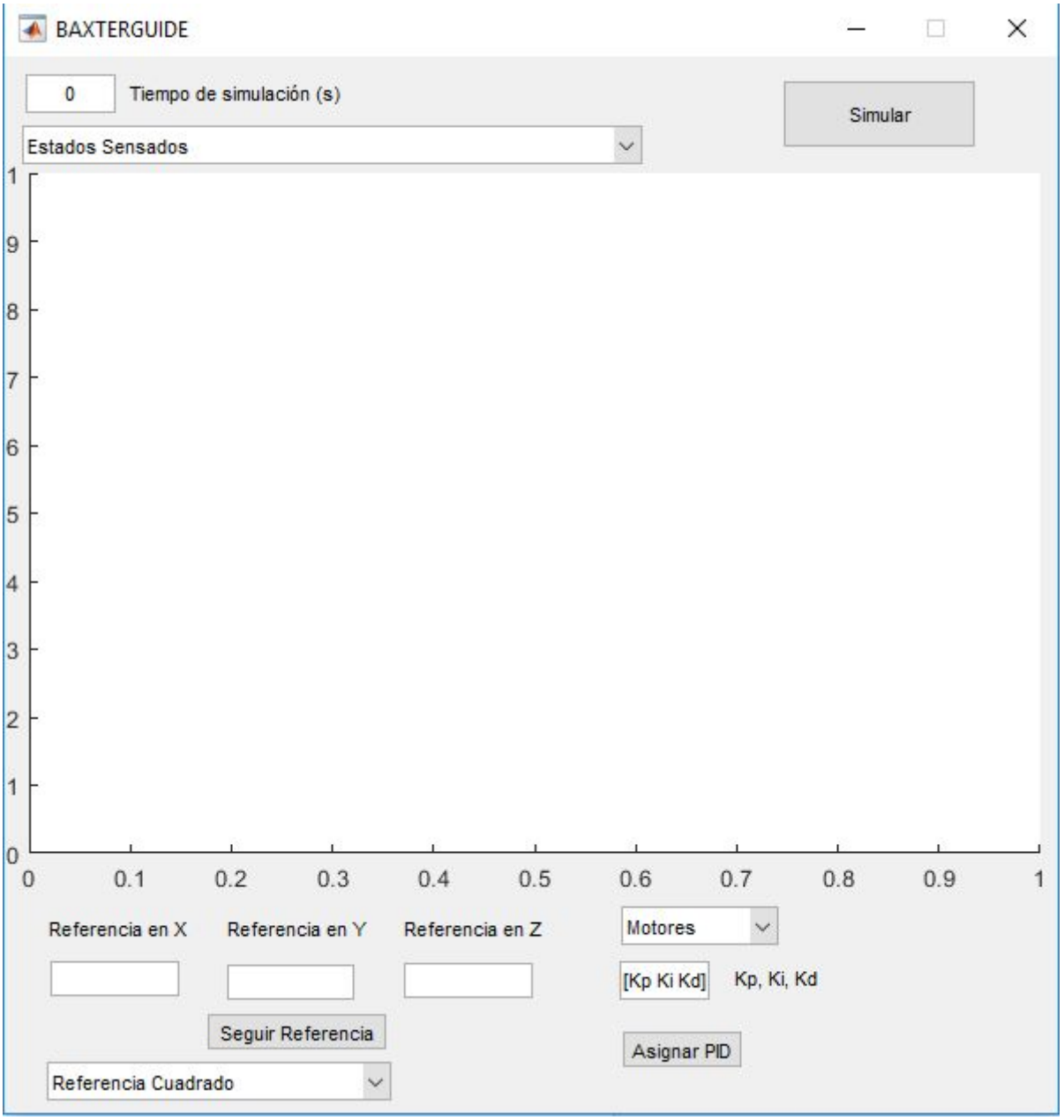


Figura 23.GUI implementada en el proyecto.

El objetivo de la GUI fue tener una interfaz para poder modificar de una manera mas comoda las variables de nuestros controladores (PID), las referencias que deseamos seguir (x, y, z), dibujar ya sea un cuadrado o un círculo, visualizar que sucede en los estados, las posición del efector final respecto a la referencia ingresada, voltajes, corriente, señales de control y el tiempo de simulación deseado.

Análisis de Resultados y Conclusiones:

Simulando en un caso ideal donde no existe un par de carga sobre los motores, nuestro modelo sigue de manera correcta las referencias ingresadas:

Simscape:

Uno de los mayores problemas que se presentaron fue al momento de exportar el modelo de SolidWorks a Simulinks, ya que se trató de definir límites en SolidWorks, sin embargo después de investigar qué es lo que realizaba Simscape al momento de exportar el modelo a Simulinks, se descubrió que los límites diseñados no se exportaban.

Otro gran problema fue establecer los ángulos iniciales de cada brazo del diseño, en nuestro caso este problema se resolvió implementando offset para que los ángulos cuadraran con el origen deseado. Posteriormente se descubrió que la manera de controlar esto era dejar nuestro brazo robot en la posición deseada antes de exportar, ya que de esta posición obtendrá sus ángulos iniciales.

Par de carga:

Se logró obtener un modelo que siguiera determinadas referencias (Cuadrados, círculos verticales, etc.) o una referencia en específico, sin embargo no se logró controlar el PID cuando a este se le implemento el par de carga, esto debido al tiempo, ya que solo se lograron estabilizar 2 de los motores, por lo cual se descarto el par de carga del producto final.

Es importante aclarar que en el método utilizado obtenemos un torque con una matriz de 1x3, lo cual se resolvió sacando la magnitud de este, sin embargo esto ocasionó que solo tuviéramos torques positivos.

Una manera alterna para solucionar este problema es utilizar los bloques diseñados por Peter Corke, en estos bloques ya existe una manera de calcular directamente el par de carga, sin embargo este método consume muchos recursos del sistema, lo cual volvió muy lento simular los resultados, ya que para simular estos resultados debemos variar manualmente las ganancias de Kp, Ki y Kd sin embargo es una opción muy viable.

Simulink

Una gran recomendación sería utilizar otro software, ya que implementar todo lo que se realizó fue bastante costoso, debido a que simulink tiene demasiadas limitaciones, algunas de las recomendaciones sería utilizar un software como ROSS.

Aprendizajes y dudas.

Realizar este proyecto requirió una gran variedad de conocimientos, donde muchos de ellos no se vieron dentro de la clase, entre los aprendizajes se encuentran:

Transformación Homogénea:
Para poder representar un espacio de n dimensiones (en nuestro caso de 3) en un sistema de coordenadas homogéneas.

Matriz de rotación:
Para poder representar la posición relativa de un objeto (barras) respecto a otro.

Cinemática Diferencial:
Se utilizó para encontrar la relación existente entre las velocidades articulares y la velocidad del efector final.

Cinemática Directa:
Se utilizó para encontrar la posición y orientación del efector final.
Cálculo del Jacobiano:

Recursividad de Newton Euler:
Se utilizó para el cálculo del par de carga que soportará cada uno de los motores.

Uso de Simscape:
Se utilizó para poder conectar SolidWorks con Simulinks.

Toolbox Peter Corke:
Se utilizó para facilitar la implementación del Jacobiano, transformación homogénea, cinemática directa y diferencial.

Implementación de un PID, con el objetivo de compensar el par de carga:
Para poder representar de manera más real el movimiento del brazo robot.

Una de nuestras mayores dudas fue que otro sistema podríamos utilizar en lugar de simulink, ya que este nos ocasionó muchas limitaciones y gran parte del trabajo no era como se resolvía algo, sino el cómo implementarlo en Simulinks.

Referencias.

CIDECAME. (5 de 12 de 2018). Obtenido de http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro39/321_transformacin_de_matrices_homogneas.html

Corke, P. (2017). *Robotics Toolbox.* MATLAB.

Spong, M. W. (1989). *Robot Modeling and Control.* New York: JOHN WILEY & SONS, INC.