UNIVERSITAT POLITÈCNICA DE VALÈNCIA

departamento de ingeniería de sistemas y automática

DISCA

CONTROL DE UN BRAZO ROBOT CON ARTICULACIONES ELÁSTICAS

MÁSTER EN AUTOMÁTICA E INFROMÁTICA INDUSTRIAL

Realizado por:
Aleks Emilov Goranov
Dirigido por:
Ranko Zotovic Stanisic

Fecha: 09/2017

Resumen

El objetivo de este proyecto es implementar distintas técnicas de control, para controlar un brazo robótico con articulaciones elásticas. Este proyecto se centra sobre todo en el control optimo con el que se procura la operación del sistema dinámico al menor costo. Se han probado tres modalidades del regulador lineal cuadrático: el clásico, con seguimiento de trayectoria y el iterativo. Previamente se ha obtenido el modelo dinámico a través de las ecuaciones de Lagrange-Euler que requieren de las energías potenciales y cinéticas de los diferentes eslabones. Después se ha procedido a linealizar el modelo para pasar al espacio de estados que es la forma recomendable para manejar y controlar procesos multivariable. Para probar el control por una parte se han realizado pruebas y simulación con Matlab y por otra se ha probado en la planta real. Para el caso de la planta real se ha tenido que usar un microcontrolador que representa el dispositivo de mando. Esto implica la interacción con sensores y actuadores y la necesidad de desarrollar un software que realice todas las tareas de interpretación de información para determinar y obtener el algoritmo de control que lleve el sistema al estado deseado. Asimismo, se ha implementado el homing y medidas de seguridad como es la parada controlada.

Palabras clave: control óptimo, brazo robótico con elasticidad, LQR, LQR con trayectoria, iLQR, STM32F4, Raspberry Pi, programación de microcontroladores.

Abstract

This project is based on the implementation of different techniques of control, to control a robotic arm with elastic joints. The main objective of this project is the design of optimal control with which, we pretend to operate the dynamic system at the lowest cost. Three modalities of the quadratic linear regulator have been tested: the classic, with trajectory tracking and the iterative. Previously the dynamic model has been obtained through the Lagrange-Euler equations that require the potential and kinetic energies of the different links. Then, the model has been linearized to move to the state space, which is the recommended way to manage and control multivariate processes. For the tests of the control, on the one hand, it has been tested and simulated with Matlab and on the other hand, it has been tested in the real plant. For the case of the real plant, a microcontroller has been used and it represents the control device. That involves interaction with sensors and actuators and the need to develop a software that performs all the tasks of interpreting information to determine and obtain the control algorithm that brings the system to the desired state. Also, homing has been implemented and security measures like controlled stop.

Key words: optimal control, robot arm with elasticity, LQR, LQR trajectory, iLQR, STM32F4, Raspberry Pi, microcontroller programming.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE CÓDIGOS

CAPÍTULO 1: INTRODUCCIÓN

En los últimos años, muchos investigadores han tratado el problema de control de robots con articulaciones flexibles/elásticas. Sin embargo, en comparación con el gran volumen de literatura disponible sobre el control de robots rígidos, se ha publicado relativamente poco sobre el control de robots con articulaciones elásticas. Los efectos de elasticidad pueden limitar la robustez y el rendimiento del controlador del robot dado, e incluso pueden conducir a la inestabilidad si se descuidan a la hora de diseñar el controlador. La flexibilidad mecánica en los robots manipuladores está presente por dos razones principales: el uso de elementos de transmisión/reducción como podrían ser correas, harmonic drives o engranajes cicloidales; y el uso de material ligero y delgado para reducir la masa de los elementos móviles.

Figura 1. Componentes de un harmonic drive.

La flexibilidad introduce deformaciones estáticas y dinámicas entre la posición de los actuadores de accionamiento y la posición del efector final del manipulador. De modo que, si no se tienen en cuenta, se observa un comportamiento oscilatorio, típicamente de pequeña magnitud, pero a una frecuencia relativamente alta, puede producirse alguna forma de inestabilidad (por ejemplo, parloteo) en tareas que implican contacto con el medio ambiente. Si asumimos que la flexibilidad es modelada como un muelle torsional lineal, se obtiene el modelo dinámico del manipulador con articulaciones flexibles con la forma que se muestra en la figura 2.

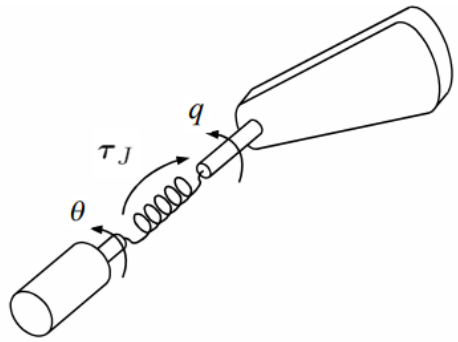


Figura 2. Articulación elástica.

1.1. Objetivos

Los objetivos del presente proyecto radican en la implementación de varias técnicas de control para el control de un brazo robótico de dos grados de libertad con articulaciones elásticas. Para ello, se han tenido que cumplir los siguientes objetivos:

- Obtener el modelo del brazo robot con los efectos de elasticidad en las articulaciones siguiendo un enfoque Lagrangiano, a partir de la energía cinética y potencial de cada eslabón.
- Analizar distintas técnicas de diseño de sistemas de control para sistemas no lineales.
- Comparar los distintos tipos de control analizando la respuesta obtenida de las simulaciones.
- Programar distintas plataformas y en cada una implementar un tipo de control.
- Establecer una comunicación entre los dos dispositivos empleados para transmitir los datos.
- Verificar el funcionamiento en la planta real y comparar con los resultados obtenidos durante las simulaciones.
- Optimizar el consumo de energía al tratarse de un dispositivo móvil alimentado por baterías.

1.2. Justificación

La teoría y la práctica del control tienen un amplio rango de aplicaciones en los campos de la ingeniería aeronáutica, química, mecánica, eléctrica, así como en muchas otras disciplinas. Las ventajas del control eficiente en la industria son inmensas, e incluyen mejoras en la calidad de los productos, reducción en el consumo de energía, mayores niveles de seguridad etc.

La mayoría de los modelos matemáticos usados tradicionalmente por teóricos y prácticos del control son lineales. De hecho, los modelos lineales son mucho más manejables que los no lineales, y pueden representar en forma precisa el comportamiento de sistemas reales en muchos casos útiles. Sin embargo, los avances tecnológicos actuales han generado una enorme variedad de nuevos problemas y aplicaciones que son no lineales en esencia. Tales fenómenos no lineales no se pueden describir mediante dinámica de modelos lineales — una razón ineludible para el uso de modelos no lineales y el desarrollo de conceptos y herramientas de sistemas no lineales de control.

Con este proyecto se pretende poner a prueba los conocimientos adquiridos en el máster de Automática e Informática Industrial y además profundizar en diversas materias relacionadas con el presente proyecto para conseguir los objetivos previamente establecidos.

CAPÍTULO 2: MODELO DINÁMICO DEL ROBOT ELÁSTICO

Todas las articulaciones se consideran flexibles, aunque se puedan encontrar situaciones en las que se usen diferentes dispositivos de transmisión. En el caso de existan reductores, se modelarán como si estos estuvieran colocados antes de que se produzca la deformación de la articulación.

Para obtener el modelo dinámico se han tenido en consideración las tres siguientes suposiciones:

1. La flexibilidad en las articulaciones es pequeña de modo que los efectos de flexibilidad tienen un comportamiento lineal.
2. Los rotores se modelan como cuerpos uniformes que tienen su centro de masa sobre el eje de rotación.
3. Los motores están situados en las articulaciones y en concreto en la posición anterior al eslabón accionado.

La elasticidad en la articulación i está modelada por un muelle de rigidez $K_i > 0$, que es torsional para las articulaciones rotacionales. La figura 2 muestra un enlace simple accionado por un motor a través de una articulación elástica rotacional. La suposición 2 implica que la matriz de inercia y el término de gravedad en el modelo dinámico del robot sean independientes de la posición angular de los motores. La suposición 3 es la configuración más típica, aunque la dislocación de los motores a lo largo de la estructura podría tener una buena influencia en la dinámica de movimiento.

El modelo dinámico de robots con articulaciones flexibles requiere el doble de variables de estado para caracterizar completamente la configuración de todos los cuerpos rígidos (motores y eslabones) que constituyen el brazo.

$$\Theta = \begin{pmatrix} q \\ \theta \end{pmatrix} \in R^{2N} \qquad (2.1)$$

donde q es el vector de las posiciones del eslabón y θ es el vector de las posiciones del motor (es decir, del rotor). Es obvio y lógico pensar que estas variables de posición tendrán un rango dinámico muy similar y que la cinemática del robot será una función solamente de las variables del eslabón q, de manera que, por la relación que existe, todas las cuestiones relacionadas con la cinemática directa / inversa serán idénticas al caso de los robots completamente rígidos.

Resulta interesante definir también la variable θ_m, es decir, el vector de las posiciones del motor antes de la reducción, que son las cantidades directamente medidas por los encoders montados en los motores. Así si el motor está directamente colocado sobre el eje de la articulación i-ésima, se tiene $\theta_{m,i} = n_i \theta_i$, donde ni ≥ 1 es la relación de reducción en la i-ésima articulación. Además, para i = 1, ..., N, la diferencia $\delta_i = q_i - \theta_i$ es la deformación en la i-ésima articulación, mientras que $\tau_{J,i} = K_i (\theta_i - q_i)$ es el par transmitido al i-ésimo eslabón a través del muelle.

Siguiendo un enfoque Lagrangiano, se describirán las contribuciones energéticas.

$$\mathcal{L} = K\left(\Theta, \dot{\Theta}\right) + P(\Theta) \tag{2.2}$$

Por una parte, la energía potencial se debe a la gravedad y la elasticidad de la articulación. En cuanto a la gravedad está relacionada con la posición del baricentro o centro de masas de los eslabones (cada uno de la masa m_i) y de los motores (de masa m_{r_i}). Debido a la suposición 2, este último será independiente de θ. De modo que se tiene

$$P_{grav} = P_{grav,eslabon}(q) + P_{grav,motor}(q) \tag{2.3}$$

Para la parte elástica de la energía potencial, debido a la suposición 1, tenemos

$$P_{elas} = \frac{1}{2}(q-\theta)^T K(q-\theta) \tag{2.4}$$

donde

$$K = diag(K_1, K_2, \dots, K_N)$$

y el resultado es

$$P(\Theta) = P_{grav}(q) + P_{elas}(q-\theta) \tag{2.5}$$

Por otra parte, la energía cinética del robot es la suma de las contribuciones del eslabón y del rotor. Para los eslabones, no hay diferencia con respecto al caso del robot rígido estándar y en general, será suficiente escribir

$$\begin{aligned} K_{eslabón} &= \sum_{i=1}^{N}\left(\frac{1}{2}m_{e_i}v_{C_i}^T v_{C_i} + \frac{1}{2}\omega_{e_i}^T\ I_{C_i}\omega_{e_i}\right) = \frac{1}{2}\sum_{i=1}^{N}\left(m_{e_i}\dot{q}^T J_{v_i}^T J_{v_i}\dot{q} + \dot{q}^T J_{w_i}^T I_{C_i} J_{w_i}\dot{q}\right) \\ &= \frac{1}{2}\dot{q}^T\sum_{i=1}^{N}\left(m_{e_i}J_{v_i}^T J_{v_i} + J_{w_i}^T I_{C_i} J_{w_i}\right)\dot{q} \end{aligned} \tag{2.6}$$

de modo que

$$M_L(q) = \sum_{i=1}^{N}\left(m_{e_i}J_{v_i}^T J_{v_i} + J_{w_i}^T I_{C_i} J_{w_i}\right) \tag{2.7}$$

$$K_{eslabón} = \frac{1}{2}\dot{q}^T M_L(q)\dot{q} \tag{2.8}$$

con la matriz de inercia del eslabón $M_L(q)$ positiva-definida y simétrica. Ahora faltaría definir los dos jacobianos J_{v_i} y J_{w_i}

$$J_{v_i} = \begin{bmatrix} \dfrac{\partial p_{C_i}}{\partial q_1} & \dfrac{\partial p_{C_i}}{\partial q_2} & \cdots & \dfrac{\partial p_{C_i}}{\partial q_i} & 0 & 0 & \cdots & 0 \end{bmatrix} \tag{2.9}$$

$$J_{w_i} = \begin{bmatrix} \bar{\varepsilon_1}z_1 & \bar{\varepsilon_2}z_2 & \cdots & \bar{\varepsilon_i}z_i & 0 & 0 & \cdots & 0 \end{bmatrix} \tag{2.10}$$

donde p_{C_i} es la distancia desde el origen hasta el centro de masas del eslabón i-ésimo; $\bar{\varepsilon_1}$ hace referencia al hecho de que si la articulación es prismática ($\bar{\varepsilon_1} = 0$) o de revolución ($\bar{\varepsilon_1} = 1$) y z_i es la tercera columna de la matriz de transformación de 0 hasta i.

Por otro lado, para la cinemática de los rotores, se necesita más detalle, ya que aquí la elasticidad sí que contribuye:

$$K_{rotor} = \sum_{i=1}^{N} \tau_{rotor_i} = \sum_{i=1}^{N} \left(\frac{1}{2} m_{r_i} {v_{r_i}}^T v_{r_i} + \frac{1}{2} \omega_{r_i}^T \, I_{r_i} \omega_{r_i} \right) \quad (2.11)$$

donde v_{r_i} es la velocidad lineal del centro de masa del i-ésimo rotor y ω_{r_i} es la velocidad angular del cuerpo del rotor i-ésimo. Todas las cantidades en las contribuciones angulares en (2.11) se expresan convenientemente en el plano de referencia de R_i. Según la suposición 2, la matriz de inercia del rotor es entonces diagonal

$$I_{r_i} = diag(I_{r_{i_{xx}}}, I_{r_{i_{yy}}}, I_{r_{i_{zz}}})$$

Con $I_{r_{i_{xx}}} = I_{r_{i_{yy}}}$. Obsérvese que v_{r_i} sólo puede expresarse como una función de q y $\dot{q}$. Mientras que la velocidad angular del i-ésimo rotor, debido a la suposición 3, tiene la expresión general

$$\omega_{r_i} = \sum_{j=1}^{i-1} J_{r_i,j}(q)\, \dot{q}_j + \begin{pmatrix} 0 \\ 0 \\ \dot{\theta}_{m,i} \end{pmatrix} \quad (2.12)$$

donde $J_{r_i,j}(q)$ es la j-ésima columna del Jacobiano que relaciona las velocidades del eslabón $\dot{q}$ con la velocidad angular del i-ésimo rotor en la cadena del robot. Al sustituir (8) en (7) y expresar $\dot{\theta}_m$ en términos de $\dot{\theta}$, se puede demostrar que

$$K_{rotor} = \frac{1}{2} \dot{q}^T [M_R(q) + S(q) B^{-1} S^T(q)] \dot{q} + \dot{q}^T S(q) \dot{\theta}^T B \dot{\theta} \quad (2.13)$$

donde B es la matriz de inercia diagonal constante que recoge los componentes inerciales de los rotores $I_{r_{i_{zz}}}$alrededor de sus ejes giratorios, $M_R(q)$ contiene las masas del rotor (y, posiblemente, los componentes inerciales del rotor a lo largo de los otros ejes principales), y la matriz cuadrada $S(q)$ expresa los acoplamientos inerciales entre los rotores y los eslabones anteriores en la cadena del robot.

A continuación, se muestra un ejemplo simple que ilustra la derivación y las expresiones reales de los términos que aparecen en (2.13). También se incluyen los elementos de reducción para mostrar cómo la componente inercial del rotor alrededor del eje giratorio aparecerá en la energía cinética, ponderada por diferentes potencias de las relaciones de reducción. Consideremos un robot plano con dos articulaciones elásticas de rotación, un primer eslabón de longitud l_1 y motores directamente montados en los ejes de la articulación. Las energías cinéticas de los dos rotores son

$$K_{rotor_1} = \frac{1}{2} I_{r_{1_{zz}}} {\dot{\theta}_{m,1}}^2 = \frac{1}{2} I_{r_{1_{zz}}} n_1^2 \dot{\theta}_1^2$$

$$K_{rotor_2} = \frac{1}{2} m_{r_2} l_1^2 \dot{q}_1^2 + \frac{1}{2} I_{r_{2_{zz}}} (\dot{q}_1 - \dot{\theta}_{m,2})^2 = \frac{1}{2} m_{r_2} l_1^2 \dot{q}_1^2 + \frac{1}{2} I_{r_{2_{zz}}} \left(\dot{q}_1^2 + 2 n_2 \dot{q}_1 \dot{\theta}_1^2 \right)$$

dando lugar a

$$B = \begin{pmatrix} I_{r_{1zz}} n_1^2 & 0 \\ 0 & I_{r_{2zz}} n_2^2 \end{pmatrix}, \qquad S = \begin{pmatrix} 0 & I_{r_{2zz}} n_2 \\ 0 & 0 \end{pmatrix},$$

$$M_R = \begin{pmatrix} m_{r_2} l_1^2 & 0 \\ 0 & 0 \end{pmatrix}, \qquad SB^{-1}S^T = \begin{pmatrix} I_{r_{2zz}} & 0 \\ 0 & 0 \end{pmatrix}$$

En este caso, la matriz S así como M_R son constantes. Obsérvese que para grandes relaciones de reducción n_i, el efecto inercial dominante debido a los rotores viene dado por la matriz B.

En general, como consecuencia de la suposición 3, la matriz $S(q)$ siempre tiene una estructura estrictamente triangular-superior con una dependencia en cascada de sus elementos no nulos:

$$S(q) = \begin{pmatrix} 0 & S_{12} & S_{13}(q_2) & S_{14}(q_2, q_3) & \cdots & \cdots & S_{1N}(q_2, \dots, q_{N-1}) \\ 0 & 0 & S_{23} & S_{24}(q_3) & \cdots & \cdots & S_{2N}(q_3, \dots, q_{N-1}) \\ 0 & 0 & 0 & \ddots & \ddots & \vdots & S_{3N}(q_4, \dots, q_{N-1}) \\ \vdots & \vdots & \vdots & 0 & \ddots & & \vdots \\ 0 & 0 & 0 & \cdots & 0 & S_{N-2,N-1} & S_{1N}(q_2, \dots, q_{N-1}) \\ 0 & 0 & 0 & \cdots & 0 & 0 & S_{N-1,N} \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 \end{pmatrix} \quad (1)$$

Resumiendo, la energía cinética total del robot es

$$K = \frac{1}{2}\dot{\Theta}^T M(\Theta)\dot{\Theta} = \frac{1}{2}(\dot{q}^T \quad \dot{\Theta}^T)\begin{pmatrix} M(q) & S(q) \\ S^T(q) & B \end{pmatrix}\begin{pmatrix} \dot{q} \\ \dot{\theta} \end{pmatrix} \quad (2.15)$$

donde

$$M(q) = M_L(q) + M_R(q) + S(q)B^{-1}S^T(q) \quad (2.16)$$

Como se anticipó, la matriz de inercia total M del robot depende solamente de q. Utilizando las ecuaciones de Lagrange finalmente se obtiene el modelo dinámico completo

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{\Theta}}\right) + \frac{\partial \mathcal{L}}{\partial \Theta} = \tau$$

$$\begin{pmatrix} M(q) & S(q) \\ S^T(q) & B \end{pmatrix}\begin{pmatrix} \ddot{q} \\ \ddot{\theta} \end{pmatrix} + \begin{pmatrix} c(q, \dot{q}) + c_1(q, \dot{q}, \dot{\theta}) \\ c_2(q, \dot{q}) \end{pmatrix} + \begin{pmatrix} g(q) + K(q - \theta) \\ K(q - \theta) \end{pmatrix} = \begin{pmatrix} 0 \\ \tau \end{pmatrix} \quad (2.17)$$

donde los términos inerciales (relacionados con la matriz de inercia total $M(q)$), los términos de Coriolis y de centrífuga (denominados colectivamente por $c_{tot}\ (\Theta, \dot{\Theta})$, y los términos potenciales $(\partial U\ (\Theta)\ /\ \partial \Theta)^T$ han sido escritas por separado. En particular, $g(q) = (\partial U_{grav}\ (q)\ /\ \partial q)^T$ Mientras que $\tau_J = K\ (\theta - q)$ es el par elástico transmitido a través de las articulaciones.

De los términos dependientes de la velocidad en (2.17) del vector $c_{tot}(q,\dot{q})$, todos los elementos son independientes de la posición del motor θ. La dependencia específica de los vectores c, c_1 y c_2 se deriva de la expresión general de los componentes de c_{tot} basada en los símbolos de Christoffel. En particular, los vectores de velocidad $c_1(q,\dot{q},\dot{\theta})$ y $c_2(q,\dot{q})$ contienen términos que surgen de la presencia de la matriz $S(q)$. En efecto, para obtener los términos de Coriolis y de centrifuga c_{tot} es necesario primero haber obtenido $M(\Theta)$ por la relación que se establece a partir de la ecuación de Lagrange

$$c_{tot}(q,\dot{q}) == \begin{pmatrix} \dot{m}_{11} & \dot{m}_{12} & \cdots & \dot{m}_{1N} \\ \dot{m}_{21} & \dot{m}_{22} & \cdots & \dot{m}_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \dot{m}_{N1} & \dot{m}_{N2} & \cdots & \dot{m}_{NN} \end{pmatrix}\dot{\Theta} - \frac{1}{2}\begin{bmatrix} \dot{\Theta}^T \begin{pmatrix} m_{111} & m_{121} & \cdots & m_{1N1} \\ m_{211} & m_{221} & \cdots & m_{2N1} \\ \vdots & \vdots & \ddots & \vdots \\ m_{N11} & m_{N21} & \cdots & m_{NN1} \end{pmatrix}\dot{\Theta} \\ \dot{\Theta}^T \begin{pmatrix} m_{112} & m_{122} & \cdots & m_{1N2} \\ m_{212} & m_{222} & \cdots & m_{2N2} \\ \vdots & \vdots & \ddots & \vdots \\ m_{N12} & m_{N22} & \cdots & m_{NN2} \end{pmatrix}\dot{\Theta} \\ \vdots \\ \dot{\Theta}^T \begin{pmatrix} m_{11N} & m_{12N} & \cdots & m_{1NN} \\ m_{21N} & m_{22N} & \cdots & m_{2NN} \\ \vdots & \vdots & \ddots & \vdots \\ m_{N1N} & m_{N2N} & \cdots & m_{NNN} \end{pmatrix}\dot{\Theta} \end{bmatrix} \tag{2.18}$$

donde

$$\dot{m}_{11} = m_{111}\dot{\Theta}_1 + m_{112}\dot{\Theta}_2 + \cdots + m_{11N}\dot{\Theta}_N \qquad con \rightarrow\ m_{ijk} = \frac{\partial m_{ij}}{\partial \Theta_k}$$

que al emplear los símbolos Christoffel, podemos obtener las fuerzas de Coriolis y de centrifuga de la siguiente forma

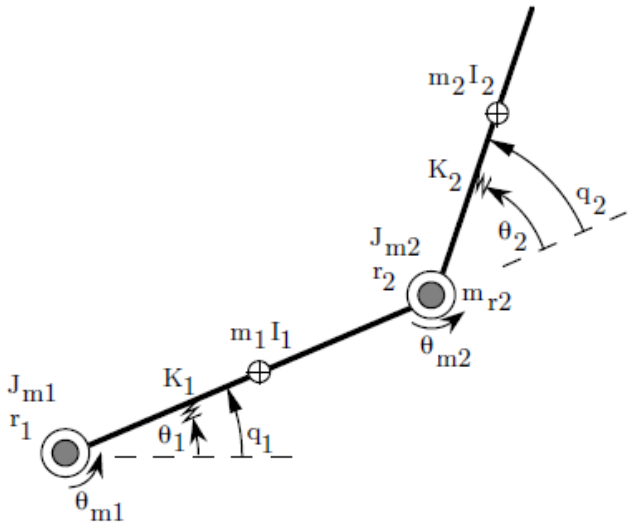
$$c_{tot}(q,\dot{q}) = \begin{bmatrix} b_{111} & b_{122} & \cdots & b_{1NN} \\ b_{211} & b_{222} & \cdots & b_{2NN} \\ \vdots & \vdots & \ddots & \vdots \\ b_{N11} & b_{N22} & \cdots & b_{NNN} \end{bmatrix}\begin{bmatrix} \dot{\Theta}_1^2 \\ \dot{\Theta}_2^2 \\ \vdots \\ \dot{\Theta}_N^2 \end{bmatrix} + \begin{bmatrix} 2b_{112} & 2b_{113} & \cdots & 2b_{1(N-1)N} \\ 2b_{212} & 2b_{213} & \cdots & 2b_{2(N-1)N} \\ \vdots & \vdots & \ddots & \vdots \\ 2b_{N12} & 2b_{N13} & \cdots & 2b_{N(N-1)N} \end{bmatrix}\begin{bmatrix} \dot{\Theta}_1\dot{\Theta}_2 \\ \dot{\Theta}_1\dot{\Theta}_3 \\ \vdots \\ \dot{\Theta}_{N-1}\dot{\Theta}_N \end{bmatrix} \tag{2.19}$$

donde cada

$$b_{ijk} = \frac{1}{2}\left(m_{ijk} + m_{ikj} + m_{jki}\right), \quad con \rightarrow \quad m_{ijk} = \frac{\partial m_{ij}}{\partial \Theta_k}; \quad m_{ikj} = \frac{\partial m_{ik}}{\partial \Theta_j} \quad y \quad m_{kji} = \frac{\partial m_{kj}}{\partial \Theta_i}$$

2.1. Caso de estudio

A continuación, se presenta el problema que se desea resolver. Se trata de un brazo robótico formado por dos eslabones y dos motores que llevan harmonic drives. Debido a ello, las dos articulaciones son elásticas. El problema se resolverá teniendo en cuenta la gravedad, las inercias tanto de los eslabones como de los motores, y por supuesto teniendo en cuenta los efectos de elasticidad. Efectos de rozamientos y viscosidad se desprecian.



Datos:

L1 = 0.5 m
m1 = 10 kg
d1 = 0.25 m (dist. c.d.m)

L2 = 0.5 m
m2 = 5 kg
mr2 = 2 kg
d2 = 0.25 m (dist. c.d.m)

K1 = 10^4 Nm/rad
r1 = 50 (ratio reducción)

K2 = 10^4 Nm/rad
r2 = 40 (ratio reducción)

Figura 3. Modelo del robot.

En primer lugar, vamos a empezar con lo sencillo, calculando la inercia de cada eslabón y a su vez la de cada rotor, con los datos que se proporcionan.

$$I_1 = \frac{m_1 L_1^2}{12} \qquad I_2 = \frac{m_2 L_2^2}{12}$$

$$J_{m_2} = \frac{(I_2 + m_2 * d_2^2)}{r_2^2} \qquad J_{m_1} = \frac{(I_1 + m_1 d_1^2 + m_{r_2} L_1^2)}{r_1^2}$$

Vamos a resolver el problema empleando el método de Lagrange, por lo tanto, tendremos que calcular la energía cinética y la energía potencia. Una vez obtenidas esas dos energías sería cuestión de realizar las correspondientes derivaciones para obtener la dinámica del brazo robótico.

Empezando por la energía cinética, tal y como vimos en el apartado anterior para los dos rotores se puede obtener de la siguiente forma, respetando la ecuación (2.11)

$$K_{rotor_1} = \frac{1}{2} J_{m_1} r_1^2 \dot{\theta}_1^2$$

$$K_{rotor_2} = \frac{1}{2} m_{r_2} L_1^2 \dot{q}_1^2 + \frac{1}{2} J_{m_2} \left(\dot{q}_1^2 + 2 r_2 \dot{q}_1 \dot{\theta}_1^2 \right)$$

de modo que las matrices M_R, B, S y $SB^{-1}S^T$, de acuerdo con (2.13) serían:

$$B = \begin{pmatrix} J_{m_1} r_1^2 & 0 \\ 0 & J_{m_2} r_2^2 \end{pmatrix};\ S = \begin{pmatrix} 0 & J_{m_2} r_2 \\ 0 & 0 \end{pmatrix};\ SB^{-1}S^T = \begin{pmatrix} J_{m_2} & 0 \\ 0 & 0 \end{pmatrix};\ M_R = \begin{pmatrix} m_{r_2} L_1^2 & 0 \\ 0 & 0 \end{pmatrix}$$

El segundo paso sería obtener la energía cinética de los eslabones, para ello, emplearemos las ecuaciones (8) y (9). Nótese que el procedimiento sería idéntico para un robot rígido. De manera que dada la siguiente expresión

$$M_L(q) = m_{e_1} J_{v_1}^T J_{v_1} + J_{w_1}^T I_{C_1} J_{w_1} + m_{e_2} J_{v_2}^T J_{v_2} + J_{w_2}^T I_{C_2} J_{w_2}$$

donde

$$J_{v_1} = \begin{bmatrix} \frac{\partial p_{C_1}}{\partial q_1} & 0 \end{bmatrix} \qquad J_{v_2} = \begin{bmatrix} \frac{\partial p_{C_2}}{\partial q_1} & \frac{\partial p_{C_2}}{\partial q_2} \end{bmatrix}$$

y

$$J_{w_1} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{bmatrix} \qquad J_{w_2} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}$$

para calcular J_{v_1} y J_{v_2} es necesario hallar la distancia desde el origen hasta el centro de masas de cada eslabón, esto es p_{C_1} y p_{C_2}, de modo que de acuerdo con la figura 3 se tiene para ambos eslabones

$${}_{C_1}^{0}P = \begin{bmatrix} d_1 C_1 \\ d_1 S_1 \\ 0 \end{bmatrix} \qquad {}_{C_2}^{0}P = \begin{bmatrix} L_1 C_1 + d_1 (C_1 C_2 - S_1 S_2) \\ L_1 S_1 + d_1 (C_1 S_2 + S_1 C_2) \\ 0 \end{bmatrix}$$

ahora ya podemos calcular J_{v_1} y J_{v_2}

$$J_{v_1} = \begin{bmatrix} -d_1 S_1 & 0 \\ d_1 C_1 & 0 \\ 0 & 0 \end{bmatrix} \qquad J_{v_2} = \begin{bmatrix} -L_1 S_1 - d_1 (S_1 C_2 + C_1 S_2) & -d_1 (C_1 S_2 + S_1 C_2) \\ L_1 C_1 - d_1 (S_1 S_2 - C_1 C_2) & d_1 (C_1 C_2 - S_1 S_2) \\ 0 & 0 \end{bmatrix}$$

se realizan por separado las siguientes multiplicaciones para calcular $M_L(q)$

$$m_{e_1} J_{v_1}^T J_{v_1} = \begin{bmatrix} m_1 d_1^2 & 0 \\ 0 & 0 \end{bmatrix}$$

$$m_{e_2} J_{v_2}^T J_{v_2} = \begin{bmatrix} m_2 L_1^2 + m_2 d_2^2 + 2 m_2 L_1 d_2 \cos(q_2) & m_2 d_2^2 + m_2 L_1 d_2 \cos(q_2) \\ m_2 d_2^2 + m_2 L_1 d_2 \cos(q_2) & m_2 d_2^2 \end{bmatrix}$$

$$J_{w_1}^T I_{C_1} J_{w_1} = \begin{bmatrix} I_1 & 0 \\ 0 & 0 \end{bmatrix}$$

$$J_{w_2}^T I_{C_2} J_{w_2} = \begin{bmatrix} I_2 & I_2 \\ I_2 & I_2 \end{bmatrix}$$

sumando todo, obtenemos la matriz $M_L(q)$ y con ella energía cinética de los eslabones

$$M_L(q) = \begin{bmatrix} I_1 + I_2 + m_2L_1^2 + m_2d_2^2 + 2m_2L_1d_2\cos(q_2) & I_2 + m_2d_2^2 + m_2L_1d_2\cos(q_2) \\ I_2 + m_2d_2^2 + m_2L_1d_2\cos(q_2) & I_2 + m_2d_2^2 \end{bmatrix}$$

Tal y como vimos en la expresión (16) para obtener $M(q)$ debemos sumar $M_L(q)$ con M_R y con $SB^{-1}S^T$, pero antes para simplificar expresiones, llamemos

$$a_1 = I_1 + m_1d_1^2 + \left(m_{r_2} + m_2\right)L_1^2 + J_{m_2}m_2d_2^2$$

$$a_2 = I_2 + m_2d_2^2$$

$$a_3 = m_2L_1d_2$$

ahora ya si

$$M(q) = \begin{pmatrix} a_1 + 2a_3\cos(q_2) & a_2 + a_3\cos(q_2) \\ a_2 + a_3\cos(q_2) & a_2 \end{pmatrix}$$

Tenemos todo lo necesario para construir la matriz $M(\Theta)$ que es la matriz de masas, es una matriz simétrica, que expresa la conexión entre la derivada temporal $\dot{\Theta}$ del vector de coordenadas generalizado Θ de un sistema y la energía cinética K de ese sistema.

$$M(\Theta) = \begin{pmatrix} M(q) & S(q) \\ S^T(q) & B \end{pmatrix} = \begin{pmatrix} a_1 + 2a_3\cos(q_2) & a_2 + a_3\cos(q_2) & 0 & J_{m_2}r_2 \\ a_2 + a_3\cos(q_2) & a_2 & 0 & 0 \\ 0 & 0 & J_{m_1}r_1^2 & 0 \\ J_{m_2}r_2 & 0 & 0 & J_{m_2}r_2^2 \end{pmatrix}$$

El segundo paso sería hallar los términos de Coriolis y centrífuga que, como ya vimos anteriormente se obtienen a partir de la matriz $M(\Theta)$. Para el actual caso con 2 g.d.l vemos que únicamente $M(q)$ es variable, $S(q)$ es constantes y B constante también. Empleando la ecuación (2.19) con la metodología de Christoffel obtenemos que

$$C_{tot}(q,\dot{q})\dot{\Theta} = \begin{bmatrix} -a_3\sin(q_2)[\dot{q}_2^2 + 2\dot{q}_2\dot{q}_1] \\ a_3\sin(q_2)\dot{q}_1^2 \\ 0 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & -a_3\sin(q_2)[\dot{q}_2 + 2\dot{q}_1] & 0 & 0 \\ a_3\sin(q_2)\dot{q}_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}$$

Ya tenemos la energía cinética, ahora tenemos que obtener la energía potencial. Esta depende de la gravedad y la elasticidad de las articulaciones. Por una parte, sabiendo que la gravedad es $g(q) = (\partial U_{grav}(q) / \partial q)^T$ tenemos que

$$\begin{aligned} g(q) &= -\left[J_{v_1}^T m_1 g + J_1^T m_{r_2} g + J_{v_2}^T m_2 g\right] \\ &= -\begin{bmatrix} -d_1 S_1 & d_1 C_1 & 0 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ -m_1 g \\ 0 \end{bmatrix} - \begin{bmatrix} -L_1 S_1 & L_1 C_1 & 0 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ -m_{r_2} g \\ 0 \end{bmatrix} \\ &\quad - \begin{bmatrix} -L_1 S_1 - d_1(S_1 C_2 + C_1 S_2) & L_1 C_1 - d_1(S_1 S_2 - C_1 C_2) & 0 \\ -d_1(C_1 S_2 + S_1 C_2) & d_1(C_1 C_2 - S_1 S_2) & 0 \end{bmatrix}\begin{bmatrix} 0 \\ -m_2 g \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} m_1 g d_1 C_1 \\ 0 \end{bmatrix} + \begin{bmatrix} m_{r_2} g L_1 C_1 \\ 0 \end{bmatrix} + \begin{bmatrix} m_2 g[L_1 C_1 - d_1(S_1 S_2 - C_1 C_2)] \\ m_2 g d_1(C_1 C_2 - S_1 S_2) \end{bmatrix} \end{aligned}$$

que al simplificar y sumar todo da como resultado

$$g = \begin{bmatrix} m_1 g d_1 C_1 + m_{r_2} g L_1 C_1 + m_2 g[L_1 C_1 - d_1(S_1 S_2 - C_1 C_2)] \\ m_2 g d_1(C_1 C_2 - S_1 S_2) \end{bmatrix}$$

Por otra parte, la elasticidad, de acuerdo con la expresión (2.4) sería

$$e = K(q - \theta) = \begin{bmatrix} K & 0 \\ 0 & K \end{bmatrix}\begin{bmatrix} q_1 - \theta_1 \\ q_2 - \theta_2 \end{bmatrix} = \begin{bmatrix} K(q_1 - \theta_1) \\ K(q_2 - \theta_2) \end{bmatrix}$$

Ahora bien, sumando las dos contribuciones se obtiene la siguiente matriz

$$P(\Theta) = \begin{bmatrix} m_1 g d_1 C_1 + m_{r_2} g L_1 C_1 + m_2 g[L_1 C_1 - d_1(S_1 S_2 - C_1 C_2)] + K(q_1 - \theta_1) \\ m_2 g d_1(C_1 C_2 - S_1 S_2) + K(q_2 - \theta_2) \\ -K(q_1 - \theta_1) \\ -K(q_2 - \theta_2) \end{bmatrix}$$

Ya tenemos resuelta la dinámica de nuestro robot, el siguiente paso sería pasar este modelo al espacio de estados que será la forma que emplearemos para realizar el control. Para terminar, recordemos que la expresión final de la dinámica para este robot es:

$$\boldsymbol{M(\Theta)\ddot{\Theta} + C_{tot}(q, \dot{q})\dot{\Theta} + P(\Theta) = B\tau}$$

con

$$B\tau = \begin{bmatrix} 0 \\ 0 \\ \tau_1 \\ \tau_2 \end{bmatrix}$$

y con todas las matrices anteriormente obtenidas.

2.2. Linealización y obtención del modelo en espacio de estados

Como las técnicas de análisis y control lineales son bien conocidas, siempre es conveniente, al analizar un sistema no lineal, comenzar linealizando el sistema alrededor de algún punto de equilibrio y estudiar el sistema lineal resultante.

Estamos trabajando con un sistema dinámico, modelado por un número finito de ecuaciones diferenciales ordinarias de primer orden acopladas entre sí, que representaremos en forma compacta con la ecuación diferencial vectorial de primer orden

$$\dot{x} = f(x, u)$$

donde $x \in R^n$ es el vector de estado y $u \in R^p$ es el vector de entradas (de control). A veces vamos a considerar también una ecuación de salida

$$y = h(x, u)$$

donde $y \in R^m$ es un vector de variables de interés, por ejemplo variables físicamente medibles o variables que deseamos se comporten de alguna forma especial.

Para nuestro brazo robótico, $f(x, u)$ tendría la siguiente forma

$$\dot{x} = \begin{bmatrix} \dot{\Theta} \\ M^{-1}(\Theta)[Bu - C_{tot}(q, \dot{q}) - P(\Theta)] \end{bmatrix}$$

A partir del modelo dinámico se pueden extraer cuatro ecuaciones de las que obtenemos 8 variables de estado para el caso del robot con articulaciones elásticas. A partir de ellas podemos obtener el modelo en el espacio de estados.

$$3\ddot{q}_1 + 1.25\ddot{q}_1 C_2 + 0.417\ddot{q}_2 + 0.625\ddot{q}_2 C_2 + 0.01\ddot{\theta}_2 - 0.625 s_2(\dot{q}_2^2 + 2\dot{q}_2\dot{q}_1) + (58{,}8 + 12{,}25C_2)C_1 - 12.25S_1S_2 + 10000(q_1 - \theta_1) = 0 \quad (1)$$

$$0.417\ddot{q}_1 + 0.625\ddot{q}_1 C_2 + 0.417\ddot{q}_2 + 0.625\dot{q}_1^2 S_2 + 12.25(C_1C_2 - S_1S_2) + 10000(q_2 - \theta_2) = 0 \quad (2)$$

$$1.33\ddot{\theta}_1 - 10000(q_1 - \theta_1) = \tau_1 \quad (3)$$

$$0.01\ddot{q}_1 + 0.417\ddot{\theta}_2 - 10000(q_2 - \theta_2) = \tau_2 \quad (4)$$

donde

$$C_1 = \cos(q_1), \;\; S_1 = \sin(q_1), \;\; C_2 = \cos(q_2), \;\; S_2 = \sin(q_2)$$

Ahora bien, para obtener el modelo en el espacio de estados siempre es recomendable realizar las siguientes transformaciones para expresar todo con "x" y sus correspondientes derivadas.

$x_1 = q_1$	$x_5 = \dot{q}_1$	$\dot{x}_1 = x_5 = \dot{q}_1$	$\dot{x}_5 = \ddot{q}_1$	$u_1 = \tau_1$
$x_2 = q_2$	$x_6 = \dot{q}_2$	$\dot{x}_2 = x_6 = \dot{q}_2$	$\dot{x}_6 = \ddot{q}_2$	$u_2 = \tau_2$
$x_3 = \theta_1$	$x_7 = \dot{\theta}_1$	$\dot{x}_3 = x_7 = \dot{\theta}_1$	$\dot{x}_7 = \ddot{\theta}_1$	
$x_4 = \theta_1$	$x_8 = \dot{\theta}_2$	$\dot{x}_4 = x_8 = \dot{\theta}_2$	$\dot{x}_8 = \ddot{\theta}_2$	

que al reemplazar en las cuatro ecuaciones anteriores pasamos a tener

$$\dot{x}_5(3 + 1.25C_2) + \dot{x}_6(0.417 + 0.625C_2) + 0.01\dot{x}_8 - 0.625S_2({x_6}^2 + 2x_6x_5) + (58{,}8 + 12{,}25C_2)C_1 - 12.25S_1S_2 + 10000(x_1 - x_3) = 0 \quad (1)$$

$$\dot{x}_5(0.417 + 0.625C_2) + 0.417\dot{x}_6 + 0.625{x_5}^2S_2 + 12.25(C_1C_2 - S_1S_2) + 10000(x_2 - x_4) = 0 \quad (2)$$

$$1.33\dot{x}_7 - 10000(x_1 - x_3) = u_1 \quad (3)$$

$$0.01\dot{x}_5 + 0.417\dot{x}_8 - 10000(x_2 - x_4) = u_2 \quad (4)$$

Despejando la ecuación (3) obtenemos $\ddot{x}_7$; de la ecuación (2) despejamos $\dot{x}_6$ y de la ecuación (4) despejamos $\dot{x}_8$

$$\ddot{x}_7 = \frac{u_1 + 10000(x_1 - x_3)}{1.33}$$

$$\dot{x}_6 = \frac{-0.625{x_5}^2S_2 - 12.25(C_1C_2 - S_1S_2) - 10000(x_2 - x_4) - \dot{x}_5(0.417 + 0.625C_2)}{0.417}$$

$$\dot{x}_8 = \frac{u_2 + 10000(x_2 - x_4) - 0.01\dot{x}_5}{0.417}$$

Si sustituimos $\dot{x}_6$ y $\dot{x}_8$ en la ecuación (1) podemos obtener $\dot{x}_5$, sin embargo las expresiones se hacen muy largas, además, después tendríamos que volver a sustituir $\dot{x}_5$ en $\dot{x}_6$ y $\dot{x}_8$ para obtener las expresiones finales de la derivada de los estados. Hacer esto a mano sería muy costoso, en cambio si lo hacemos con un programa como es Matlab podríamos obtenerlos de una forma mucho más rápida y fácil. Si planteamos el problema en Matlab de la siguiente forma:

```
x_dot = inv(M)*(tau-Cc*q_prim-P)
```

con las matrices M (matriz de masas), Cc (centrípeta y coriolis), P (gravedad más efectos de elasticidad) y el vector tau (los dos pares), todas ellas definidas y están en función de las variables o estados empleando el *Symbolic Math Toolbox* de Matlab. Obtenemos `x_dot que` contendría los estados $\dot{x}_5$, $\dot{x}_6$, $\dot{x}_7$ y $\dot{x}_8$[1]

Linealizando alrededor de un punto de equilibrio con una serie de Taylor de la que consideraremos solo los términos de primer orden despreciando los demás términos se obtiene

$$\dot{x} \approx f(x_0, u_0) + \left.\frac{\partial f}{\partial x}\right|_{x_0,u_0}(x - x_0) + \left.\frac{\partial f}{\partial u}\right|_{x_0,u_0}(u - u_0)$$

[1] En el archivo "linealizacion.m" están todos los pasos seguidos para obtener $\dot{x}_5$, $\dot{x}_6$, $\dot{x}_7$ y $\dot{x}_8$

de la expresión anterior podríamos obtener los matices A y B. Observe que se trata de una matriz Jacobiana en la que se deben realizar todas las derivadas parciales con respecto a los distintos estados (x) y las entradas (u).

$$A = \frac{\partial f}{\partial x} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \frac{\partial \dot{x}_5}{\partial x_1} & \frac{\partial \dot{x}_5}{\partial x_2} & \frac{\partial \dot{x}_5}{\partial x_3} & \frac{\partial \dot{x}_5}{\partial x_4} & \frac{\partial \dot{x}_5}{\partial x_5} & \frac{\partial \dot{x}_5}{\partial x_6} & \frac{\partial \dot{x}_5}{\partial x_7} & \frac{\partial \dot{x}_5}{\partial x_8} \\ \frac{\partial \dot{x}_6}{\partial x_1} & \frac{\partial \dot{x}_6}{\partial x_2} & \frac{\partial \dot{x}_6}{\partial x_3} & \frac{\partial \dot{x}_6}{\partial x_4} & \frac{\partial \dot{x}_6}{\partial x_5} & \frac{\partial \dot{x}_6}{\partial x_6} & \frac{\partial \dot{x}_6}{\partial x_7} & \frac{\partial \dot{x}_6}{\partial x_8} \\ \frac{\partial \dot{x}_7}{\partial x_1} & \frac{\partial \dot{x}_7}{\partial x_2} & \frac{\partial \dot{x}_7}{\partial x_3} & \frac{\partial \dot{x}_7}{\partial x_4} & \frac{\partial \dot{x}_7}{\partial x_5} & \frac{\partial \dot{x}_7}{\partial x_6} & \frac{\partial \dot{x}_7}{\partial x_7} & \frac{\partial \dot{x}_7}{\partial x_8} \\ \frac{\partial \dot{x}_8}{\partial x_1} & \frac{\partial \dot{x}_8}{\partial x_2} & \frac{\partial \dot{x}_8}{\partial x_3} & \frac{\partial \dot{x}_8}{\partial x_4} & \frac{\partial \dot{x}_8}{\partial x_5} & \frac{\partial \dot{x}_8}{\partial x_6} & \frac{\partial \dot{x}_8}{\partial x_7} & \frac{\partial \dot{x}_8}{\partial x_8} \end{bmatrix}_{x_0,u_0}$$

$$B = \frac{\partial f}{\partial u} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \frac{\partial \dot{x}_5}{\partial u_1} & \frac{\partial \dot{x}_5}{\partial u_2} \\ \frac{\partial \dot{x}_6}{\partial u_1} & \frac{\partial \dot{x}_6}{\partial u_2} \\ \frac{\partial \dot{x}_7}{\partial u_1} & \frac{\partial \dot{x}_7}{\partial u_2} \\ \frac{\partial \dot{x}_8}{\partial u_1} & \frac{\partial \dot{x}_8}{\partial u_2} \end{bmatrix}_{x_0,u_0}$$

Para ello, primero debemos obtener los puntos de equilibrio que serán los puntos donde evaluaremos el resultado de las derivadas parciales. Un punto $x = x^*$ en el espacio de estado es un punto de equilibrio (PE) de (1.3), si tiene la propiedad de que cuando el estado inicial del sistema es x^* , el estado permanece en x^* en todo tiempo futuro. Entonces podemos decir que los puntos de equilibrio de (1.3) son las raíces de la ecuación

$$f(x, u) = 0$$

de modo que cuando

$\dot{x}_1 = \dot{x}_2 = \dot{x}_3 = \dot{x}_4 = 0 \quad se\ obtiene\ que \rightarrow \quad x_5 = x_6 = x_7 = x_8 = 0$

a partir de la ecuación (3)

$\dot{x}_7 = u_1 = 0 \qquad se\ obtiene\ que \rightarrow \quad x_1 = x_3$

a partir de la ecuación (4)

$u_2 = \dot{x}_5 = \dot{x}_8 = 0 \qquad se\ obtiene\ que \rightarrow \quad x_2 = x_4$

Ahora bien, a partir de la ecuación (1) y (2) y teniendo en cuenta los datos anteriores se obtiene un sistema de dos inecuaciones. Si nos fijamos bien, vemos que se corresponde con los efectos de gravedad. Así pues obtendremos el valor de x_1 y x_2 siendo el mismo para x_3 y x_4 respectivamente.

$$\begin{cases} 58.8\cos(x_1) + 12.25\cos(x_2)\cos(x_1) - 12.25sin(x_1)sin(x_2) = 0 \\ 12.25\cos(x_2)\cos(x_1) - 12.25sin(x_1)sin(x_2) = 0 \end{cases}$$

resolviendo

$$\cos(x_1) = 0 \rightarrow x_1 = \ x_3 = \pm 1.57\ rad$$

$$-12.25sin(x_2) = 0 \rightarrow x_2 = \ x_4 = 0\ rad \ | \ \pi\ rad$$

Llegado a este punto, a partir de las matrices jacobianas A y B anteriormente descritas evaluadas en los puntos de equilibrio: $x_1 = x_3 = -1.57 \ \ y\ x_2 = x_4 = x_5 = x_6 = x_7 = x_8 = 0$, así como $u_1 = u_2 = 0$ se obtiene el modelo linealizado.

$$\Delta\dot{x} = A \cdot \Delta x + B \cdot \Delta u$$

con

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ -6100 & 15049 & 6076 & -15038 & 0 & 0 & 0 & 0 \\ 15221 & -61651 & -15190 & 61595 & 0 & 0 & 0 & 0 \\ 7500 & 0 & -7500 & 0 & 0 & 0 & 0 & 0 \\ 153 & 23624 & -152 & -23624 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & -0.0152 \\ 0 & 0.0380 \\ 0.75 & 0 \\ 0 & 2.4004 \end{bmatrix}$$

Nótese que el mismo resultado se podría haber obtenido a partir de la siguiente expresión teniendo en cuenta las propiedades de las matrices simétricas y aplicando la regla de la cadena evaluada sobre el punto de equilibrio.

$$\begin{bmatrix} \Delta\dot{q} \\ \Delta\ddot{q} \end{bmatrix} = \begin{bmatrix} 0 & I \\ -M_{|q=0}^{-1} \cdot \dfrac{\partial G}{\partial q_{|q=0}} & -M_{|q=0}^{-1} \cdot C_{|q=0} \end{bmatrix} \begin{bmatrix} \Delta q \\ \Delta\dot{q} \end{bmatrix} + \begin{bmatrix} 0 \\ -M_{|q=0}^{-1} \end{bmatrix} \Delta\tau$$

2.2.1. Sistema lineal variante en el tiempo

Linealizar el sistema alrededor de algún punto de equilibrio no es suficiente debido básicamente a dos razones:

- la linealización solo predice el comportamiento local, no sirve para estudiar el comportamiento lejos del punto de operación;
- la dinámica de un sistema no lineal es mucho más rica que la de un sistema lineal debido a la presencia de fenómenos no lineales como: múltiples PE aislados, ciclos limite, oscilaciones sub-armónicas, armónicas o casi-periódicas, caos, etc.

Más adelante en el aparatado de control y en concreto en el control iterativo con el regulador lineal cuadrático modificado, emplearemos una linealización para distintos puntos, es decir obtendremos matrices A y B variantes con el tiempo. De modo que obtendremos una aproximación más próxima a la dinámica del sistema no lineal.

Para ello, partimos de $\dot{x} = f(x,u)$, que al expandir f(x, u) con una serie de Taylor alrededor de una trayectoria deseada, se obtiene la forma general de un sistema lineal variante en el tiempo (LTV)[2]:

$$\dot{\boldsymbol{x}} = f(x_0,u_0) + A(t)(x - x_0) + B(t)(u - u_0)$$

donde consideraremos los siguientes criterios

$$\bar{x}(t) = x - x_0(t), \qquad \bar{u}(t) = u - u_0(t) \qquad \text{y} \qquad \dot{\bar{x}} = \dot{x} - \dot{x}_0(t) = \dot{x} - f(x_0,u_0)$$

Nótese que estamos haciendo un cambio de coordenadas para que $\dot{x}$ se mueva con la trayectoria, esto es $\dot{\boldsymbol{x}} - \boldsymbol{f(x_0,u_0)}$. De modo que, $x_0(t)$ y $u_0(t)$ tienen que ser solución de $\dot{x} = f(x,u)$. La expresión final que describe la dinámica de un sistema lineal variante con el tiempo (LTV) es:

$$\dot{\bar{x}}(t) = A(t)\bar{x}(t) + B(t)\bar{u}(t) \qquad (2.20)$$

donde las matrices A(t) y B(t) son las matrices Jacobianas del sistema no lineal de la ecuación (2.20) y se definen como se muestra a continuación

$$A(t) = \left.\frac{\partial f}{\partial x}\right|_{(x_0(t),u_0(t))} \qquad B\,(t) = \left.\frac{\partial f}{\partial u}\right|_{(x_0(t),u_0(t))} \qquad (2.21)$$

2.3. Discretización

Como ya vimos la representación de un sistema en espacio de estados permite relacionar las variables de entrada(t) y de salida y(t) mediante una ecuación diferencial matricial de primer orden (Ecuación de Estado) y una combinación lineal de la entrada y los estados x(t) (Ecuación de Salida) de la forma:

$$\dot{x}(t) = Ax(t) + Bu(t) \qquad (2.22)$$

$$y(t) = Cx(t) + Du(t) \qquad (2.23)$$

[2] Lecture 10 | MIT 6.832 Underactuated Robotics, Spring 2009
https://www.youtube.com/watch?v=xwgIkdBQku4

La ecuación de estado (2.22) puede ser resuelta de la siguiente manera:

$$\dot{x}(t) - Ax(t) = Bu(t) \tag{2.24}$$

Si premultiplicamos ambos lados de la ecuación por e^{-AT}

$$e^{-AT}[\dot{x}(t) - Ax(t)] = e^{-AT}Bu(t) = \frac{d}{dt}[e^{-At}x(t)] \tag{2.25}$$

Luego, al integrar entre 0 y t se obtiene:

$$e^{-At}x(t) = x(0) + \int_0^t e^{-A\tau}Bu(\tau)d\tau \tag{2.26}$$

Al multiplicar a la izquierda por e^{AT}

$$x(t) = e^{AT}\left[x(0) + \int_0^t e^{-A\tau}Bu(\tau)d\tau\right] \tag{2.27}$$

Al discretizar la ecuación (2.27) de la forma $t = k + 1$ y $t = k$ se obtiene respectivamente:

$$x((k+1)T) = e^{A(k+1)T}\left[x(0) + \int_0^{(k+1)T} e^{-A\tau}Bu(\tau)d\tau\right] \tag{2.28}$$

$$x(kT) = e^{AkT}\left[x(0) + \int_0^{kT} e^{-A\tau}Bu(\tau)d\tau\right] \tag{2.29}$$

Al multiplicar la ecuación (2.29) por e^{AT} y restarla a la ecuación (2.28) se tiene:

$$x((k+1)T) = e^{A(k+1)T}\left[x(0) + \int_0^{(k+1)T} e^{-A\tau}Bu(\tau)d\tau\right] \tag{2.30}$$

$$e^{AT}x(kT) = e^{AT}e^{AkT}\left[x(0) + \int_0^{kT} e^{-A\tau}Bu(\tau)d\tau\right] \tag{2.31}$$

Para restar (2.29) de (2.28):

$$x((k+1)T) - e^{AT}x(kT) = e^{A(k+1)T}\int_{kT}^{(k+1)T} e^{-A\tau}Bu(\tau)d\tau \tag{2.32}$$

Al despejar $x((k+1)T)$ obtenemos:

$$x((k+1)T) = e^{AT}x(kT) + e^{A(k+1)T}\int_{kT}^{(k+1)T} e^{-A\tau}Bu(\tau)d\tau \tag{2.32}$$

$$x((k+1)T) = e^{AT}\left[x(kT) + \int_{kT}^{(k+1)T} e^{-A(\tau-kT)}Bu(\tau)d\tau\right] \tag{2.32}$$

Para sistemas en tiempo discreto la variable de entrada $u(kT)$ se mantiene constante durante un intervalo de tiempo (intervalo de muestreo T). De forma equivalente para una señal adquirida por un sistema digital, la señal de entrada es muestreada y luego retenida en un valor constante $u(kT)$ durante un intervalo de tiempo T, proceso conocido como retención de datos de orden cero, para estos casos es correcto afirmar que

$$u(t) = u(kT) \quad para\ KT \leq t \leq (k+1)T \tag{2.33}$$

Además con el cambio de variable $t = \tau - kT$

$$x((k+1)T) = e^{AT}\left[x(kT) + \int_{kT}^{(k+1)T} e^{-At}Bu(kT)d\tau\right] \tag{2.34}$$

Haciendo $\lambda = T - t$ y sabiendo que la entrada es constante para un intervalo de muestreo T, entonces

$$x((k+1)T) = e^{AT}x(kT) + \left[\int_0^T e^{A\lambda}d\tau\right]Bu(kT) \tag{2.35}$$

Esta es la Ecuación de Estado en tiempo discreto, en forma de ecuación de diferencias matricial de la forma:

$$x(k+1) = \bar{A}(k)x(k) + \bar{B}(k)u(k) \tag{2.36}$$

Observando tenemos la equivalencia:

$$\bar{A}(T) = e^{AT} \tag{2.37}$$

$$\bar{B}(T) = \left(\int_0^T e^{A\lambda}d\tau\right)B \tag{2.37}$$

La discretización de la ecuación de salida se obtiene al reemplazar en la ecuación (2.23) $t = kT$ estableciendo la ecuación de salida en tiempo discreto

$$y(k) = C(k)x(k) + D(k)u(k) \tag{2.39}$$

La Serie de Taylor para e^{AT} es:

$$e^{AT} = I + AT + \frac{1}{2!}(AT)^2 + \cdots\frac{1}{n!}(AT)^n + \sum_{n=0}^{\infty}\frac{1}{n!}(AT)^n \tag{2.40}$$

Teniendo en cuenta que T es el periodo de muestreo y que es un valor muy pequeño, generalmente una centésima o milésima de segundo, es válido afirmar que T^2 es mucho más pequeño, y que T^2 es menor que los anteriores.

Con base en esto la expresión e AT se puede aproximar numéricamente a:

$$\bar{A} \approx I + AT \tag{2.41}$$

De igual forma se determina el valor de $\bar{B}$:

$$\bar{B} = \left(\int_0^T e^{A\lambda} d\tau\right) B = (A^{-1}\left[e^{A\lambda}\right]_0^T)B \tag{2.42}$$

$$\bar{B} = (A^{-1}[e^{AT} - I])B = A^{-1}(\bar{A} - I)B \tag{2.43}$$

$$\bar{B} \approx (A^{-1}AT)B \approx -BT \tag{2.44}$$

Se puede seguir un enfoque alternativo más aproximado para calcular estas matrices, evitando la integral involucrada en B de (2.37). Considere un vector de "estado" aumentado/siguiente, incluyendo la ecuación $u = 0$.

$$x_e = \begin{pmatrix} x \\ u \end{pmatrix}; \qquad \dot{x}_e = \begin{pmatrix} A & B \\ 0 & I \end{pmatrix} \begin{pmatrix} x \\ u \end{pmatrix} = A_e x_e \tag{2.45}$$

la respuesta entre muestras puede interpretarse como una "respuesta libre" del sistema aumentado. A partir de (2.45), para $KT \leq t \leq (k+1)T$:

$$x_e(t) = \begin{pmatrix} x(t) \\ u(t) \end{pmatrix} = e^{A_e(t-kT)} x_e(kT) \tag{2.46}$$

Así pues, dividiendo la matriz exponencial, las matrices correspondientes a (2.37) y (2.38) son:

$$\begin{pmatrix} \bar{A} & \bar{B} \\ 0 & I \end{pmatrix} = e^{A_e T} \tag{2.47}$$

Ejemplo de implementación en Matlab de (2.47)

```
n = length(A); nb = length(B)
s = expm([[A B]*dt; zeros(nb,n+nb)]);
Ad = s(1:n,1:n); Bd = s(1:n,n+1:n+nb);
```

Código 1. Discretización: método alternativo.

Por otro lado, el equivalente discreto de la ecuación (2.20) se obtiene discretizando el sistema LTV, esto puede escribirse como:

$$x_{k+1} = A_k x_k + B_k u_k \tag{2.22}$$

donde x_k es el vector de estados evaluado en $t = k\Delta Ts$, u_k denota el vector de entradas de control evaluado en $t = k\Delta Ts$, x_{k+1} representa el vector de estados evaluado en $t = (k+1)\Delta Ts$, A_k y B_k son las versiones discretas de las matrices descriptas en ec. (2.20). Entonces lo que habría que hacer es para cada A_k y B_k obtener su equivalente discreto siguiendo el método que se acaba de describir.

En Matlab, existe una función dedicada a esa tarea, con lo cual podríamos emplear directamente esa función. Suponiendo que el periodo de muestro es inferior 1ms, con (2.41) y (2.44) la aproximación es bastante buena pudiendo emplear la forma simplificada. Sin embargo, durante las simulaciones se ha observado que causa muchas problemas en Matlab, sobre todo en la parte del iLQR donde se linealiza para distintos puntos; y en concreto cuando se realizan las inversas de algunas operaciones con esas matrices discretas, debido a la presencia de ceros aparecen advertencias. De modo que, el método descrito en 2.47 resulta ser el mejor y es el que se ha empleado.

Para nuestro caso, las matrices A y B anteriormente vistas, dan lugar a las dos nuevas matrices discretas:

$$A_d = \begin{bmatrix} 0.7961 & 0.3246 & 0.2049 & -0.3249 & 0.0092 & 0.0016 & 0.0008 & -0.0016 \\ 0.4034 & -0.3725 & -0.4043 & 1.3742 & 0.0018 & 0.0035 & -0.0018 & 0.0066 \\ 0.3364 & 0.0337 & 0.6636 & -0.0338 & 0.0012 & 0.0001 & 0.0088 & -0.0001 \\ 0.1129 & 0.5539 & -0.1132 & 0.4465 & 0.0003 & 0.0026 & -0.0003 & 0.0074 \\ -26.0581 & 4.0978 & 26.2347 & -4.1107 & 0.7961 & 0.3246 & 0.2049 & -0.3249 \\ 29.4853 & -31.1940 & -29.5567 & 31.3740 & 0.4034 & -0.3725 & -0.4043 & 1.3742 \\ 60.2503 & 11.2594 & -60.2239 & -11.2682 & 0.3364 & 0.0337 & 0.6636 & -0.0338 \\ 36.6336 & 20.9949 & -36.7142 & -20.8482 & 0.1129 & 0.5539 & -0.1132 & 0.4465 \end{bmatrix}$$

$$B_d = \begin{bmatrix} 0.0000 & -0.0000 \\ -0.0000 & 0.0000 \\ 0.0000 & -0.0000 \\ -0.0000 & 0.0001 \\ 0.0006 & -0.0039 \\ -0.0013 & 0.0158 \\ 0.0066 & -0.0002 \\ -0.0002 & 0.0179 \end{bmatrix}$$

con periodo de muestreo de 10 ms.

CAPÍTULO 3: CONTROL PROPORCIONAL DERIVATIVO

El control proporcional derivativo es un mecanismo de control que se basa en el concepto fundamental de realimentación, que consiste en el proceso de medir las variables de interés en el sistema y usar esa información para controlar su comportamiento.

Un aspecto importante de la presencia de elasticidad en las articulaciones es que la parte de realimentación de la ley de control puede depender en general de cuatro variables para cada articulación: la posición del motor y del eslabón, y la velocidad del motor y del eslabón. De modo que se necesitan cuatro sensores para las mediciones: dos sensores de posición (por ejemplo, un encoder) uno antes y otro después de los efectos de elasticidad. En algunos casos, se emplean tacómetro como sensor de velocidad. Cuando no hay sensores de velocidad, la velocidad se reconstruye típicamente mediante una diferenciación numérica adecuada de mediciones de posición a alta resolución.

3.1. Control PD

PD hace referencia a la combinación de las acciones de control proporcional y derivativo cuyo comportamiento presenta tanto las ventajas como las desventajas de cada acción de control como si fueran implementadas de manera individual. En el tiempo continuo (dominio de la frecuencia) la función de transferencia de las dos acciones aplicadas en paralelo se representa como se ve a continuación, donde K_d es la constante de control derivativa, K_p la proporcional.

$$c(s) = K_p + K_d \cdot s$$

Por lo general, una gran pendiente en $e(t)$ en un sistema lineal correspondiente a una entrada escalón considerable produce un gran sobreimpulso en la variable controlada. El control derivativo mide la pendiente instantánea de $e(t)$, prediciendo que tan grande será el sobreimpulso aplicando las correcciones apropiadas antes de que se presente ese sobreimpulso.

A la hora de realizar el control es lógico pensar que, si se desea controlar la posición y/o velocidad del eslabón, se debe realimentar justo la posición y velocidad del lado del eslabón.

$$\tau = Kp_e(q_d - q) - Kd_e\dot{q}$$

donde Kp_e y Kd_e son las ganancias de la realimentación y q_d es la referencia.

Sin embargo, esta configuración resulta inestable y sobre todo cuando los cambios son bruscos. Se puede verificar que los polos en bucle cerrado serán inestables, independientemente de cómo se elijan las ganancias. De manera similar, la combinación de la posición del motor y la retroalimentación de la velocidad del eslabón también resulta inestable. Por lo tanto, descartaremos estas dos opciones.

Otra estrategia de retroalimentación mixta sería usar la posición del eslabón y la velocidad del motor.

$$\tau = Kp_e(q_d - q) - Kd_m\dot{\theta}$$

Esta configuración resulta ser estable y se puede demostrar utilizando el criterio de Routh, que la estabilidad asintótica se produce sí y sólo sí, la ganancia de velocidad del motor $Kd_m > 0$ y la ganancia de la posición del eslabón satisface $0 < Kp_e < K$. La realimentación de la acción proporcional no debe anular la rigidez del muelle. Si lo que se desea es alcanzar una posición determinada con mayor exactitud esta sería la configuración recomendable. Sin embargo, las limitaciones anteriormente citadas a veces podrían limitan el uso de esta configuración.

Finalmente llegamos a la última posible configuración y es realimentar la posición y la velocidad del lado del motor

$$\tau = Kp_m(\theta_d - \theta) - Kd_m\dot{\theta} \quad considerando \;\; \theta_d \approx q_d$$

esta configuración en bucle cerrado es siempre estable, con la condición de que tanto Kp_m como Kd_m sean estrictamente positivos (y de otra manera arbitrariamente grandes). Resulta que esta es la configuración preferible para robots con numerosos eslabones y es la que emplearemos en este proyecto.

La dificultad de implementar el controlador PD se encuentra en la estructuración de un algoritmo que permita la realización de un control que pueda manejar múltiples entradas y salidas, adicional a esto es necesario determinar las constantes de control que sean las más apropiadas para la dinámica del proceso.

3.1.1. Simulaciones

Para realizar las simulaciones tanto del controlador PD como del controlador PD con compensación de gravedad emplearemos Simulink que funciona sobre el entorno de programación Matlab.

Figura 4. Logotipo de Matlab.

Trabajar en un entorno de programación visual siempre resulta más fácil, rápido y cómodo. Con Simulink se podría simular y probar el controlador directamente sobre el sistema no lineal. Evidentemente primero se tendría que crear, este programa permite crear subsistemas con lo cual podríamos ir creando los distintos bloques por separado hasta construir el modelo no lineal final, o directamente empleando un bloque de función de Matlab donde se tendrán en cuenta todas las ecuaciones diferenciales.

A continuación, se muestra la implementación del controlador PD con todos los bloques necesarios. Nótese que estamos ante un control MIMO de modo que tanto en la entrada como en la salida empleamos multiplexores. Para el control de un brazo robot con dos articulaciones se necesitan dos motores por lo tanto dos variables de entrada a controlar. Se han implementado los dos PD por separado donde cada regulador tiene sus ganancias propias.

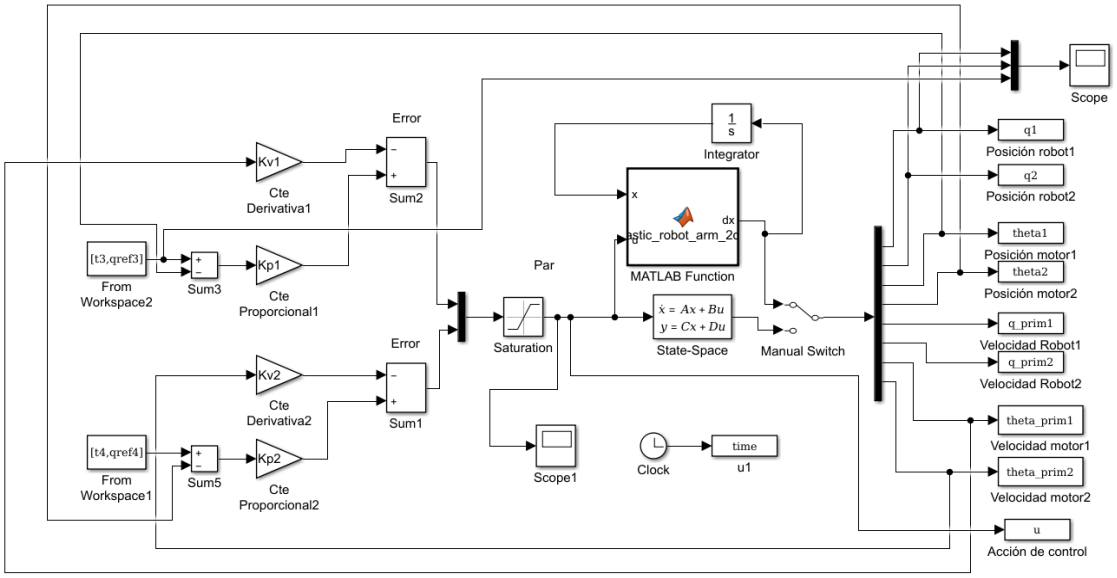


Figura 5. Modelo en simulink con el controlador PD.

En cuanto a los resultados obtenidos en la siguiente grafica se representa la respuesta ante escalón de la posición de cada eslabón empleando el modelo lineal.

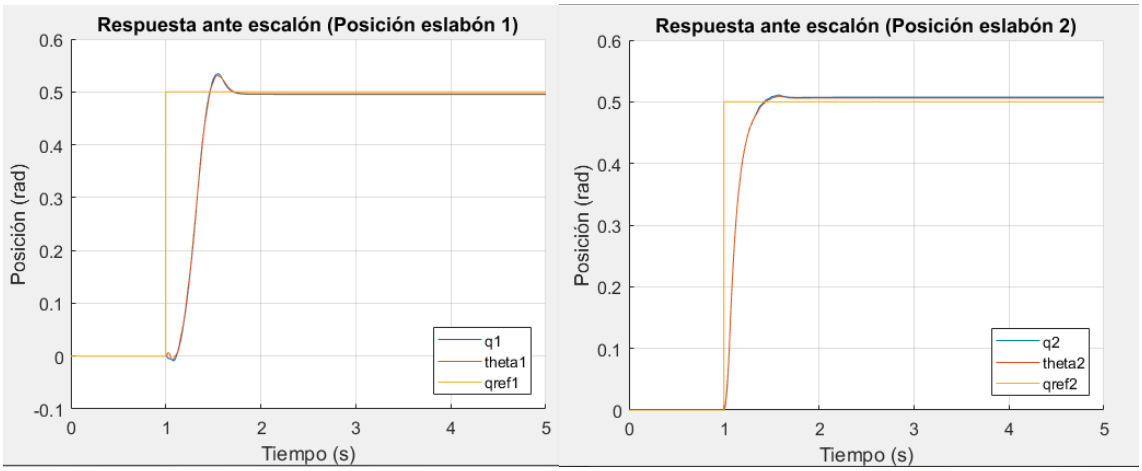


Figura 6. Respuesta ante escalón de la posición con el regulador PD.

El tiempo de establecimiento es rápido y el error en estado estacionario casi nulo, con lo cual podemos afirmar que el regulador PD está haciendo bien su trabajo. Tanto la posición angular del lado del motor como del lado del eslabón son prácticamente iguales.

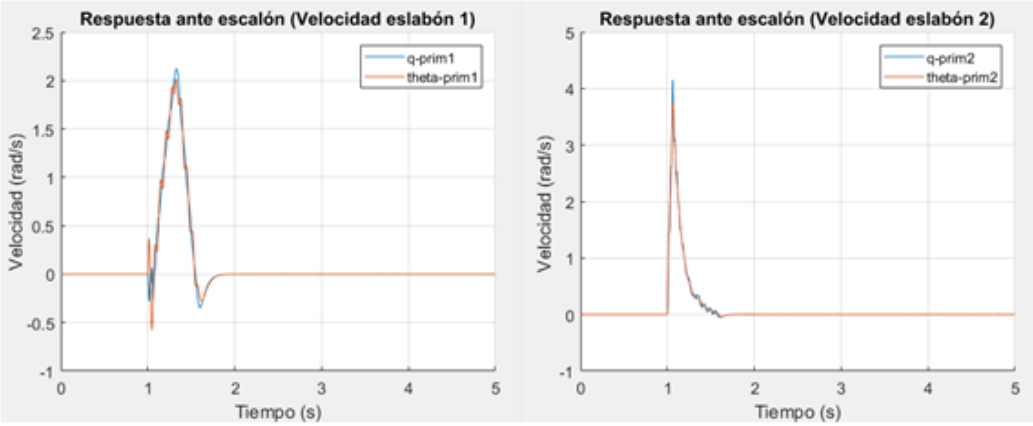


Figura 7. Respuesta ante escalón de la velocidad con el regulador PD.

Evidentemente si queremos que la respuesta sea rápida la variación de la posición es grande y es justo lo que se refleja en la figura anterior.

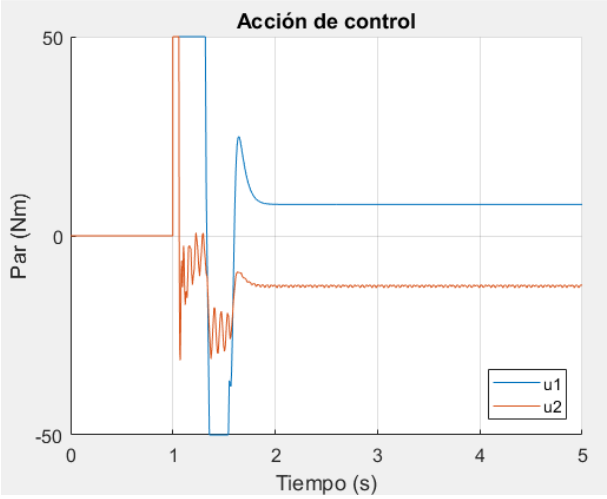


Figura 8. Acción de control

En cuanto a la acción de control vemos que el par generado o necesario es bastante alto con lo cual hay varios instantes en los que el par es máximo, en el transitorio se estabiliza. Con todas las gráficas analizada parece que el controlador funciona bastante bien, aunque recordar que era lo esperado ya que estamos realimentando del lado del motor y la estabilidad estaba garantizada si las ganancias se escogen bien.

Hemos visto la respuesta del PD ante escalón, sin embargo, en las aplicaciones de robótica los movimientos tienen que ser más suaves, aunque la respuesta sea más lenta. De modo que se emplean generadores de trayectoria que básicamente generar todos los puntos de la trayectoria para los distintos instantes de tiempo, ya sea para la posición, velocidad o alguna otra derivada superior, dependiendo del orden de la trayectoria empleada.

Para el control PD emplearemos una trapezoidal de orden 2 o de orden 4, en el apartado de Anexos se describen algunas de estas trayectorias.

Para analizar mejor la eficacia del controlador PD y luego comparar los resultados con el controlador PD con compensación de gravedad se ha optado por simular directamente con el modelo no lineal donde se tiene en cuenta la evolución de la gravedad según la posición de las articulaciones. Para ello se ha creado un bloque de función de Matlab[3] donde se han introducido las 8 ecuaciones diferenciales de los 8 estados.

Además, se tienen en cuenta las siguientes condiciones iniciales, la posición inicial del primer eslabón es de -90 grados, esto equivale a cuando el robot está en la posición inferior.

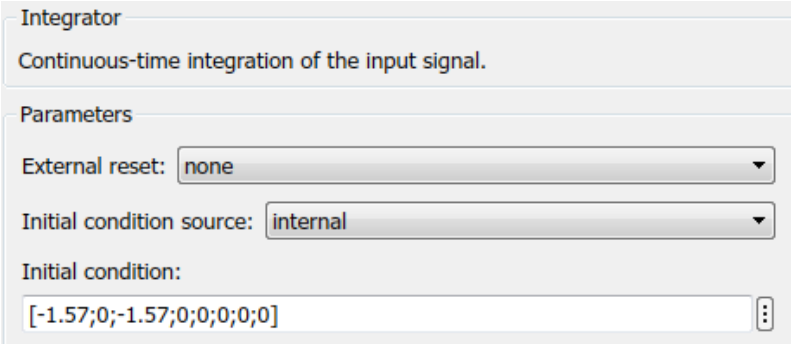


Figura 9. Estados iniciales

A continuación, se facilitan los resultados obtenidos para el seguimiento de posición de la trayectoria trapezoidal de orden 4. Las ganancias empleadas son **Kp1 = 2500, Kp2 = 2500 y Kd1 = Kd2 = 50.**

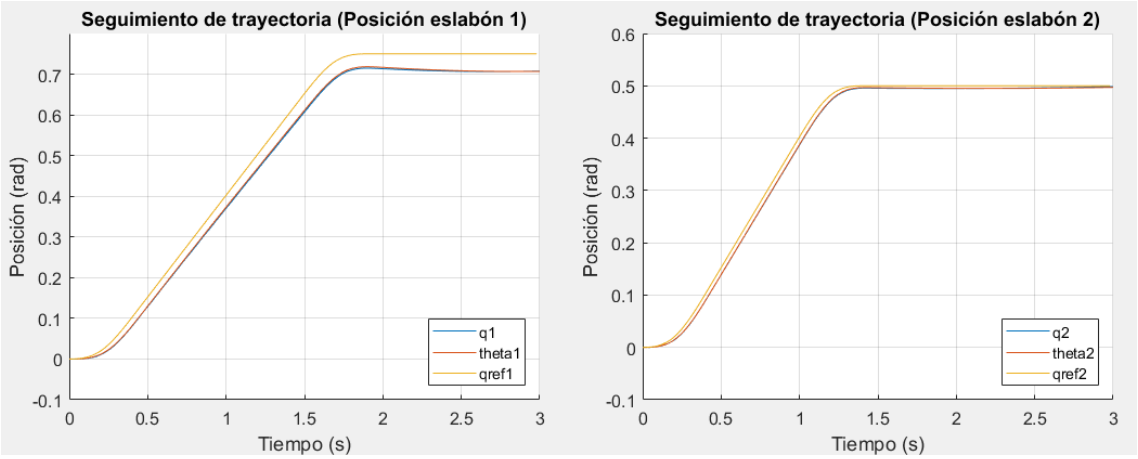


Figura 10. Resultado del seguimiento de la trayectoria de la posición con el PD.

El seguimiento de la trayectoria no es perfecto, sin embargo, es bastante bueno. El error en estado estacionario del primer elemento es apreciable a simple vista, además, se aprecia en el **error cuadrático medio** que es de **0.3017rad**. En cuanto al segundo eslabón el seguimiento es bastante mejor y el error cuadrático medio es de tan solo **0.1567radianes**. Se podría mejorar aumentando las ganancias, pero por ahora, consideremos este ejemplo para comparar con el PD con compensación de gravedad.

[3] Ver el fichero de Simulink adjunto con la memoria.

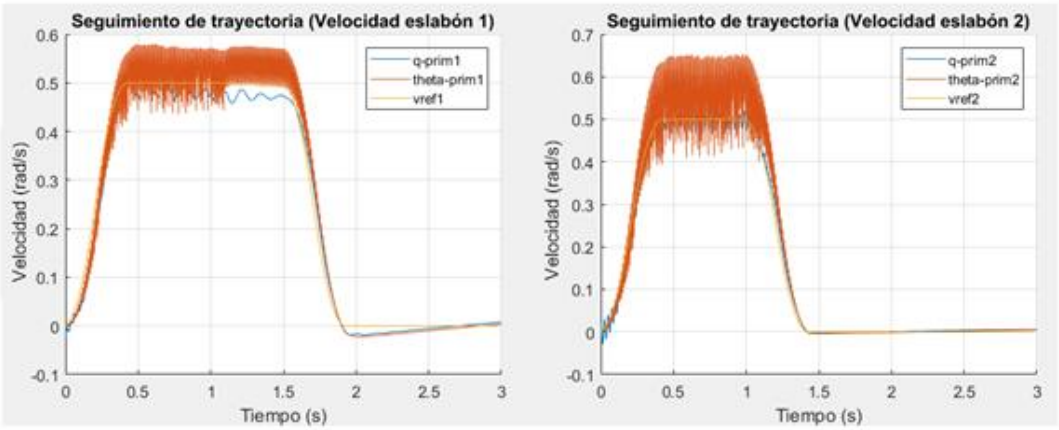


Figura 11. Resultado del seguimiento de la trayectoria de la velocidad con el PD.

Por otra parte, tenemos el seguimiento de la velocidad, aunque realmente no podemos hablar de seguimiento ya que la velocidad de referencia generada con el generador de trayectorias no se incluye, es decir se realimenta solamente la velocidad obtenida del modelo. Pero aun así vemos, que al tener la posición de referencia es esperado que tenga un comportamiento muy parecido a la generada. A partir de las gráficas (figura 11) se puede decir que el seguimiento no es tan malo, aunque con presencia de muchas oscilaciones. Esto podría ser debido al acoplo de las articulaciones, así como debido a la presencia de elasticidad. En cuanto al **error cuadrático medio de velocidad**, para el primer elemento es de **0.1517 rad/s** y de **0.1377 rad/s** para el segundo.

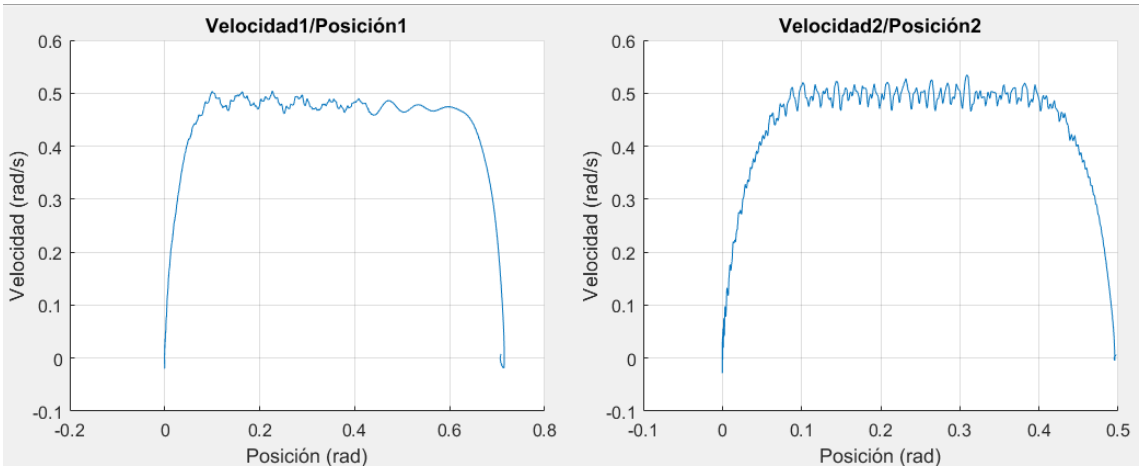


Figura 12. Velocidad frente a posición

En la figura 12 se ve cómo evoluciona la posición con la velocidad, normalmente según la trayectoria descrita se pueden sacar muchas conclusiones de cómo es la respuesta, de si es suave o hay cambios bruscos etc. En este caso se podría deducir un comportamiento un poco oscilante.

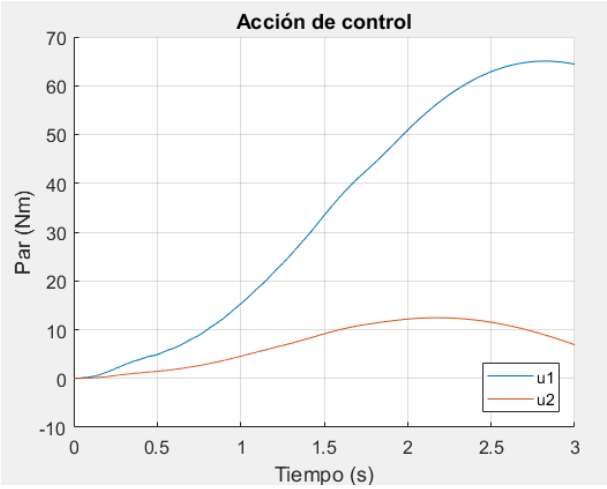


Figura 13. Acción de control

Finalmente tenemos la acción de control, que parece que no llega a estabilizarse en ningún valor concreto para ambos casos. Analizando todas las gráficas, en general podemos afirmar que el control realizado cumple con las expectativas y con un ajuste mejor de las ganancias se podrían conseguir muy buenos resultados. Una mejora seria añadir compensación de gravedad que veremos en el siguiente apartado u otra solución sería realimentar más estados, algo que ya incorpora el LQR que veremos en el apartado de control óptimo.

3.2. Control PD con compensación de gravedad online

La presencia de gravedad requiere la adición de alguna forma de compensación de gravedad a la acción de PD. La modificación más simple para manejar la presencia de gravedad es considerar la adición de un término constante que compense exactamente la carga de gravedad en el estado estacionario, sin embargo, no es la mejor solución, es recomendable considerar un término variable que se corresponde con la gravedad obtenida del modelo dinámico y que depende de la posición angular de las articulaciones. Un control PD con compensación de gravedad *online* se puede expresar de la siguiente manera.

$$\tau = Kp_m(\theta_d - \theta) - Kd_m\dot{\theta} + g(q_d)$$

con $Kp_m > 0$, $Kd_m > 0$, matrices simétricas (típicamente diagonales). El uso de esta configuración con compensación *online* de la gravedad, suele proporcionar un transcurso más suave y una notable reducción de los errores de posición en el transitorios, sin ningún esfuerzo de control adicional en términos de pares máximos y medios.

3.2.1. Simulaciones

De forma similar partiendo del controlador PD básico obtenido en el apartado 3.1.1 lo extendemos un poco más añadiendo los efectos de gravedad y de esta manera obtenemos el controlador PD con compensación de gravedad para el brazo robot de 2 grados de libertad. Obsérvese que se incluyen dos subsistemas donde se realizan una serie de operaciones que estiman la fuerza debida a la gravedad y se suma a la acción de control.

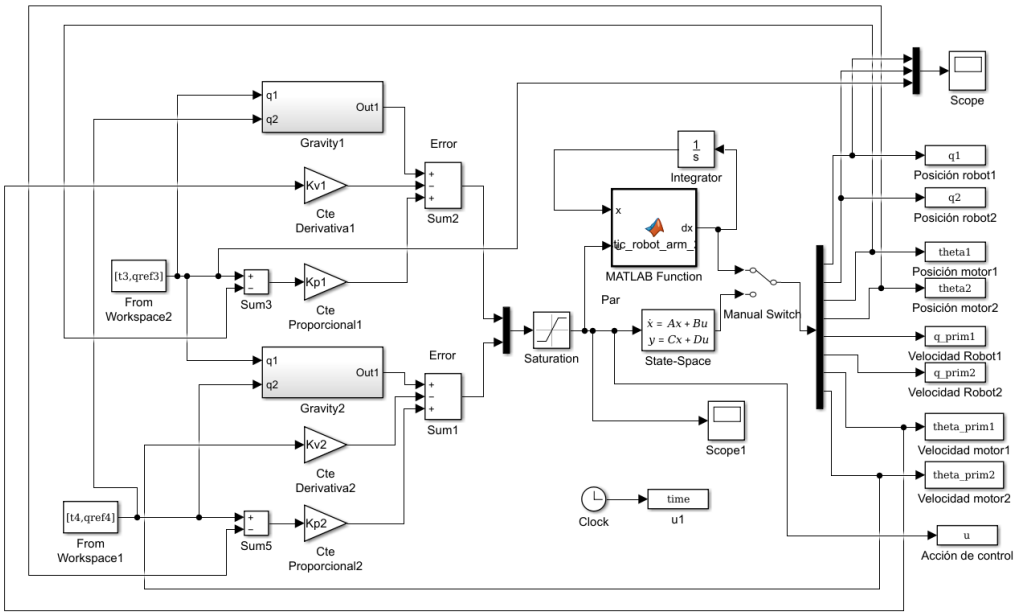


Figura 14. Control PD con compensación de gravedad.

Procederemos a simularemos directamente para el caso de seguimiento de trayectoria saltando el paso de simulación de respuesta ante escalón que hicimos en el caso del PD sin compensación de gravedad. Cabe destacar que este controlador se aplicará directamente sobre el modelo no lineal al igual que en el PD normal. Que como vimos, de esta manera se tiene en cuenta la evolución de la gravedad en función de la posición angular de las articulaciones. Así pues, simulando para las mismas ganancias: **Kp1 = 2500, Kp2 = 2500 y Kd1 = Kd2 = 50**

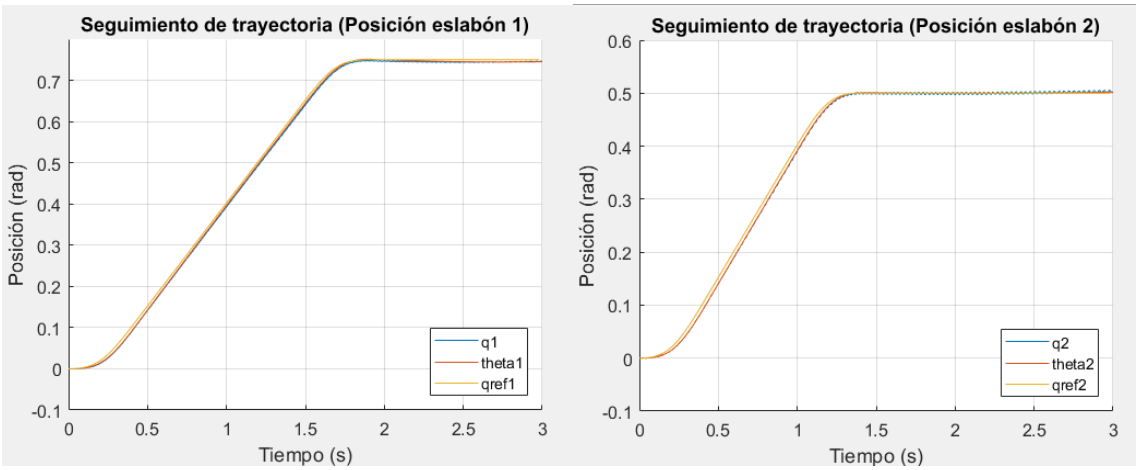


Figura 15. Resultado del seguimiento de la trayectoria de la posición con el controlador PD más compensación de gravedad.

A partir de la figura 15, podemos decir que el seguimiento de la posición tanto del eslabón 1 como del 2 es considerablemente mejor. No se aprecia error en estado estacionario a simple vista. El **error cuadrático medio de posición** es **0.0712**y **0.1132radianes** para el eslabón 1 y el eslabón 2 respectivamente.

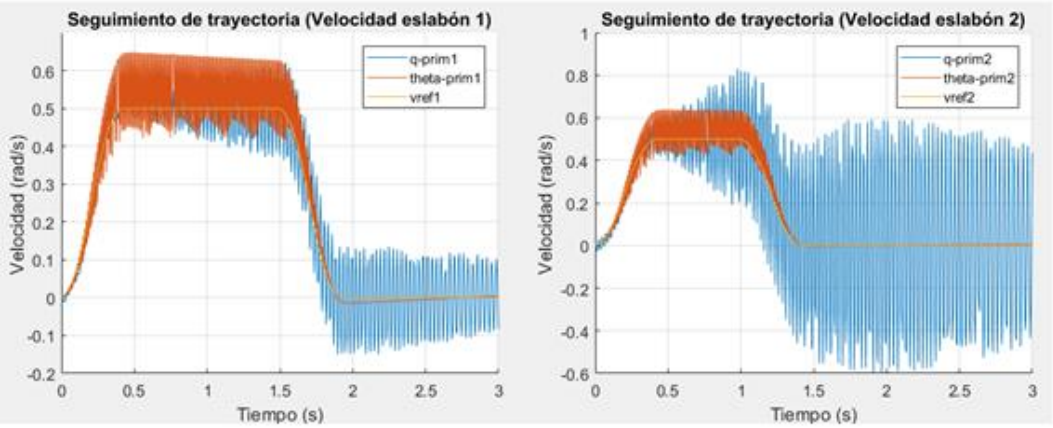


Figura 16. Resultado del seguimiento de la trayectoria de la velocidad con el controlador PD más compensación de gravedad.

A partir de la figura 16 y en comparación con la 11, se observa que se experimenta un incremento de las oscilaciones sobre todo en los tramos donde la velocidad es constante. Esto es un comportamiento un poco diferente al esperado, ya que, sabemos que en robots rígidos, con la compensación de gravedad las oscilaciones suelen desaparecer en la mayor medida. Si miramos el **error cuadrático medio de velocidad** para el primer elemento es de **0.0260rad/s** frente a los **0.1517 rad/s** del caso anterior y para el segundo eslabón **0.1402rad/s** frente a los **0.1377 rad/s** del caso anterior.

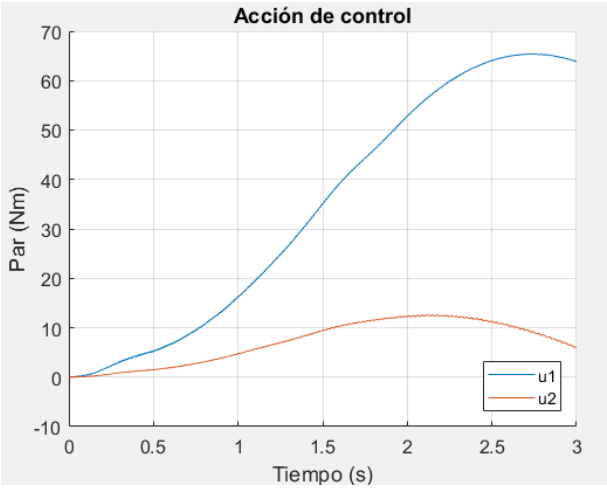


Figura 17. Acción de control.

Finalmente, en la figura 17 tenemos la acción de control, a simple vista se observa que es muy similar a la obtenida en el caso anterior sin compensación de gravedad, sin embargo, si miramos los valores, veremos que en cada instante se ha sumado un valor a la acción de control del PD, correspondiente a la compensación de la gravedad.

En conclusión, podemos decir que compensando la gravedad se ha conseguido un seguimiento de trayectoria de la posición mejor y se puede considerar una técnica buena y de sencilla implementación para controlar brazos robóticos, aun con presencia de elasticidad.

CAPÍTULO 4: CONTROL ÓPTIMO

La palabra óptimo intuitivamente significa hacer un trabajo de la mejor manera posible. Antes de comenzar una búsqueda de tal solución óptima, el trabajo debe ser definido, una escala matemática debe ser establecida para cuantificar lo que significa mejor. A menos que los cualificadores sean claros y consistentes, declarar que un sistema es óptimo no tiene sentido real. Un sistema crudo e impreciso puede ser considerado óptimo porque es barato, es fácil de fabricar y da un rendimiento adecuado. Por el contrario, un sistema muy preciso y elegante podría ser rechazado como no óptimo porque es demasiado caro o es pesado o llevaría demasiado tiempo para desarrollarse.

Para entender la idea de criterio de optimización en variable de estados, hablemos de sistemas de tiempo discreto, que son más simples. El estado de un sistema discreto describe una trayectoria haciendo transiciones discretas de un estado a otro bajo el efecto de una entrada también aplicada en tiempo discreto.

Cuando se asocia un criterio de optimización al sistema, cada transición de estado tiene asociado un costo o penalidad. Por ejemplo, pueden penalizarse las transiciones de estado que se alejan demasiado del estado final deseado, o las acciones de control de valores demasiado elevados. A medida que el sistema evoluciona de estado en estado, los costos se suman hasta acumular un costo total asociado a la trayectoria. Ilustramos el concepto con el grafo de la Figura 18, que representa 8 estados de un sistema discreto con sus transiciones posibles. El estado inicial es el 1, y el final el 8. El sistema pasa de un estado a otro en cada tiempo k determinado por la entrada u_k y las ecuaciones $x_{k+1} = Ax_k + Bu_k$.

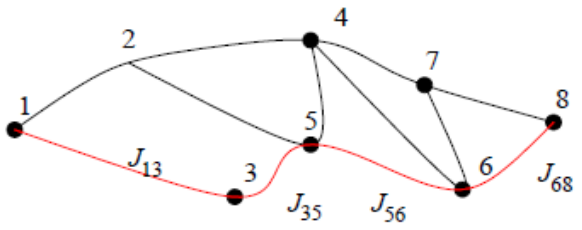


Figura 18. Posibles trayectorias de 1 al 8.

Las transiciones posibles se representan por los arcos que conectan el estado inicial al final a través de los estados intermedios. El costo asociado a cada transición se representa con la letra J; por ejemplo, el costo de moverse del estado 3 al 5 es J_{35}.

Asumiendo que los costos se acumulan en forma aditiva, vemos que la trayectoria marcada en rojo, por ejemplo, tiene un costo total $J_{13} + J_{35} + J_{56} + J_{68}$.

Como hay varias rutas alternativas del estado 1 al 8, el costo total dependerá de la trayectoria elegida. La señal de control u_k^* que determina la trayectoria de menor costo es la estrategia óptima. En sistemas de tiempo continuo, la acumulación de costos se representa mediante integración, en vez de suma.

4.1. Regulador Linear Cuadrático (LQR)

Se dice que es una estrategia de control óptima ya que se procura la operación del sistema dinámico al menor costo, es decir que los ajustes del controlador proporcional se encuentran usando un algoritmo matemático que minimiza la función de coste o desviaciones no deseadas.

Este tipo de control ha sido concebido para manejar procesos multivariables como lo son los SIMO (una entrada, múltiples salidas) y los MIMO (múltiples entradas, múltiples salidas) los cuales se caracterizan por presentar "matrices de transferencia" en lugar de "funciones de transferencia".

4.1.1. Control LQ discreto

Cuando se desea implementar un controlador en un microcontrolador y en general el diseño de controles digitales, implica el uso de sistemas en la forma discreta. Entonces para este caso, consideremos el sistema en tiempo discreto definido por

$$x_{k+1} = Ax_k + Bu_k \tag{4.1}$$

El problema de control que queremos resolver es: encontrar la secuencia de control u_k que lleve al sistema (4.1) de la condición inicial $x_i = x_0$ al estado final $x_N = x_F$, minimizando el funcional cuadrático

$$J_{i,N} = \frac{1}{2}x_N^T Q_f x_N + \frac{1}{2}\sum_{k=i}^{N-1} x_k^T Q x_k + u_k^T R u_k \tag{4.2}$$

El funcional (4.2) puede interpretarse como el costo total de la transición de x_i a x_N y, en particular, el término $x_N^T Q_f x_N$ penaliza el error en alcanzar el estado final deseado.

Las matrices de peso Q_f, Q y R pueden seleccionarse para penalizar ciertos estados/entradas más que otros. Como veremos, las matrices Q_f y Q deben ser semi-definidas positivas, y la R definida positiva.

Supongamos que estamos en el paso $N-1$ de la trayectoria óptima. Así, el costo (4.2) de la transición de $N-1$ a N es

$$J_{N-1,N} = \frac{1}{2}\left[x_N^T Q_f x_N + x_{N-1}^T Q x_{N-1} + u_{N-1}^T R u_{N-1}\right] \tag{4.3}$$

Usando (4.1), substituimos x_N como una función de u_{N-1}, lo que da

$$J_{N-1,N} = \frac{1}{2}\left[(Ax_{N-1} + Bu_{N-1})^T Q_f (Ax_{N-1} + Bu_{N-1}) + x_{N-1}^T Q x_{N-1} + u_{N-1}^T R u_{N-1}\right] \tag{4.4}$$

Como J es cuadrático en u, podemos minimizarlo diferenciando

$$\begin{aligned} 0 = \frac{\partial^T J_{N-1,N}}{\partial u_{N-1}} &= B^T Q_f (Ax_{N-1} + Bu_{N-1}) + Ru_{N-1} \\ &= \left(R + B^T Q_f B\right)u_{N-1} + B^T Q_f A x_{N-1} \end{aligned} \tag{4.5}$$

De (4.5) obtenemos el último elemento de la secuencia de control óptima

$$u_{N-1}^{*} = -(R + B^T Q_f B)^{-1} B^T Q_f A x_{N-1} \tag{4.6}$$

que resulta ser un mínimo ya que si realizamos la segunda derivada el resultado es mayor a 0.

Como vemos, (4.6) es un control lineal por realimentación de estados de la forma habitual

$$u_{N-1}^{*} = -K_{N-1} x_{N-1} \tag{4.7}$$

donde

$$K_{N-1} = (R + B^T S B)^{-1} B^T Q_f A \tag{4.8}$$

El valor del costo mínimo $J_{N-1,N}$ obtenido con u_{N-1}^{*} es

$$\begin{aligned} J_{N-1,N} = \frac{1}{2}\big[&(Ax_{N-1} - BK_{N-1}x_{N-1})Q_f(Ax_{N-1} - BK_{N-1}x_{N-1}) + x_{N-1}^T Q x_{N-1} \\ &+ K_{N-1}x_{N-1} R K_{N-1} x_{N-1}\big] \\ &= \frac{1}{2} x_{N-1}^T \big[(A - BK_{N-1})Q_f(A - BK_{N-1}) + Q \\ &+ K_{N-1}^T R K_{N-1}\big] x_{N-1} = \frac{1}{2} x_{N-1}^T S_{N-1} x_{N-1} \end{aligned} \tag{4.9}$$

donde

$$S_{N-1} = (A - BK_{N-1})Q_f(A - BK_{N-1}) + Q + K_{N-1}^T R K_{N-1}$$

$$S_N = Q_f$$

$$K_N = (R + B^T S_N B)^{-1} B^T S_N A$$

Tomamos otro paso hacia atrás en el cómputo del control óptimo y consideremos ahora que estamos en el paso $N-2$. Notemos primero de (4.2) que

$$J_{N-2,N} = J_{N-2,N-1} + J_{N-1,N}$$

Por lo tanto, para calcular la estrategia óptima para ir del estado $N-2$ al estado final N, usamos el principio de optimalidad deduciendo que

$$J_{N-2,N}^{*} = J_{N-2,N-1} + J_{N-1,N}^{*}$$

y así reducimos el problema al de calcular la estrategia óptima para ir de $N-2$ a $N-1$ (la de $N-1$ a N ya la obtuvimos). Así, ahora el paso $N-1$ es el "final", y podemos usar la misma ecuación (4.3) pero con $N-1$ en vez de N, y $N-2$ en vez de $N-1$.

$$J_{N-2,N-1} = \frac{1}{2}\left[x_{N-1}^T Q_f x_{N-1} + x_{N-2}^T Q x_{N-2} + u_{N-2}^T R u_{N-2}\right]$$

Igual que antes, podemos minimizar $J_{N-2,N-1}$ sobre todos los posibles u_{N-2}, y es fácil ver que las expresiones son las mismas pero con $N-1$ en vez de N, y $N-2$ en vez de $N-1$,

$$u_{N-2}^{*} = -K_{N-2} x_{N-2}$$

donde

$$K_{N-2} = (R + B^T S_{N-1} B)^{-1} B^T S_{N-1} A$$

Generalizado para cualquier transición y sabiendo que S_k se resuelve de forma recursiva para atrás, comenzando en $S_N = Q_f$. Podemos representar el controlador LQ completo.

$$u_k^* = -K_k x_k \tag{4.10}$$

$$K_k = (R + B^T S_{k+1} B)^{-1} B^T S_{k+1} \tag{4.11}$$

$$S_k = (A - BK_k) S_{k+1} (A - BK_k) + Q + K_k^T R K_k \tag{4.12}$$

Con estas ecuaciones ya podemos obtener las ganancias óptimas en cada instante. En el apartado 4.1.4 se muestra la implementación en Matlab y en el apartado 4.1.5. se muestran simulaciones utilizando este algoritmo de control. En el apartado 4.1.3. se formula un LQR modificado para seguir una trayectoria y más adelante el mismo problema, pero teniendo en cuenta la dinámica variante.

4.1.2. Controlabilidad

Un sistema controlable es aquel que converge a una referencia en un tiempo finito, es decir que sin importar cuál sea su estado actual en algún momento la respuesta se estabilizará en un punto definido. Si se aplica una acción de control cualquiera, esta tendrá la capacidad de llevar los estados del proceso a un valor de referencia en un lapso de tiempo finito, se sabe que los estados son afectados por las entradas y por lo tanto el control se efectúa manipulando las entradas.

Entonces si un proceso es controlable debe existir una función de entrada o control capaz de lograr que los estados del sistema cambien de un valor a otro en un tiempo determinado, hay un acople entre las entradas y estados todos los estados deben ser afectados por las entradas.

Para comprobar la controlabilidad se puede establecer que el rango de la matriz de controlabilidad coincida con el orden del sistema como se muestra en (38) donde A y B son las matrices de estado y n es el orden; con esto se garantiza que en el comportamiento dinámico del sistema no exista un punto en el que se torne inestable.

$$C = [A|AB|A^2B| \dots |A^{n-1}B] \tag{4.13}$$

Agregar términos adicionales a la matriz de controlabilidad con potencias superiores de la matriz A no aumentará el rango de la matriz de controlabilidad ya que estos términos adicionales serán combinaciones lineales de los términos anteriores.

En Matlab comprobar si un sistema es controlable es muy sencillo, simplemente podemos emplear el comando ctrb() que calculará la matriz de controlabilidad del sistema en espacio de estados LTI, entonces comparando si coincide con el orden del sistema podríamos afirmar si lo es o no lo es.

4.1.3. Seguimiento de trayectoria

En esta sección queremos construir un esquema de control que haga que el sistema siga (o rastree) una trayectoria deseada durante todo el intervalo de tiempo usando una ley de control de lazo cerrado. Tales estrategias de control son importantes, por ejemplo, en el control de vehículos espaciales y de robots. En este apartado nos ocuparemos de los sistemas invariantes en el tiempo, en el apartado 4.2 veremos cómo se hace para sistemas variantes con el tiempo (LTV).

Considerando la dinámica de la planta

$$x_{k+1} = Ax_k + Bu_k \tag{4.14}$$

queremos seguir una señal de referencia conocida r_k durante un intervalo de tiempo [0, N], entonces podemos minimizar la función de coste que incluye la referencia

$$J_0 = \frac{1}{2}(Cx_N - r_N)^T Q_f (Cx_N - r_N) + \frac{1}{2}\sum_{k=0}^{N-1}((Cx_k - r_k)^T Q(Cx_k - r_k) + {u_k}^T R u_k) \tag{4.15}$$

donde $Q_f \geq 0, Q \geq 0, R > 0$, y el valor real de x_N es alcanzable.

Para resolver este problema del regulador lineal cuadrático (LQ), podríamos comenzar con la función hamiltoniana y a partir de ella tenemos

- Sistema de estado

$$x_{k+1} = f(x_k, u_k) \tag{4.16}$$

- Sistema de co-estado

$$\lambda_k = \left(\frac{\partial f}{\partial x_k}\right)^T \lambda_{k+1} + C^T Q C x_k - C^T Q r_k \tag{4.17}$$

- Condición de estacionariedad

$$0 = \left(\frac{\partial f}{\partial u_k}\right)^T \lambda_{k+1} + R u_k \tag{4.18}$$

con las condiciones

$$x_0 \; es \; dado \tag{4.19}$$

$$\lambda_N = C^T Q_f (Cx_N - r_N) \tag{4.20}$$

A partir de la condición de estacionariedad (43), el control óptimo es

$$u_k = -R^{-1}\left(\frac{\partial f}{\partial u_k}\right)^T \lambda_{k+1} \tag{4.21}$$

La entrada Jacobiana, en general, depende de x_k y u_k de una manera no lineal, de modo que el control óptimo es, en general, una retroalimentación no lineal del estado y del co-estado.

El efecto de seguimiento de referencia r_k añade términos adicionales a las ecuaciones del regulador normal. De (4.20), es evidente que el co-estado final ya no es proporcional al estado como lo fue en el problema del regulador básico. Además, aunque podríamos utilizar (4.21) para eliminar u_k en las ecuaciones de estado y co-estado, está claro de (4.17) que el sistema hamiltoniano resultante ya no es homogéneo; Ahora es impulsado por una función de forzamiento $C^T Q r_k$ que es dependiente de la trayectoria deseada.

Dicho lo anterior, volviendo al caso lineal (4.14) se tiene que

$$x_{k+1} = Ax_k + Bu_k$$

$$\lambda_k = A^T \lambda_{k+1} + C^T QCx_k - C^T Qr_k \tag{4.22}$$

$$0 = B^T \lambda_{k+1} + Ru_k \tag{4.23}$$

$$u_k = -R^{-1} B^T \lambda_{k+1} \tag{4.24}$$

El sistema Hamiltoniano no homogéneo sería

$$\begin{bmatrix} x_{k+1} \\ \lambda_k \end{bmatrix} = \begin{bmatrix} A & -BR^{-1}B^T \\ C^T QC & A^T \end{bmatrix} \begin{bmatrix} x_k \\ \lambda_{k+1} \end{bmatrix} + \begin{bmatrix} 0 \\ -C^T Q \end{bmatrix} r_k \tag{4.25}$$

Podemos expresar u_k como una combinación de una respuesta de estado variable lineal más un término dependiendo de r_k. De la forma que (4.20), parece razonable asumir que para todo $k \leq N$, podemos escribir

$$\lambda_k = S_k x_k - v_k \tag{4.26}$$

con unas secuencias auxiliares todavía desconocidas S_k y v_k. Obsérvese que S_k es una matriz $n \times n$, mientras que v_k es un vector de dimensión n. Esto resultará ser una suposición válida si se pueden encontrar ecuaciones consistentes para S_k y v_k. Para encontrar estas ecuaciones, sustituiremos (4.26) en la parte de la ecuación de estado (4.25) obteniendo

$$x_{k+1} = Ax_k - BR^{-1}B^T S_{k+1} x_{k+1} + BR^{-1}B^T v_{k+1}$$

que si resolvemos para x_{k+1} se obtiene

$$x_{k+1} = (I - BR^{-1}B^T S_{k+1})^{-1}(Ax_k - BR^{-1}B^T v_{k+1}) \tag{4.27}$$

Usando (4.26) y (4.27) en la ecuación de co-estado se obtiene

$$\begin{aligned} S_k x_k - v_k = {} & C^T QCx_k + A^T S_k (I - BR^{-1}B^T S_{k+1})^{-1}(Ax_k - BR^{-1}B^T v_{k+1}) \\ & - A^T v_{k+1} - C^T Qr_k \end{aligned} \tag{4.28}$$

Esta ecuación debe ser válida para todas las secuencias de estados x_k dada cualquier x_0, de modo que los términos entre corchetes deben desaparecer individualmente. Usando el lema de inversión de matriz, podemos escribir

$$S_k = A^T [S_{k+1} - S_{k+1} B(B^T S_{k+1} B + R)^{-1} B^T S_{k+1}] A + C^T QC \tag{4.29}$$

y

$$v_k = [A^T - A^T S_{k+1} B(B^T S_{k+1} B + R)^{-1} B^T] v_{k+1} + C^T Q r_k \tag{4.30}$$

Al comparar (4.22) y (4.20), las condiciones de contorno para estas recursiones se consideran como

$$S_N = C^T Q_f C \tag{4.31}$$

$$v_N = C^T Q_f r_N \tag{4.32}$$

Puesto que las secuencias S_k y v_k ya pueden ser computadas, asumiendo que la suposición (4.26) fue válida, el control óptimo será

$$u_k = -R^{-1} B^T \lambda_{k+1} = -R^{-1} B^T (S_{k+1} x_{k+1} - v_{k+1}) \tag{4.33}$$

Sin embargo, todavía queda un paso más, ya que este control depende de x_{k+1}, que no se conoce en el momento k. Así pues, vamos a sustituir la ecuación de estado (4.22) en la (4.33), esto da lugar a

$$u_k = -R^{-1} B^T (S_{k+1}(A x_k + B u_k) + R^{-1} B^T v_{k+1})$$

si pre-multiplicamos por R y resolvemos para u_k

$$u_k = (B^T S_{k+1} B + R)^{-1} B^T (-S_{k+1} A x_k + v_{k+1}) \tag{4.34}$$

Esta es la ley de control que hemos estado buscando de la cual claramente se pueden extraer dos ganancias

$$K_k = (B^T S_{k+1} B + R)^{-1} B^T S_{k+1} A \tag{4.35}$$

$$K_k^v = (B^T S_{k+1} B + R)^{-1} B^T \tag{4.36}$$

En resuenen, se muestran todas las ecuaciones obtenidas necesarias para diseñar el algoritmo de control.

$$u_k = -K_k x_k + K_k^v v_{k+1}$$

$$K_k = (B^T S_{k+1} B + R)^{-1} B^T S_{k+1} A$$

$$K_k^v = (B^T S_{k+1} B + R)^{-1} B^T$$

$$S_k = A^T S_{k+1}(A - B K_k) + C^T Q C \qquad S_N = C^T Q_f C$$

$$v_k = (A - B K_k)^T v_{k+1} + C^T Q r_k \qquad v_N = C^T Q_f r_N$$

Con la planta en bucle cerrado bajo la influencia de este control, obtenemos el sistema no homogéneo que varía en el tiempo

$$x_{k+1} = (A - B K_k) x_k + B K_k^v v_{k+1}$$

En el siguiente apartado se muestra su implementación en Matlab y más adelante aparecen los resultados de las simulaciones. En el capítulo 5 veremos cómo implementar este algoritmo en la Raspberry Pi 3 empleando programación en Python, que utilizaremos para controlar la planta real.

4.1.4. Implementación en Matlab

En la primera parte, implementaremos el regulador lineal cuadrático clásico cuya función es llevar los estados cero, esto coincide con la condición cuando las derivadas de los estados sean nulas. Podríamos llevar el sistema a un punto concreto, es decir una posición determinada, cambiando nuestro sistema de coordenadas para que en ese punto sea cero. Es obvio que, si estamos trabajando con un sistema no lineal, primero hay que linealizar para un punto de funcionamiento y obtener el modelo discreto lineal en el espacio de estados. Después partiendo de unas condiciones iniciales/estados iniciales, aplicaremos el LQR y veremos cómo evoluciona el comportamiento del sistema. Para ello hay que obtener la ley de control que implica la resolución de la ec. diferencial de Riccati con la que obtendremos las ganancias K de realimentación para cada instante.

```
%% Solución de la ecuación diferencial de Riccati
for k=N-1:-1:1
    K(:,:,k)=inv(R+B'*S*B)*B'*S*A;
    S=(A-B*K(:,:,k))'*S*(A-B*K(:,:,k))+Q+K(:,:,k)'*R*K(:,:,k);
end
```

Código 2. LQR discreto, resolución ec. diferencia de Riccati y obtención de K_k.

```
Q = diag([10, 1,10,1,1,1,1,1]);
P=diag([100, 1,100,1,1,1,1,1]); R = [1 0; 0 0.1];
x0=[1.0400 0.3500 1.0400 0.3500 0 0 0 0]'
```

Código 3. Selección de matrices Q, R, S y estado inicial.

Una vez obtenidas las ganancias podemos simular el proceso con la realimentación de estas ganancias pada cada estado. El código para simular el proceso en bucle cerrado se muestra a continuación. Se almacena tanto la trayectoria generada, es decir la evolución de los estados, como la acción de control que se está aplicando en cada instante.

```
for k=1:N-1
    x(:,k+1) = A*x(:,k)-B*K(:,:,k)*x(:,k);
    u(:,k)=-K(:,:,k)*x(:,k);
end
```

Código 4. Simulación del proceso en bucle cerrado con LQR discreto.

En la segunda parte tenemos la implementación del regulador lineal cuadrático con seguimiento de trayectoria. De forma similar, el proceso consiste básicamente en resolver de forma recursiva, hacia atrás en el tiempo, la ecuación diferencial de Riccati, con el nuevo término de la secuencia auxiliar v_k que deriva del seguimiento de la referencia.

Con esto se podrán determinan las dos matrices de ganancias K y Kv. También cabe resaltar que la trayectoria empleada en este caso es una trayectoria trapezoidal de cuarto orden, de la cual, los parámetros que utilizaremos son la posición en cada instante `qrefX(k)` y la velocidad en cada instante `vrefX(k)`. Como tenemos dos eslabones se han generado dos trayectorias.

```
for k=N-1:-1:1
    K(:,:,k)=inv(R+B'*S*B)*B'*S*A;
    Kv(:,:,k)= inv(R+B'*S*B)*B';
    S=A'*S*(A-B*K(:,:,k))+ C'*Q*C;
    v(:,k)=(A-
B*K(:,:,k))'*v(:,k+1)+C*Q*[qref1(k);qref2(k);qref1(k);qref2(k);vref1
(k);vref2(k);vref1(k);vref2(k)];
end
```

Código 5. LQR discreto con seguimiento de trayectoria, resolución ec. diferencia de Riccati y secuencia auxiliar v_k.

El segundo paso es simular y analizar los resultados. El código que simula el proceso en bucle cerrado se muestra a continuación.

```
for k=1:N-1
    x(:,k+1) = A*x(:,k)-B*K(:,:,k)*x(:,k)+B*Kv(:,:,k)*v(:,k+1);
    u(:,k)=-K(:,:,k)*x(:,k)+Kv(:,:,k)*v(:,k+1);
end
```

Código 6. Simulación del proceso en bucle cerrado con LQR discreto con seguimiento de trayectoria.

con él se obtiene, por una parte, la trayectoria generada al aplicar la acción de control correspondiente y por otra, la misma acción de control. Estas dos variables se almacenan, cada una de ellas en un vector/matriz, que después representaremos de forma gráfica junto con la referencia de la trayectoria para analizar y extraer conclusiones del seguimiento de la trayectoria.

Otro aspecto importante es que si igualamos Q y r_k a cero, teniendo solo r_N entonces lo que estamos haciendo es conducir el estado cerca de un valor deseado r_N sin usar demasiada energía de control. Esto también nos servirá para comparar los resultados con el iLQR.

4.1.5. Simulación

En primer lugar, vamos a probar el regulador lineal cuadrático clásico. Partiendo del modelo linealizado discreto obtenido en el en el apartado 2.3 y empleando las siguientes matrices $Q, R\ y\ S$ y el estado inicial x_0.

```
Q = diag([10, 1,10,1,1,1,1,1]); R = [1 0; 0 0.1];
S = diag([100, 1,100,1,1,1,1,1]);
x0 = [1.0400 0.3500 1.0400 0.3500 0 0 0 0]'
```

Código 7. Matrices Q, R, S y las condiciones iniciales.

Hay que aclarar que a partir de aquí y en adelante cuando nos referimos a la matriz S, se hace referencia a la matriz Q_f. Sin olvidar que S también hace referencia a la secuencia que resolvemos con la ecuación diferencial de Riccati y por tanto $S_N = Q_F$. También decir que la posición angular inicial, se da cuando el brazo robot está situado en la posición inferior y se corresponde con 0 rad.

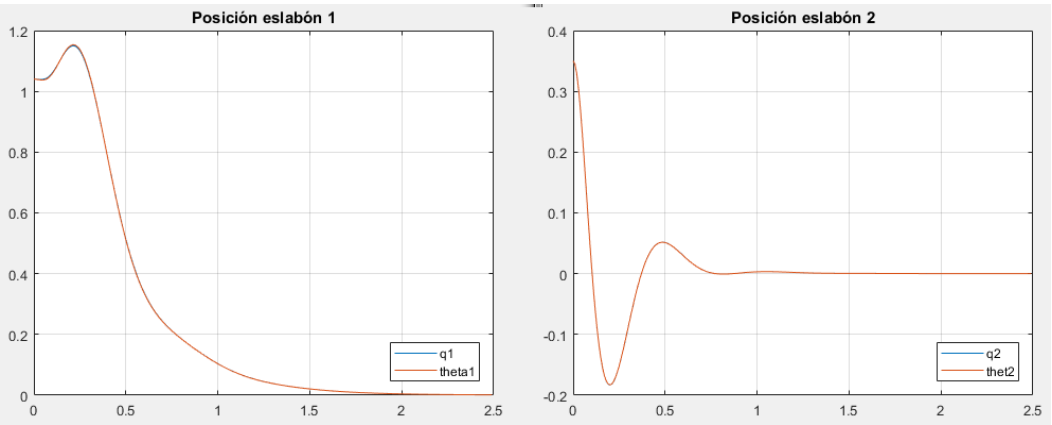


Figura 19. Evolución de la posición

Partiendo de las condiciones iniciales, es decir una posición y velocidad determinada, vemos en la figura 19 la evolución de la posición de los dos eslabones hasta llegar a cero.

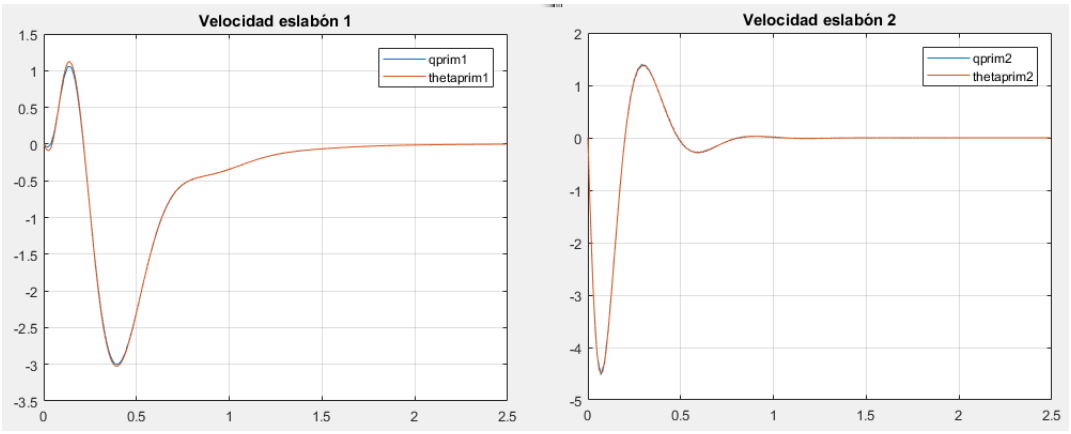


Figura 20. Evolución de la velocidad

En la figura 20 se muestra la evolución de la velocidad de los dos eslabones hasta alcanzar velocidad nula. De modo que, el LQR está cumpliendo con su trabajo al estar llevando los estados a cero. Finalmente, en la figura 21 tenemos la acción de control aplicada. Es importante indicar que se requiere mucho par, aun así, hay que tener en cuenta que se dispone de un reductor anclado con lo cual alcanzar ese rango de pares es posible.

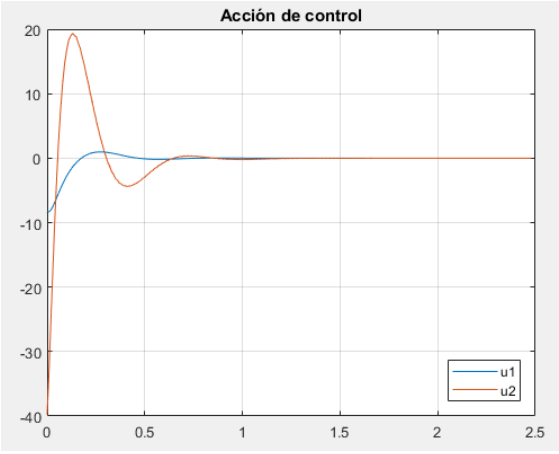


Figura 21. Acción de control

En segundo lugar, tenemos el LQR con seguimiento de trayectoria. El primero paso sería seleccionar bien las matrices Q, R y S ya que en caso contrario no se alcanzaría la posición final. En la figura 22 se muestra un ejemplo de comportamiento no deseado, en la que los valores de la matriz S son bajos.

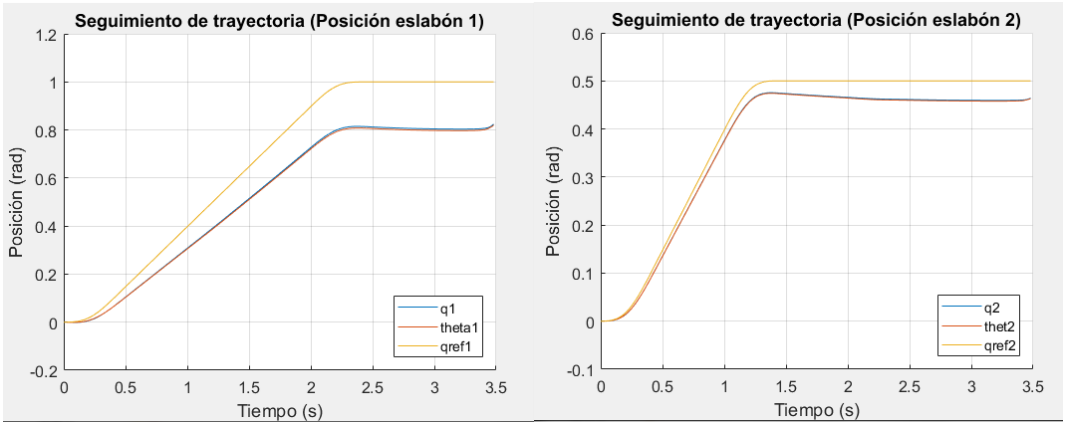


Figura 22. Seguimiento de la trayectoria de la posición con LQR.

Como se puede observar a partir de las gráficas, ninguno de los dos eslabones es capaz de alcanzar la referencia. Es necesario incrementar los valores de la matriz S para alcanzar la posición final. Tras realizar varias simulaciones en búsqueda de una buena respuesta, finalmente se ha concluido que las matrices Q, R y S que lo consiguen, podrían ser:

```
Q = diag([100,100, 100, 100, 100, 100, 100, 100]);
S = diag([1000,1000,1000,1000,1000,1000,1000,1000]);
R = [0.0001 0; 0 0.0001];
```

Código 8. Experimento 1: matrices Q, R y S.

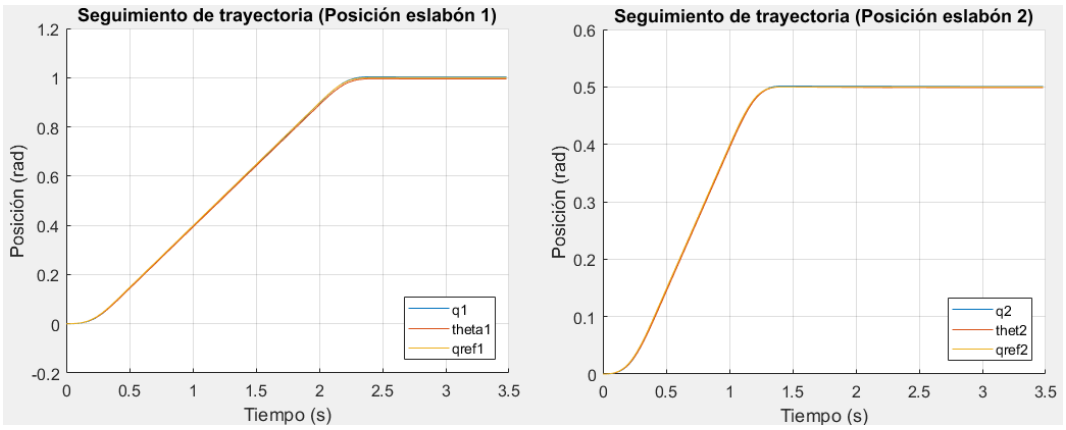


Figura 23. Experimento 1: seguimiento de la trayectoria de la posición con LQR.

Ahora sí, a partir de la figura 23 podemos afirmar que, este controlador es capaz de llevar el sistema desde el estado inicial hasta el estado final siguiendo la trayectoria indicada. El **error cuadrático de posición** del primer elemento es **2.0022e-05 rad** y del segundo **4.9125e-06 rad**. En comparación con la que obtuvimos con el PD, esta es considerablemente mejor sobre todo si nos fijamos en la respuesta de velocidad.

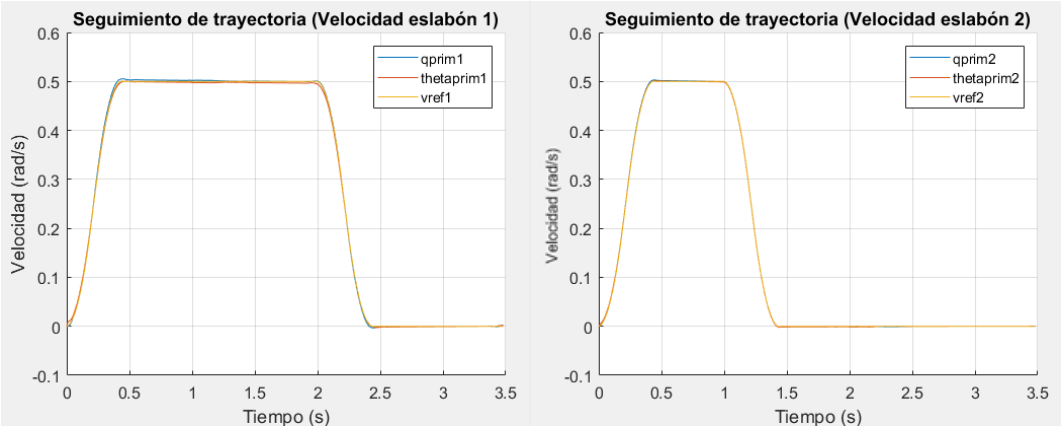


Figura 24. Experimento 1: seguimiento de la trayectoria de la posición con LQR.

Es evidente que, el correcto seguimiento de la posición, también implica el correcto seguimiento de la velocidad como es el caso. El **error cuadrático de velocidad** es de **0.1144 rad/s** para el primer elemento y **0.0640 rad/s** para el segundo. En la figura 25 se representa la evolución de la velocidad con respecto de la posición.

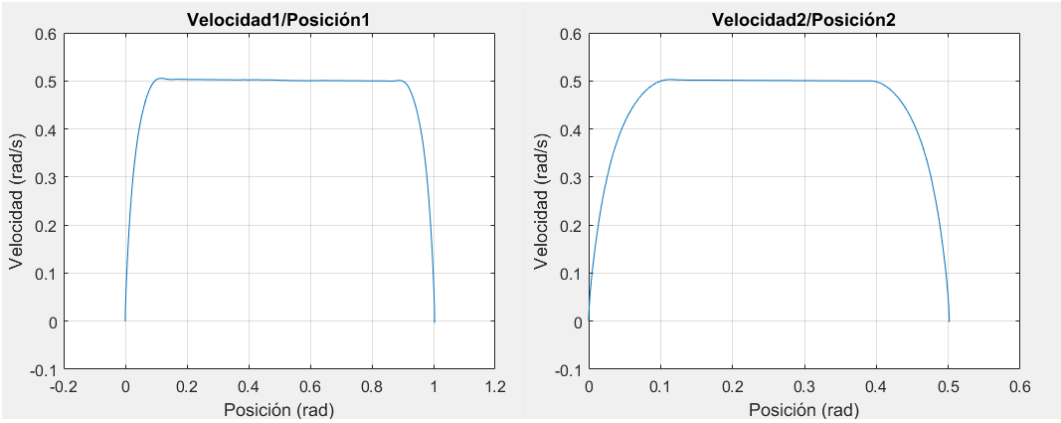


Figura 25. Experimento 1: velocidad frente a posición

Finalmente tenemos la acción de control, observe que prácticamente no hay cambios bruscos, es decir evoluciona de forma suave.

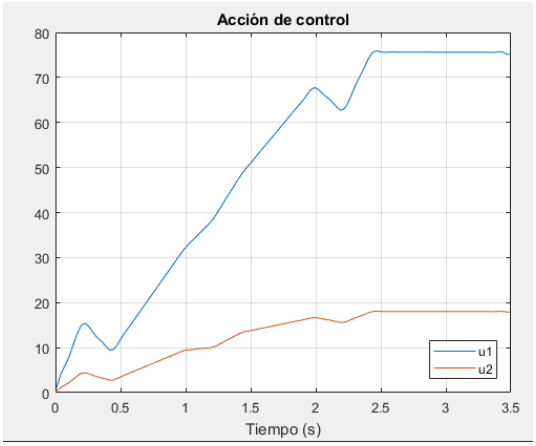


Figura 26. Experimento 1: acción de control.

Si comparamos esta acción de control con la que obtuvimos con el regulador PD donde la respuesta era muy oscilante, podemos afirmar que la del LQR es bastante mejor.

Como ya se mencionó anteriormente, uno de los objetivos de este trabajo es **optimizar el consumo de energía**, para ello analizaremos la importancia de las matrices Q, R y S y como afectan a la trayectoria seguida de posición y con ello, ver si es posible llegar a la posición deseada, minimizando el consumo, aunque sea a costo de un seguimiento de la trayectoria no perfecto. No existe una regla determinada para hacerlo, sin embargo, la estrategia LQR tiene la ventaja de que independientemente de la selección de Q, R y S sigue siendo capaz de estabilizar al sistema.

Existe una relación de proporcionalidad entre par e intensidad $\tau = i \cdot K_m$. Al realizar control por corriente, podríamos utilizar el **valor medio cuadrático del par** como referencia del consumo, cuanto más par más consumo de corriente. Para el primer experimento y en concreto para el primer eslabón es **59.4859 Nm** y para el segundo eslabón es de **14.6762 Nm**. Utilizaremos estos datos como valores de referencia y a partir de ellos obtendremos una estimación del ahorro de energía.

Hay que aclarar que, manteniendo los valores de la matriz R, los valores de la matriz S se han fijado a un valor con el que se ha comprobado previamente que es posible llegar al punto final deseado. Después se han ido incrementando los valores de la matriz Q que afectan, en mayor o menor medida la precisión del seguimiento de la trayectoria de referencia. De modo que, para el segundo experimento se emplearán las siguiente matrices Q, R y S

```
Q = diag([10,10, 10, 10, 10, 10, 10, 10]);
S = diag([1000,1000,1000,1000,1000,1000,1000,1000]);
R = [0.0001 0; 0 0.0001];
```

Código 9. Experimento 2: matrices Q, R y S.

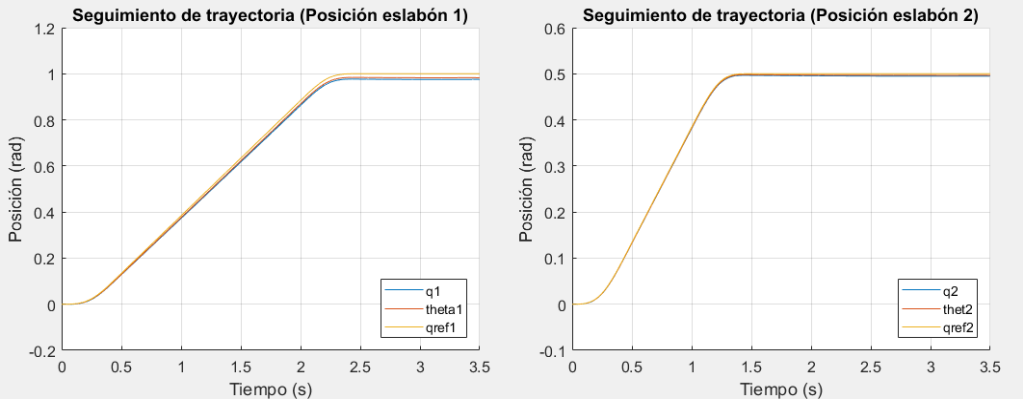


Figura 27. Experimento 2: seguimiento de la trayectoria de la posición con LQR.

A partir de la figura 26 podemos decir que aparentemente no hay mucha diferencia en el seguimiento de la trayectoria con respecto al caso del experimento 1. El nuevo **error cuadrático de posición** para el eslabón 1 es de **3.3376e-04 rad** frente a los **2.0022e-05 rad** del experimento anterior y para el segundo **7.0030e-05 rad** frente a los **4.9125e-06 rad** del experimento 1. Una diferencia relativamente pequeña.

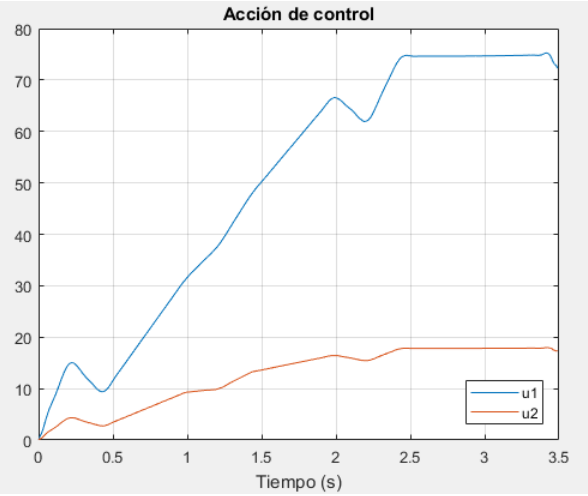


Figura 28. Experimento 2: acción de control.

En cuanto a la acción de control, se puede observar que la respuesta es muy similar a la obtenida del experimento anterior. En cuanto al **valor medio cuadrático del par,** este es de **54.6575 Nm** para el primer eslabón y de **13.5427 Nm** para el segundo. Una diferencia de 4.82 Nm para el caso del eslabón 1 y 1,13 Nm para el segundo.

Siguiendo con el experimento 3 emplearemos los siguientes datos:

```
Q = diag([1,1, 1, 1, 1, 1, 1, 1);
S = diag([1000,1000,1000,1000,1000,1000,1000,1000]);
R = [0.0001 0; 0 0.0001];
```

Código 10. Experimento 3: matrices Q, R y S.

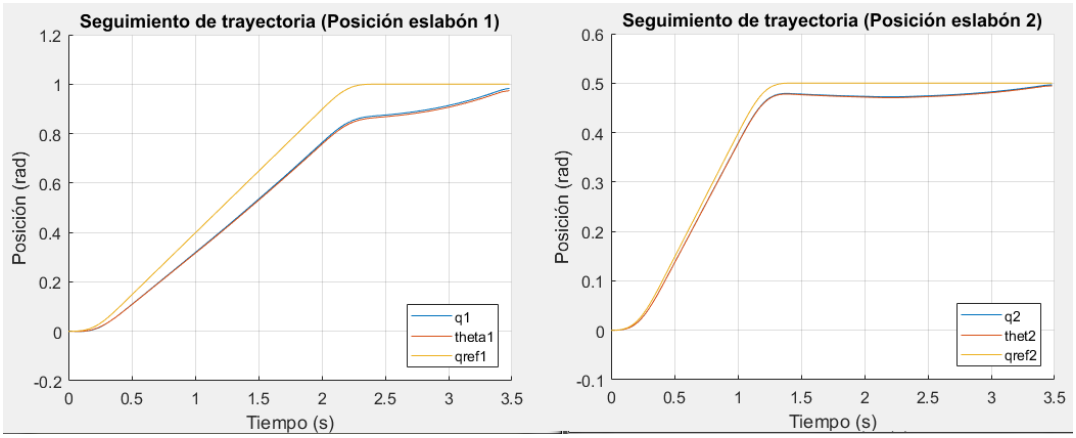


Figura 29. Experimento 3: seguimiento de la trayectoria de la posición con LQR.

Como se puede observar analizando la gráfica anterior ya se está empezando a notar que el seguimiento de la trayectoria no es el indicado. Es más, si obtenemos el **error cuadrático medio** de la posición vemos que es de **0.0089 rad** para el primer eslabón y **0.0066 rad** para el segundo. Sin embargo, se alcanzan los puntos finales de la trayectoria. El cambio es aún más notable para la velocidad.

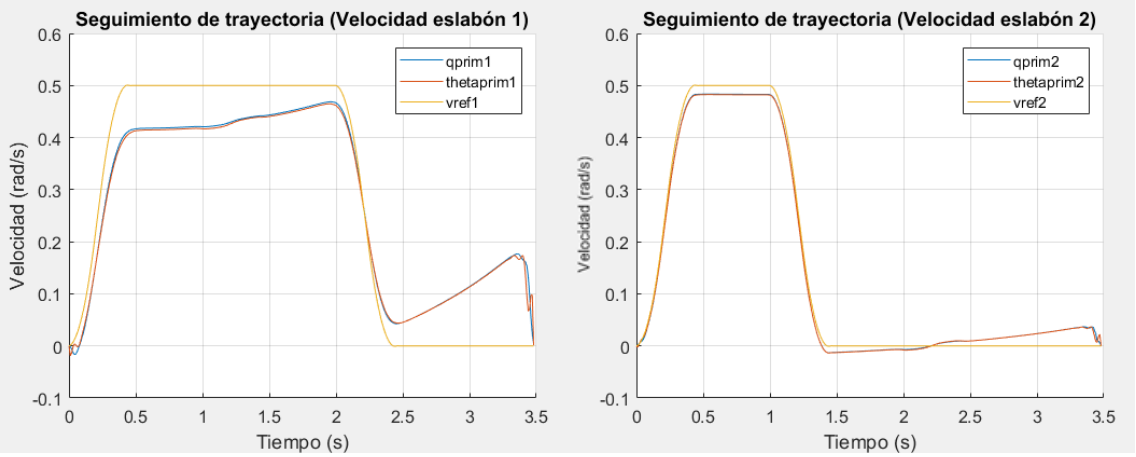


Figura 30. Experimento 3: seguimiento de la trayectoria de la posición con LQR.

En la figura 29 vemos claramente que el seguimiento de la trayectoria de velocidad para el primer eslabón no tiene que ver mucho con la trayectoria de referencia de velocidad. También se observa que en el tramo [2.5 – 3.5] segundos, la velocidad no se mantiene constante a cero, sino que aumenta y finalmente se frena. El **error cuadrático de velocidad** para cada eslabón es de **0.1260 rad/s** y **0.0671 rad/s** respectivamente.

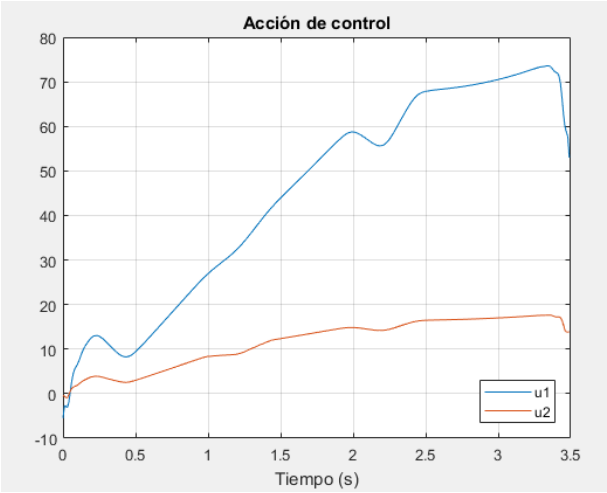


Figura 31. Experimento 3: acción de control

En cuanto a la acción de control, en comparación con los experimentos anteriores, la pendiente es más suave. El **valor medio cuadrático del par** para el primer eslabón es de **50.3792 Nm** y para el segundo **12.6785 Nm**. Obteniendo así, una reducción del consumo del orden de un 10 % respecto del experimento 1.

Por último, para el experimento 4, la nueva matriz Q es:

```
Q = diag([0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1);
S = diag([1000,1000,1000,1000,1000,1000,1000,1000]);
R = [0.0001 0; 0 0.0001];
```

Código 11. Experimento 4: matrices Q, R y S.

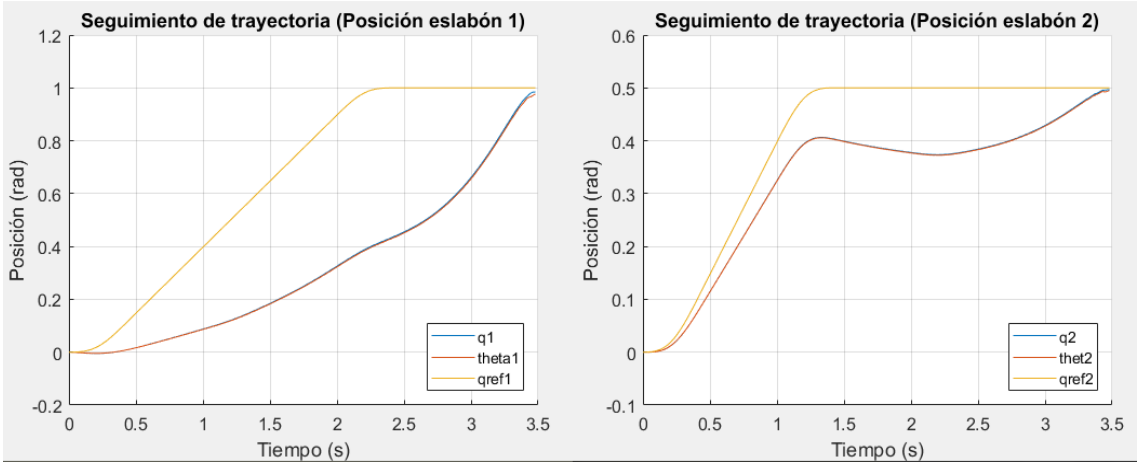


Figura 32. Experimento 4: seguimiento de la trayectoria de la posición con LQR.

Se observa que la trayectoria seguida de posición del primer eslabón es similar a una función exponencial. Aquí prácticamente no podemos hablar de seguimiento de trayectoria, aunque es verdad que el eslabón 2 lo hace en la mayor parte. En cuanto al **error cuadrático de posición** para el primer eslabón es de **0.1531 rad** y para el segundo es de **0.1474 rad**.

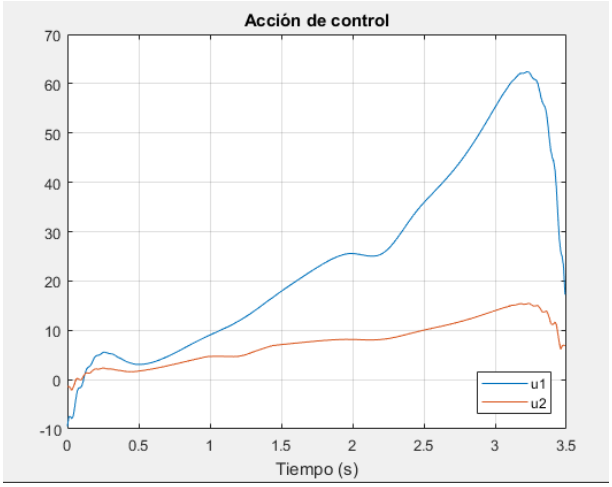


Figura 33. Experimento 4: acción de control

En cuanto a la acción de control, se ve claramente que la pendiente es más pequeña. Además, en comparación con el resto de experimento, aquí el par máximo alcanzado es el más bajo, 60 Nm frente a los 75 Nm del primer experimento. En cuanto al **valor medio cuadrático del par**, este es de **30.5481 Nm** para el motor del primer eslabón y **8.5102 Nm** para el segundo eslabón. Esto equivale a aproximadamente un **48% de ahorro de energía** del primer eslabón y un 41 % del segundo.

Para concluir este apartado, podemos decir que, el control LQR es una técnica muy eficaz que además de permitir un perfecto seguimiento de trayectorias cuando es lo que se desea, puede ser empleado para el manejo del ahorro de consumo.

4.2. Regulador Lineal Cuadrático iterativo (iLQR)

En los apartados 4.1. y 4.1.4 hablamos del LQR y método con seguimiento de trayectoria, sin embargo, cabe destacar que la principal desventaja de este regulador es que opera sin tener en cuenta los cambios en el futuro. Se optimiza asumiendo la dinámica lineal aproximada en un punto para todo el tiempo. De modo que sería muy bueno tener un algoritmo que fuera capaz de planificar y optimizar una secuencia, consciente de la dinámica cambiante del sistema.

A continuación, hablaremos del método iterativo del regulador lineal cuadrático (iLQR), para el control de sistemas dinámicos no lineales. Se trata de una extensión del control LQR, y la idea aquí es básicamente optimizar una secuencia de control completa en lugar de sólo la señal de control para el punto actual en el tiempo. El nuevo método utiliza la linealización iterativa del sistema no lineal alrededor de una trayectoria nominal y calcula una ley de control de realimentación localmente óptima a través de una técnica LQR modificada. Esta ley de control se aplica entonces al sistema linealizado, y el resultado se utiliza para mejorar la trayectoria nominal de forma incremental.

Dado el sistema dinámico no lineal en tiempo discreto con variable de estado $x_k \in R^{n_x}$ y control $u_k \in R^{n_u}$

$$x_{k+1} = f(x_k + u_k) \tag{4.37}$$

y la función de coste

$$J_0 = \frac{1}{2}(x_N - x^*)^T Q_f (x_N - x^*) + \frac{1}{2}\sum_{k=0}^{N-1}({x_k}^T Q x_k + {u_k}^T R u_k), \tag{4.38}$$

donde x_N equivale al estado final y x^* es el objetivo que se desea alcanzar. Las matrices que penalizan los estados Q y Q_f son simétricas y positivas semi-definidas, la matriz que penaliza la acción de control R es definida positiva. Además, todas estas matrices tienen que tener las dimensiones adecuadas.

El algoritmo consiste en que, cada iteración comienza con una secuencia de control nominal u_k, y una trayectoria nominal correspondiente x_k obtenida aplicando u_k al sistema dinámico en bucle abierto. La inicialización es u_k = 0. La iteración produce una secuencia mejorada de u_k, al linealizar la dinámica del sistema alrededor de u_k, x_k y resolviendo el LQR modificado. El proceso se repite hasta la convergencia o hasta N iteraciones. Sean las desviaciones del u_k nominal, x_k sea δu_k, δx_k. La linealización es

$$\delta x_{k+1} = A_k \delta x_k + B_k \delta u_k, \tag{4.39}$$

donde $A_k = D_x f(x_k + u_k)$, $B_k = D_u f(x_k + u_k)$. D_x denota el Jacobiano de $f(.)$ con respecto a x, D_u denota el Jacobiano de $f(.)$ con respecto a u, y los Jacobianos son evaluados a lo largo de x_k y u_k.

A partir del modelo linealizado (4.39), podemos resolver el problema de LQR con la siguiente función de coste

$$J = \frac{1}{2}(x_N + \delta x_N - x^*)^T Q_f (x_N + \delta x_N - x^*) + \frac{1}{2}\sum_{k=0}^{N-1}[(x_k + \delta x_k)^T Q(x_k + \delta x_k) + (u_k + \delta u_k)^T R(u_k + \delta u_k)] \tag{4.40}$$

Sea la función hamiltoniana

$$H_k = (x_k + \delta x_k)^T Q(x_k + \delta x_k) + (u_k + \delta u_k)^T R(u_k + \delta u_k) + \delta\lambda_{k+1}^T (A_k \delta x_k + B_k \delta u_k) \tag{4.41}$$

donde $\delta\lambda_{k+1}$ es un multiplicador de LaGrange

La mejora del control óptimo δu_k se obtiene resolviendo la ecuación de estado (4.39), igualando la siguiente ecuación

$$\delta\lambda_k = \mathrm{A}_k^T \delta\lambda_{k+1} + Q(x_k + \delta x_k) \tag{4.42}$$

y la condición estacionaria que se puede obtener poniendo la derivada de la función hamiltoniana con respecto a δu_k a cero

$$0 = R(u_k + \delta u_k) + \mathrm{B}_k^T \delta\lambda_{k+1} \tag{4.43}$$

con la condición final

$$\delta\lambda_N = Q_f(x_N + \delta x_N - x^*) \tag{4.44}$$

Resolviendo a partir de (4.43)

$$\delta u_k = -R^{-1}\mathrm{B}_k^T \delta\lambda_{k+1} - u_k \tag{4.45}$$

Por lo tanto, sustituyendo (4.45) en (4.39) y combinándolo con (4.42), el sistema hamiltoniano resultante es

$$\begin{pmatrix} \delta x_{k+1} \\ \delta\lambda_k \end{pmatrix} = \begin{pmatrix} A_k & B_k R^{-1}\mathrm{B}_k^T \\ Q & A_k^T \end{pmatrix} \begin{pmatrix} \delta x_k \\ \delta\lambda_{k+1} \end{pmatrix} + \begin{pmatrix} -B_k u_k \\ Q x_k \end{pmatrix} \tag{4.46}$$

Se puede observar que el sistema hamiltoniano no es homogéneo, sino que es impulsado por un término forzante dependiente de la trayectoria actual x_k y u_k. Debido a ese término, no es posible expresar la ley de control óptima en forma de retroalimentación de estado lineal (como en el caso del LQR clásico). Sin embargo, podemos expresar δu_k como una combinación de una realimentación de estado lineal más términos adicionales, que dependen de la función de forzamiento.

A partir de la condición (4.44), asumimos

$$\delta\lambda_k = S_k \delta x_k + v_k \tag{4.47}$$

Para algunas secuencias desconocidas S_k y v_k. Sustituyendo la suposición anterior en la ecuación (4.39) y (4.42), y aplicando el lema de inversión de la matriz se obtiene el controlador óptimo

$$\delta u_k = -K\delta x_k - K_v v_{k+1} - K_u u_k \tag{4.48}$$

$$K = (\mathrm{B}_k^T S_{k+1} B_k + R)^{-1} \mathrm{B}_k^T S_{k+1} A_k \tag{4.49}$$

$$K_v = (\mathrm{B}_k^T S_{k+1} B_k + R)^{-1} \mathrm{B}_k^T \tag{4.50}$$

$$K_u = (\mathrm{B}_k^T S_{k+1} B_k + R)^{-1} R \tag{4.51}$$

$$S_k = \mathrm{A}_k^T S_{k+1}(A_k - B_k K) + Q \tag{4.52}$$

$$v_k = (A_k - B_k K) v_{k+1} - K^T R u_k + Q x_k \tag{4.53}$$

con las condiciones finales

$$S_N = Q_f, \qquad v_N = Q_f(x_N - x^*) \tag{4.54}$$

La obtención de las ecuaciones (4.48) - (4.53) se obtienen al sustituir (11) en la ecuación de estado (4.39) para obtener

$$\delta x_{k+1} = (I + B_k R^{-1} \mathrm{B}_k^T S_{k+1})^{-1}(A_k \delta x_k - B_k R^{-1} \mathrm{B}_k^T v_{k+1} - B_k u_k) \tag{4.55}$$

Sustituyendo (4.47) y la ecuación anterior en la ecuación (4.42) se obtiene

$$\begin{aligned} S_k \delta x_k + v_k = Q\delta x_k + \mathrm{A}_k^T S_{k+1}(I + B_k R^{-1} \mathrm{B}_k^T S_{k+1})^{-1} \\ (A_k \delta x_k - B_k R^{-1} \mathrm{B}_k^T v_{k+1} - B_k u_k) + \mathrm{A}_k^T v_{k+1} + Q x_k \end{aligned} \tag{4.56}$$

Aplicando el lema de la inversión matricial, obtenemos

$$S_k = \mathrm{A}_k^T v_{k+1}[I - B_k(\mathrm{B}_k^T S_{k+1} B_k + R)^{-1} \mathrm{B}_k^T S_{k+1} A_k + Q] \tag{4.57}$$

y

$$\begin{aligned} v_k = \mathrm{A}_k^T v_{k+1} - \mathrm{A}_k^T S_{k+1}[I - B_k(\mathrm{B}_k^T S_{k+1} B_k + R)^{-1} \mathrm{B}_k^T S_{k+1}] B_k R^{-1} \mathrm{B}_k^T v_{k+1} \\ - \mathrm{A}_k^T S_{k+1}[I - B_k(\mathrm{B}_k^T S_{k+1} B_k + R)^{-1} \mathrm{B}_k^T S_{k+1}] B_k u_k + Q x_k \end{aligned} \tag{4.58}$$

Haciendo que $(R + \mathrm{B}_k^T S_{k+1} B_k)^{-1} = R^{-1} - (R + \mathrm{B}_k^T S_{k+1} B_k)^{-1} \mathrm{B}_k^T S_{k+1} B_k R^{-1}$, el segundo término en v_k se convierte en

$$-\mathrm{A}_k^T S_{k+1} B_k (R + \mathrm{B}_k^T S_{k+1} B_k)^{-1} \mathrm{B}_k^T v_{k+1}$$

y el tercer término en v_k puede ser reescrito como

$$-\mathrm{A}_k^T S_{k+1} B_k (R + \mathrm{B}_k^T S_{k+1} B_k)^{-1} \mathrm{R} u_k$$

Por lo tanto, con la definición de K en (4.48), el S_k anterior y v_k se pueden escribir de la forma dada en (4.52) y (4.53).

Además, al sustituir (4.47) y (4.54) en (4.45) se obtiene

$$\begin{aligned} \delta u_k = -(R + \mathrm{B}_k^T S_{k+1} B_k)^{-1} \mathrm{B}_k^T S_{k+1} A_k \delta x_k - (\mathrm{B}_k^T S_{k+1} B_k + R)^{-1} \mathrm{B}_k^T v_{k+1} \\ - (\mathrm{B}_k^T S_{k+1} B_k + R)^{-1} R u_k \end{aligned} \tag{4.59}$$

Dadas las definiciones de K, K_v y K_u en (4.49) - (4.51), podemos reescribir δu_k tal y como aparece en (4.48).

Con la condición, dada S_N como la matriz de ponderación del estado final en la función de coste (4.40), podemos resolver para una secuencia entera de S_k recurriendo hacia atrás (4.51). Es interesante notar que la ley de control δu_k consiste en tres términos: un término lineal δx_k cuya ganancia depende de la solución a la ecuación de Riccati; un segundo término dependiente de una secuencia auxiliar v_k que se deriva de la ecuación diferencia (4.52); y un tercer término dependiente del control nominal u_k cuya ganancia también se basa en la solución de la ecuación de Riccati. Una vez resuelto el problema de LQR modificado, se puede encontrar una secuencia de control nominal mejorada: $u_k^* = u_k + \delta u_k$

En la mayoría de las aplicaciones de robótica, normalmente se exige un seguimiento de trayectorias, de modo que, en vez de tener solo un punto de referencia para el estado final, se tendría una secuencia de puntos de referencia. Este caso ya lo analizamos en el LQR con seguimiento de trayectoria. Es importante señalar que no hay garantías de que el objetivo que se desea alcanzar x^* sea una trayectoria factible, en la mayoria de los casos no lo es.

En resumen, para tratar de entender mejor cómo funciona el algoritmo iLQR se facilita el siguiente diagrama en el que se describe de forma simplificada, los pasos que se han de seguir.

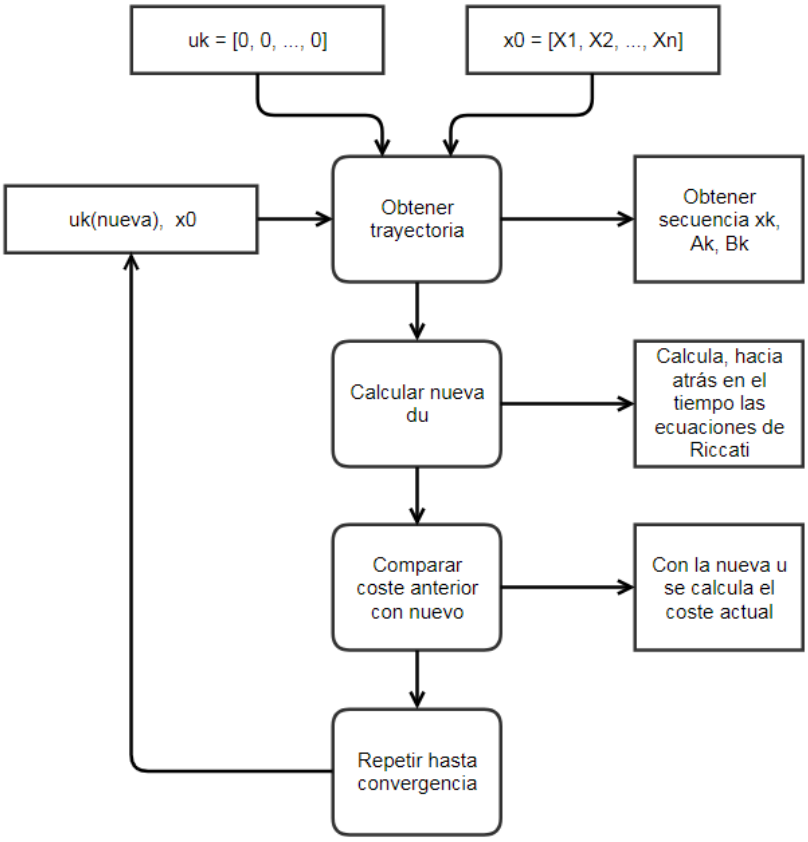


Figura 34. Diagrama del algoritmo iLQR.

4.2.1. Implementación en Matlab

A continuación, se muestra la función implementada en Matlab que resuelve las ecuaciones de Riccati hacia atrás en el tiempo de forma recursiva. Nótese que partimos de unas condiciones iniciales o estados iniciales x_0 y aplicamos una acción de control u_k al sistema, inicialmente $u_k = [0, 0 \dots, 0]^T$. Con ello generamos una secuencia/trayectoria x_k (simulación en bucle abierto), así como una secuencia de matrices A_k y B_k (de cada punto de la trayectoria x_k). Con estos datos, podemos calcular las matrices de ganancia K, K_v, K_u, así como las matrices S_k y v_k para así poder obtener δu_k y posteriormente $u_k^* = u_k + \delta u_k$. La idea es básicamente buscar la u_k^* que haga que la J converja.

```
function [du K Kv Ku] = ilqr_iterate(x0, xf, u, Qf, Q, R, dt)
    % Se crea matriz para las acciones de control
    du = zeros(size(u));
    % Simular en bucle abierto
    x = ilqr_sim(x0, u, dt);
    % Obtener tamaño de x e u
    N = length(x); Nx = length(x0); Nu = length(u(1,:));
    % Creación de matrices de ganancias K, Kv y Ku
    K = zeros(Nu, Nx, N-1); Kv = zeros(Nu, Nx, N-1);
    Ku = zeros(Nu, Nu, N-1);
    % Creación de matrices S y v
    S = zeros(Nx, Nx, N); v = zeros(N, Nx);
    % dx = x - x0
    dx = 0.001*ones(size(x));
    % Creación de Matrices A y B
    A = zeros(Nx, Nx, N-1); B = zeros(Nx, Nu, N-1);
    for i = 1:N-1
        % Linear Time Variant System LTV
        [A(:,:,i) B(:,:,i)] = ilqr_linearization_robot_arm(x(i,:),
u(i,:), dt);
    end
    % Condición final para las matrices S y v
    S(:,:,N) = Qf;  v(N,:) = (Qf*(x(N,:) - xf)')';
    % Bucle para calcular ecuaciones de Riccati
    for k = N-1:-1:1
        K(:,:,k) = inv(B(:,:,k)' * S(:,:,k+1) * B(:,:,k) + R) *
                   B(:,:,k)' * S(:,:,k+1) * A(:,:,k);
        Kv(:,:,k) = inv(B(:,:,k)' * S(:,:,k+1) * B(:,:,k) + R) *
                    B(:,:,k)';
        Ku(:,:,k) = inv(B(:,:,k)' * S(:,:,k+1) * B(:,:,k) + R) * R;
        S(:,:,k) = A(:,:,k)' * S(:,:,k+1) * (A(:,:,k) - B(:,:,k) *
                   K(:,:,k)) + Q;
        v(k,:) = ((A(:,:,k) - B(:,:,k) * K(:,:,k))' * v(k+1,:)' -
                 K(:,:,k)' * R * u(k,:)' + Q * x(k,:)')';
        du(k,:) = (-1) * (K(:,:,k)* dx(k,:)' + Kv(:,:,k) * v(k+1,:)'
                  + Ku(:,:,k) * u(k,:)')';
    end
end
```

Código 12. Función para obtener du (*backward pass*).

En el apartado 4.2 vimos que la función de coste para el sistema linealizado tenía la forma como se muestra en (4.38). De modo que se ha creado una función que nos permita calcular el coste en cada iteración.

```
function J = ilqr_cost(x0, xf, u,  Qf, Q, R, dt)
    x = ilqr_sim(x0, u, dt);
    N = length(x);
    %% Coste final
    J = (x(N,:) - xf) * Qf * (x(N,:) - xf)';
    for i = 1:N-1
        % Coste inmediato
        J = J + (x(i,:)) * Q * (x(i,:))' + u(i,:) * R * u(i,:)';
    end
    J = 1/2*J;
end
```

Código 13. Función de coste

Empezamos con una trayectoria inicial y un coste inicial, pero esto es sólo una suposición. Queremos que nuestro algoritmo lo tome y luego converge a un mínimo local. Para ello en cada iteración se calcula el incremento de acción de control y con ello la nueva trayectoria de $\boldsymbol{x}$, así como el nuevo coste. Observe que, para realizar estas operaciones, en cada iteración se llama a las dos funciones anteriores.

```
function traju = ilqr_control(x0, xf, u, Qf, Q, R, dt, iter,
conveg_cond)

    du = u;
    % Coste inicial
    J_old = ilqr_cost(x0, xf, du, Qf, Q, R, dt);

    for i = 1:iter
        % Se calculan los incrementos de control
        [du_new K Kv Ku] = ilqr_iterate(x0, xf, du, Qf, Q, R, dt);

        du =  du + 0.15 * du_new; %
        % Se vuelve a calcular el coste con para la nueva du
        J = ilqr_cost(x0, xf, du, Qf, Q, R, dt)

      % Se comparan los dos costes, sí son iguales quiere decir
      que el sistema ha convergido, en caso contrario hasta que
      terminen las iteraciones
        if(abs(J - J_old) < conveg_cond)
            break
        end;
        J_old = J;
    end;
    % Se obtienen la trayectoria u para alcanzar xf
    traju = du;
end
```

Código 14. Algoritmo iLQR.

Aparte hay más archivos[4], como el script principal para ejecutar el control iLQR que incorpora las condiciones iniciales y las matrices Q, S y R; así como, la funciones para linealizar el modelo para cada punto de la trayectoria y para obtener las matrices A y B.

[4] Se adjuntan con la memoria, en la carpeta iLQR.

4.2.2. Simulación

En esta sección se muestran los resultados obtenidos tras aplicar el regulador lineal cuadrático iterativo. Cabe destacar que la referencia que se quiere alcanzar es un punto, es decir, no se realiza un seguimiento de una trayectoria. En el primer experimento igualaremos Q a un valor muy próximo al cero.

```
S=diag([5000.0 5000.0 5000.0 5000.0 5000.0 5000.0 5000.0 5000.0]);
Q = diag([0.0001 0.0001 0.0001 0.0001 0.0001 0.0001 0.0001 0.0001]);
R = [0.001 0; 0 0.001];
```

Código 15. Experimento 1: matrices Q R y S (iLQR).

En la figura 35 se muestra la trayectoria seguida por los dos eslabones. Se puede observar que los dos eslabones alcanzan la referencia, aunque el segundo de una forma más suave. Tiene un comportamiento un poco extraño, primero se mueve en un sentido y luego avanza en sentido contrario, ganando inercia para llegar a la referencia. Este comportamiento es muy utilizado para estabilizar, por ejemplo, un péndulo invertido.

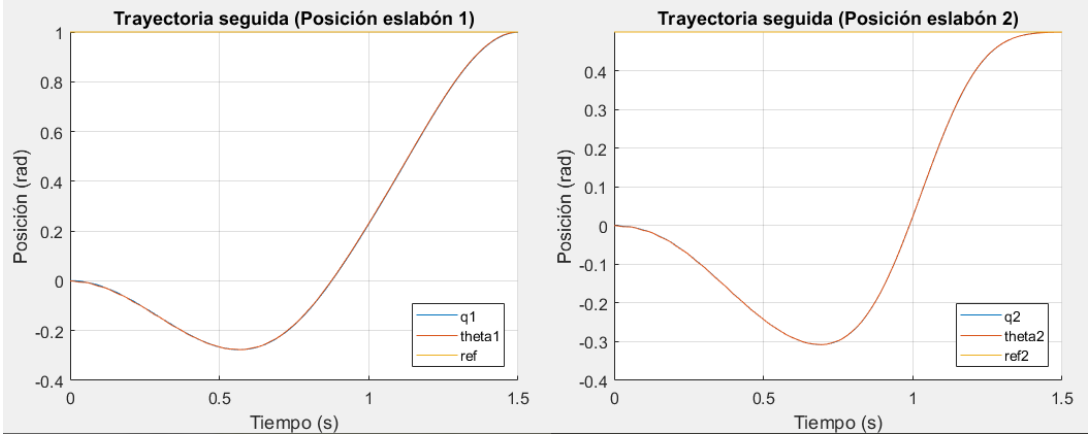


Figura 35. Experimento 1: trayectoria seguida por los eslabones de la posición (iLQR).

En el estado final se desea que las velocidades angulares sean nulas, este objetivo se consigue. En cuanto a la respuesta en sí, vemos que hay presencia de oscilaciones en ambas respuestas.

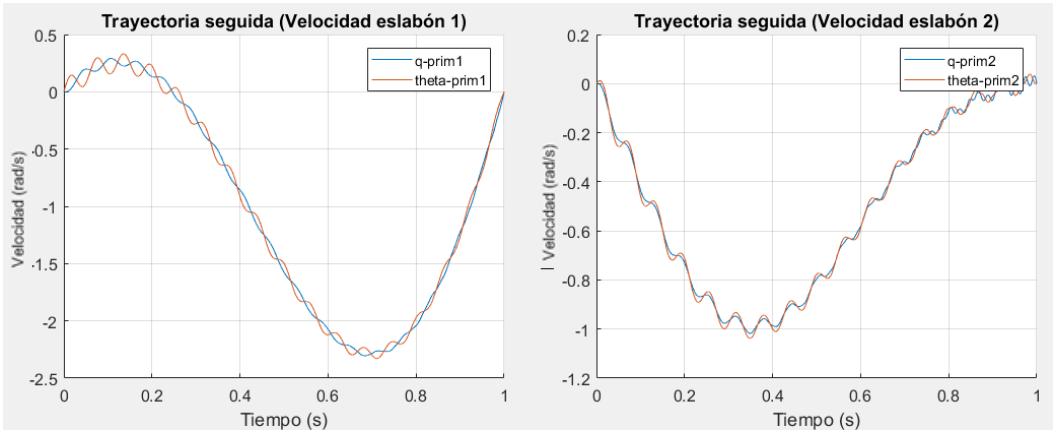


Figura 36. Experimento 1: trayectoria seguida por los eslabones de la velocidad (iLQR).

A continuación, se muestra una gráfica con representación de la velocidad frente a la posición. Es muy útil para analizar como varían los estados a lo largo del tiempo. Además, se puede deducir si el movimiento es fluido o presenta cambios bruscos, de la figura 37 podemos afirmar que la trayectoria o el movimiento seguido es bastante suave.

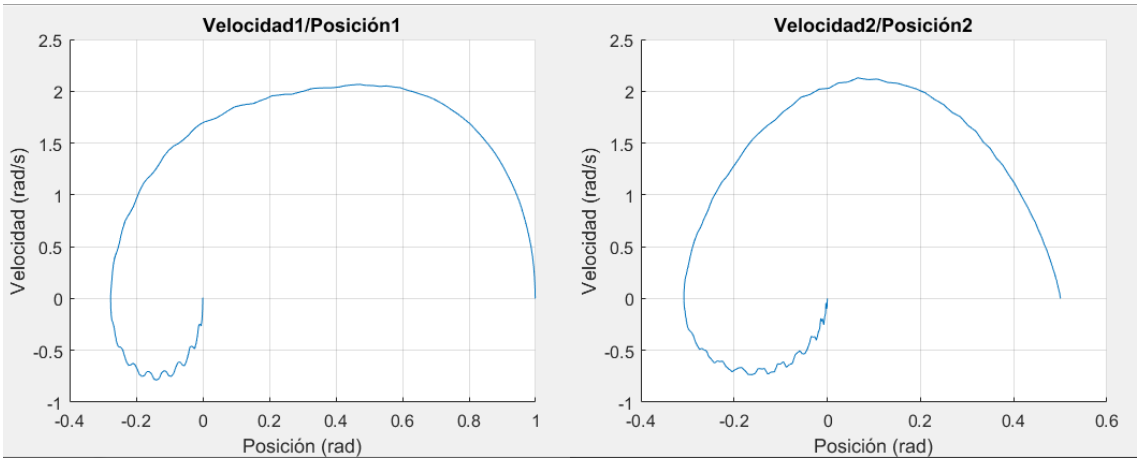


Figura 37. Experimento 1: evolución de la posición frente a la velocidad (iLQR)

Finalmente tenemos la acción de control que se muestra en la figura 37. Ambas acciones de control tienen forma más bien parabólica, alcanzando un par máximo de aproximadamente 27.2 Nm para el caso del primer eslabón y 7.12 Nm para el segundo.

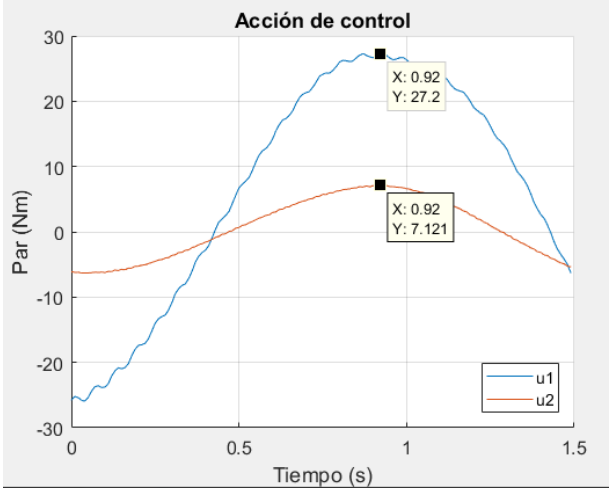


Figura 38. Experimento 1: acción de control (iLQR)

Veamos ahora un ejemplo, cuando Q es distinto de cero. Así pues, los valores escogidos para el segundo experimento son los empleados anteriormente más la nueva Q:

```
Q = diag([1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0]);
```

Código 16. Experimento 2: Q distinto de 0.

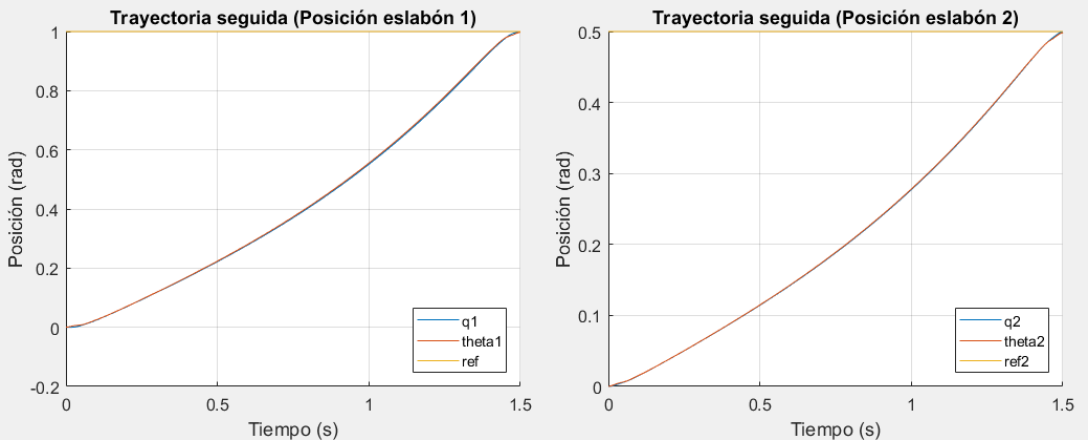


Figura 39. Experimento 2: trayectoria seguida por los eslabones de la posición (iLQR).

Podemos notar que el comportamiento es bastante diferente al del experimento 1, es un poco más agresivo, es decir, desde el primer instante la posición empieza a aumentar, con una respuesta muy similar a rampa.

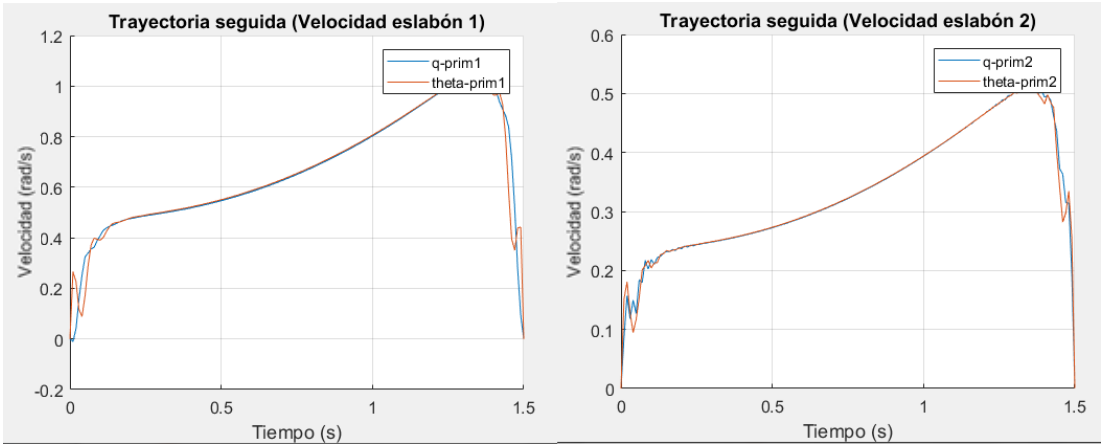


Figura 40. Experimento 2: trayectoria seguida por los eslabones de la velocidad (iLQR).

En la figura 40 se ve claramente como la velocidad aumenta progresivamente, pero en los últimos instantes se frena de forma excesiva, esto también se refleja en la acción de control.

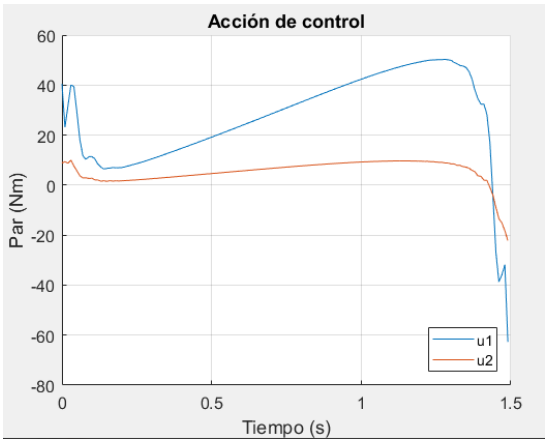


Figura 41. Experimento 2: acción de control.

También, resulta interesante analizar los pares máximos alcanzados, comparando ambos experimentos vemos que las exigencias en el segundo requieren más par en algunos intervalos de tiempo, pudiendo alcanzar pares superiores a 60 Nm para el caso del primer motor. En cambio, en el experimento 1, el par máximo variaba entre -30 y 30 Nm para el motor 1.

Por otro lado, no podemos comparar de forma directa las soluciones obtenidas del iLQR con las que obtuvimos con el regulador PD y LQR, ya que, las condiciones son distintas, aquí no estamos realizando un seguimiento de trayectoria. Tanto en el PD como en el LQR empleamos una trayectoria trapezoidal como referencia. Sin embargo, para el caso del LQR, cuando la Q la igualábamos a un valor muy próximo a cero, y es cuando prácticamente no se realizaba seguimiento de trayectoria; se puede observar una similitud entre las respuestas obtenidas con las de este apartado.

Asimismo, cabe destacar que la implementación de este control es muy costosa y requiere mucho cálculo, de tal punto que hasta las simulaciones realizadas en un PC podrían llegar a tardar minutos dependiendo del periodo de muestreo y tiempo de simulación. Utilizando la función de Matlab c2d el proceso de discretización es relativamente lento, se ha conseguido acelerar empleando la ecuación 2.47 que se refleja en el código 1, sin embargo, para periodos de muestro bajos es necesario mayor número de iteraciones, por lo tanto, sigue siendo algo costoso. Otro factor que influye es el número de variables de estado, con más de cinco el proceso se ralentiza bastante.

Implementar este control en un microcontrolador lo más probable es que no sea muy viable, aparte de que sería muy complicado. Por lo tanto, se debe optar por un dispositivo con más potencial, por ejemplo, un mini pc como es la Raspberry Pi.

Para concluir este apartado, podemos decir que considerando que el iLQR es un control en el que se tiene en cuenta la dinámica variante, se pueden resolver muchos de los problemas no lineales. Además, si probásemos este control en la planta real, los resultados deberían de ser bastante buenos y en comparación con los resultados simulados bastante realistas. Evidentemente, es necesario que el modelo también sea bueno. De hecho, si el modelo es exacto, podríamos coger directamente la acción de control obtenida de las simulaciones y aplicarla en bucle abierto, el resultado sería la trayectoria obtenida de la simulación.

CAPÍTULO 5: DISEÑO DEL SOFTWARE

5.1. Introducción

En este apartado hablaremos de la parte de programación, para ello es importante conocer el entorno de programación y las limitaciones del microcontrolador con el cual se esté trabajando. En efecto, para este proyecto se han utilizado dos dispositivos, por una parte, la STM32F4 en la que emplearemos el lenguaje de programación C, y por otra, una Raspberry PI 3 que dispone de sistema operativo Linux en el que se desarrollará una aplicación en Python. De modo que, en el primer microcontrolador se implementará el control PD con compensación de gravedad online y en el segundo, que no es un microcontrolador, el LQR con seguimiento de trayectoria. La razón principal de implementar el LQR en la Raspberry es la facilidad de trabajar con matrices en la programación de Python. Además, al disponer de sistema operativo se pueden desarrollar multitud de aplicaciones. Sin embargo, por esta razón, es necesario realizar una comunicación entre ambos micros ya que, la STM32F4 será la principal y se encargará de obtendrán todas las lecturas de sensores y aplicará las acciones de control.

5.2. Microcontrolador STM32F4

La serie STM32F4 de unidades de microcontroladores da un alto rendimiento con instrucciones de procesador de señal digital (DSP) y unidad de punto flotante (FPU). Está basada en ARM® Cortex®-M4, cuenta con la tecnología NVM de STMicroelectronics y el ART Accelerator™ de ST para alcanzar los resultados de referencia más altos de la industria para microcontroladores basados en Cortex-M.

5.2.1. Configuración de puertos y pines

Para que el microcontrolador pueda interactuar con el mundo externo, se dotan de puertos o pines que se conectan a otros sistemas o componentes diversos. Muchos de los pines de entrada/salida están multiplexados con funciones alternativas.

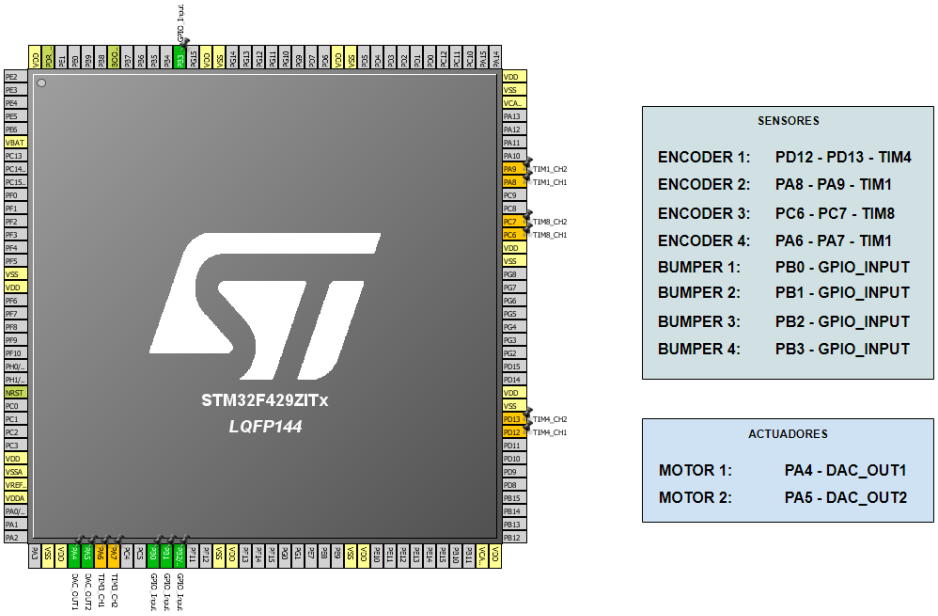


Figura 42. Asignación de pines.

En la figura 39 se muestran los pines principales utilizados, de los cuales, hay 2 pines de salida y 16 de entrada. Los dos pines de salida están configurados en modo DAC, es decir están multiplexados con el chip DAC que tiene incorporado el microcontrolador, esto nos permite generar la señal para la acción de control. En cuanto a los pines de entrada, por una parte, se han configurado 4 *timers* en modo *encoder* y de cada *timer* se utilizan dos canales para la lectura del *encoder*. Por otro lado 4 pines genéricos en modo entrada con interrupciones habilitadas para los 4 finales de carrera empleados para el *homing* y la parada controlada.

5.2.2. Configuración de encoders

El primer paso sería crear las estructuras con las que vayamos a trabajar. Estas estructuras están predefinidas e incorporan todos los parámetros que se pueden aplicar a la nueva estructura. Por ejemplo, para la estructura `GPIO_InitTypeDef`, los parámetros serían los pines que se desean asignar, el puerto, el modo (ya sea entrada, salida, modo alternativo), velocidad del reloj, etc.

```
/* Definir estructuras */
GPIO_InitTypeDef GPIO_InitStructure;
TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
```

Código 17. Creación de estructuras.

El segundo paso sería, habilitar el reloj del periférico APB1 y del periférico AHB1. Este paso es muy importante, equivale a darle vida al puerto con el que se esté trabajando. En este caso, el puerto GPIOD y el TIM4.

```
/* Inicializamos reloj del puerto GPIOC */
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
/* Inicializamos reloj del TIM3 */
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);
```

Código 18. Habilitación del reloj.

El tercer paso sería asignar la configuración de los pines accediendo a las opciones disponibles de la estructura. Entre ellas, los pines que se vayan a emplear, el modo en este caso alternativo, la velocidad del bus, si hay resistencia de *pull-up* o *pull-down*. Finalmente aplicamos la estructura al puerto deseado.

```
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
// aplicamos la configuración.
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13;
GPIO_Init(GPIOD, &GPIO_InitStructure);
```

Código 19. Configuración de los pines.

En el siguiente paso multiplexamos los pines PD12, PD13 con el TIM4.

```
//Timer AF Pins Configuration
GPIO_PinAFConfig(GPIOD, GPIO_PinSource12, GPIO_AF_TIM4);
GPIO_PinAFConfig(GPIOD, GPIO_PinSource13, GPIO_AF_TIM4);
```

Código 20. Asignación de modo alternativo.

Los pasos anteriores servían para configurar los pines de un puerto y enlazarlos con el timer 3, ahora bien, también hay que especificar la configuración para el *timer*. Se tiene que indicar el periodo después del cual se produce la auto recarga, el *prescaler* para determinar la velocidad o frecuencia, e indicar si las cuentas irán incrementando o decrementando.

```
TIM_TimeBaseStructure.TIM_Period = 0xffff;
TIM_TimeBaseStructure.TIM_Prescaler = 0;
TIM_TimeBaseStructure.TIM_ClockDivision = 0;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;

TIM_TimeBaseInit(TIM4, &TIM_TimeBaseStructure);
```

Código 21. Definición de la estructura del timer.

A continuación, se especifica que el timer se utilizará en modo encoder, se indica el valor al que se actualizan las cuentas cuando se produce la auto-recarga y por último se habilita el timer.

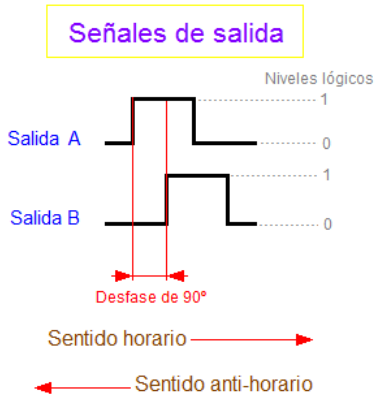
```
TIM_EncoderInterfaceConfig (TIM4, TIM_EncoderMode_TI12,
TIM_ICPolarity_Rising, TIM_ICPolarity_Rising);
TIM_SetAutoreload (TIM4, 0xffff);

TIM_Cmd (TIM4, ENABLE);
```

Código 22. Configuración en modo encoder.

Seguidos todos los pasos anteriores habremos configurado con éxito los pines PD12 y PD13 para obtener lecturas de un encoder. Para el caso de un brazo robot con dos articulaciones elásticas se necesitan 4 encoders, 2 por cada articulación. Uno antes del efecto de elasticidad y el otro después del efecto de elasticidad. De modo que uno irá colocado directamente sobre el rotor y el otro después del mecanismo que produce la elasticidad que en este caso es el *harmonic drive*. El procedimiento para configurar los otros tres encoders es muy similar, por lo tanto, no merece la pena describir el proceso de nuevo. Sin embargo, hay que recalcar que para la configuración se emplearan otros *timers* con sus correspondientes pines.

5.2.3. Interpretación de la información de los encoders

Es muy importante interpretar bien la información de los encoders. Para realizar el control es necesario conocer la posición y velocidad angular en cada iteración. La posición la podemos medir directamente ya que se han empleado encoders incrementales.



Los encoders incrementales tienen 2 salidas básicamente que dan las dos ondas, en teoría cuadradas, desfasadas 90 grados cuando hay movimiento. Los encoders industriales muchas veces tienen (sin contar las de alimentación si las tuviera y otras características adicionales) una tercera salida llamada "*Index*" que sirve para indicar que se ha dado una vuelta completa. Para obtener la posición angular hay que pasar de pulso por revolución a radianes.

Los encoders empleados dan 2000 pulsos por revolución de modo que si establecemos una regla de tres podríamos obtener el resultado en radianes. Otro aspecto importante a tener en cuenta es que los timer del microcontrolador son de 16 bits de modo que cuando se produce la auto-recarga esta se debe tener en cuenta. En el siguiente código se muestran las condiciones y la forma empleada para obtener bien la posición angular.

```
float PPR_to_Rad(int32_t encoder, int32_t encoder_previous){
	float aux;

	if(encoder-encoder_previous > 65500){
		aux = ((float)(encoder - 65536)*2*Pi)/PPV_Encoder;
		return aux;
	}

	else if(encoder-encoder_previous < -65500){
		aux = ((float)(encoder + 65536)*2*Pi)/PPV_Encoder;
		return aux;

	}
	else{
		return (float)((encoder*2.0*Pi)/PPV_Encoder);
	}
}
```

Código 23. Función para pasar de pulsos por revolución a radianes.

Ejemplo de uso.

```
// Lectura de la posición del encoder en pulsos por revolución
encoder1.posEncoder_PPR = TIM_GetCounter(TIM4);

// Conversión de la posición de PPR a radianes
encoder1.posEncoder_rad = PPR_to_Rad(encoder1.posEncoder_PPR,
encoder1.previous_posEncoder_PPR);
```

Código 24. Ejemplo de uso de la función PPR_to_Rad().

Por otro lado, para calcular la velocidad se necesitan dos lecturas de la posición, es decir para el instante de t y $t-1$. De modo que conocido el tiempo de muestreo es fácil de obtener. Sin embargo, es muy importante tener en cuenta que la presencia de ruido al realizar las lecturas de posición podría resultar en unas medidas incorrectas y con ello fastidiar el control. Para solucionar este problema se podría implementar un filtro paso bajo o una solución mejor podría ser la implementación de un filtro de Kalman. A continuación, se muestra la función creada para obtener la velocidad angular.

```
float speed_estimation(float pos_actual, float pos_anterior, float
tiempo){

      return ((pos_actual-pos_anterior) / tiempo);
}
```

Código 25. Función para obtener la velocidad angular.

5.2.4. Diseño del controlador PD

En el capítulo 3 vimos la implementación del PD con y sin compensación de gravedad. De modo que simplemente tendremos que implementar una función que calcule la acción proporcional-derivativa. A continuación, se muestra el código implementado.

```
float PD_action(float pos, float vel, float pos_ref){
      float pos_error, pos_error_last, acc_P, acc_D, PD;
      // Calculamos el error //
      pos_error = pos_ref - pos;
      // Calculamos la acción del proporcional ///
      acc_P = pos_error * Kp;
      // Calculamos la acción derivativa //
      acc_D = vel * Kd;
      // Obtenemos la acción del PI //
      PD = acc_P - acc_D;
      // Nos aseguramos de que no sobrepase un límite (SATURACIÓN)//
      if(PD>PD_max)PD = PD_max;
      else if(PD<(PD_min))PD = PD_min;
      // Almaenamos el error atual
      pos_error_last = pos_error;

      return PD;
}
```

Código 26. Función para calcular la acción de control del PD.

Observe que los parámetros que debemos pasar son la lectura del encoder de la posición angular del lado del motor, la posición de referencia y la velocidad angular estimada del lado del motor.

Por otro lado, tenemos el controlador PD con compensación de gravedad. Para implementar este controlador necesitamos extraer la gravedad a partir de la dinámica de nuestro sistema. De ahí se obtienen dos ecuaciones, una para el eslabón 1 otra para el eslabón 2. Dependiendo de la posición de cada eslabón la fuerza debida a la gravedad varía para cada articulación. Así mismo, no olvidar que el primer eslabón lleva también la gravedad del segundo.

Por lo tanto, lo que hay que hacer es, utilizar el controlador PD normal para calcular la acción de control proporcional y la derivativa y después sumar el valor variable de la gravedad. Para cada eslabón será distinto el valor obtenido de modo que tendremos dos fórmulas. A continuación, se muestra como se ha implementado:

```
if(t<=my_trajectory1.size){
      pos_ref1 = my_trajectory1.position[t];
      g1 = (myrobot.m1*g*myrobot.d1+myrobot.mr2*g*myrobot.L1)
      *cos(pos_ref1)+myrobot.m2*g*(myrobot.L1*cos(pos_ref1)-
      myrobot.d1*(sin(pos_ref1)*sin(pos_ref1)-
      cos(pos_ref1)*cos(pos_ref1)));
}

if(t<=my_trajectory2.size){
      pos_ref2 = my_trajectory2.position[t];
      g2 = myrobot.m2*g*myrobot.d1*(sin(pos_ref2)* sin(pos_ref2)-
      cos(pos_ref2)*cos(pos_ref2));
}

u1 = PD_action(encoder3.posEncoder_rad,
encoder3.previous_velEncoder_rad ,pos_ref1) + g1;

u2 = PD_action(encoder4.posEncoder_rad,
encoder4.previous_velEncoder_rad, pos_ref2) + g2;
```

Código 27. Acción de control obtenida del PD con compensación de gravedad.

Nótese que los parámetros que necesitamos son las dos posiciones angulares, las dos posiciones de referencia y las dos velocidades angulares. Además, la estructura del robot que incluye los pesos de cada eslabón, así como, las longitudes del eslabón y de los centros de masas. La acción de control resultante es la acción del PD más la acción debida a la fuerza de gravedad.

Tanto el PD básico como el PD con compensación de gravedad tienen limitada la acción de control máxima y mínima para evitar que se produzcan acciones de control demasiado altas de acuerdo con las especificaciones de los circuitos con los que se esté trabajando, que en caso contrario podrían dañar algo.

5.2.5. Diseño del bucle de control

El bucle de control es equivalente al programa principal, se encarga de ejecuta repetidamente el código que tiene entre llaves hasta que termine el programa, alguna condición deje de cumplirse, u otro código de mayor prioridad empiece a ejecutarse (por ejemplo, las interrupciones).

El bucle de control normalmente debe ser periódico, es decir, tiene asignado un periodo. Para ello se ha tenido que configurar un timer que cada cierto tiempo produce una interrupción. Los pasos seguidos se detallan a continuación, por la existencia de similitudes con la configuración que empleamos para configurar los encoders lo describiremos de forma resumida.

Centrándonos en la obtención del periodo y del valor del *prescaler*. Conocida la frecuencia del reloj que le llega al timer y, estableciendo el valor del *prescaler*, el periodo tiene que ser un valor o número de cuentas que a la frecuencia que incrementan las cuentas (limitada por el *prescaler*) sea equivalente al tiempo deseado. Por ejemplo, con el valor del *prescaler* igual a 420, las cuentas incrementarían cada $10\ us$, ahora bien, supongamos que la frecuencia que queremos es de 1000 Hz (1ms) el periodo obtenido serían 100 cuentas. El resultado es $100 \cdot 10E^{-6} = 1\ ms$

```
ARR = 84000000.0F / ((840+1)*frequency) -1;
period = (int) ARR;
if(ARR != period){
      period++;
}// +1 en caso de numero fraccionario

PrescalerValue =420;          /(SysClk / 2)/420 = 10 us/cuenta

TIM_TimeBaseStructure.TIM_Period = period;
TIM_TimeBaseStructure.TIM_Prescaler = PrescalerValue;
TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);

TIM_Cmd(TIM2, ENABLE);
```

Código 28. Configuración del periodo del bucle de control.

Una de las características clave del microcontrolador STM32 es el sistema de interrupciones. El Cortex-M4 tiene un sofisticado sistema de interrupción llamado *Nested Vectored Interrupt Controller* (NVIC). Para configurar las interrupciones indicamos el canal (`TIM2_IRQn`) y asignamos las prioridades.

```
// Enable the TIM2 gloabal Interrupt
NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;

NVIC_Init(&NVIC_InitStructure);
```

Código 29. Configuración de interrupciones

Cuando se produce una interrupción, al tener una prioridad mayor, básicamente la ejecución del programa principal se interrumpe para atender la interrupción que en este caso es la que produce `TIM2_IRQn`. Para habilitar o deshabilitar las interrupciones del reseteo de las cuentas, simplemente tenemos que ejecutar la siguiente función indicando si se desea habilitar o deshabilitar.

```
TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
TIM_ITConfig(TIM2, TIM_IT_Update, DISABLE);
```

Código 30. Habilitar o deshabilitar las interrupciones.

Cuando se producen las interrupciones, estas son atendidas por su correspondiente manejador, para el caso del timer 2 sería el `TIM2_IRQHandler`. De modo que, cuando se produce la auto-recarga se ejecuta el siguiente código.

```
void TIM2_IRQHandler(void)
{
       //Se produce auto-reload
       if (TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET)
       {
             flag_Exp = 1;

             //Clear interruption flag
             TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
       }
 }
```

Código 31. Manejador de la interrupción.

Volviendo al bucle de control, las tareas que realiza son básicamente lectura de sensores, calcular una acción de control y aplicar dicha acción de control. La estructura es la siguiente:

```
Control_TimerInit(1000);  //  Bucle de control de 1 ms

TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);

for(i = 0; i <= duration; i++){
      flag_Exp = 0;
      tiempo = t*tsampling;

      // Lectura de los sensores
      // Tratamiento de la información
      // Cálculo de la acción de control para el control indicado
      // Aplicar la acción de control a los actuadores

      while(flag_Exp != 1);

      t++;
}
TIM_ITConfig(TIM2, TIM_IT_Update, DISABLE);
```

Código 32. Bucle de control

La condición de *while(flag_Exp ¡=1)* hace que el bucle de control sea periódico. Cuando el timer produce una interrupción (cada 1 ms) el *flag* cambia a 1 y se produce la salida del bucle *while*. Es muy importante que todo lo que se tenga anteriormente dentro del bucle for, ya sean operaciones, asignación de variables o llamadas a otras funciones, todo ello tiene que realizarse en un tiempo inferior o igual al periodo establecido. Si no se cumple esta condición el bucle de control no sería periódico.

5.2.6. Comunicación SPI con la Raspberry Pi

El Bus SPI (del inglés *Serial Peripheral Interface*) es un estándar de comunicaciones, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos. El bus de interfaz de periféricos serie o bus SPI es un estándar para controlar casi cualquier dispositivo electrónico digital que acepte un flujo de bits serie regulado por un reloj (comunicación sincrónica).

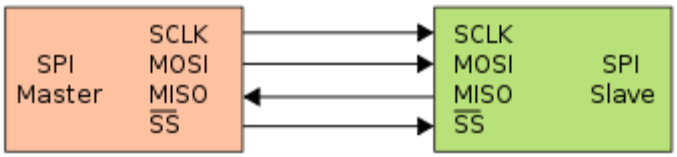


Figura 43. Comunicación SPI entre un maestro y un esclavo.

El SPI es un protocolo síncrono. La sincronización y la transmisión de datos se realiza por medio de 4 señales. Incluye una línea de reloj, dato entrante, dato saliente y un pin de chip *select*, que conecta o desconecta la operación del dispositivo con el que uno desea comunicarse. De esta forma, este estándar permite multiplexar las líneas de reloj.

A la hora de transferir la información de un dispositivo a otro, emplearemos el acceso directo a memoria (DMA) con el fin de no emplear la CPU, de modo que, la transferencia se lleva a cabo por el controlador DMA. Las transferencias DMA son esenciales para aumentar el rendimiento de aplicaciones que requieran muchos recursos.

La STM32F4 cuenta con tres SPIs en modos esclavo y maestro y con modos de comunicación *full-duplex* y *simplex*. A la hora de elegir cual utilizar, debemos tener en cuenta las configuraciones de pines hasta ahora realizadas. De modo que, finalmente se ha decidido emplear el SPI2 cuyos pines son: PB13 para la línea de reloj, PB14 para los datos entrantes y PB15 para los datos salientes, siendo este dispositivo el esclavo.

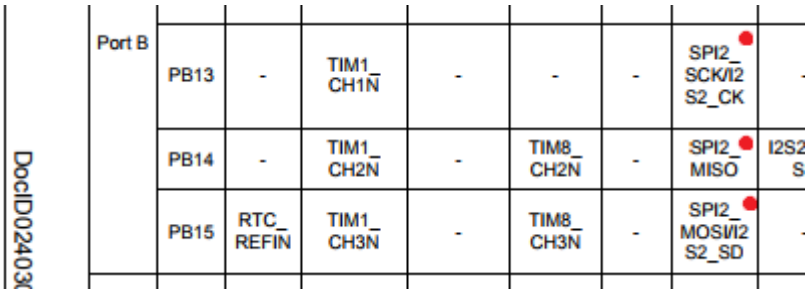


Figura 44. Datasheet: tabla de los pines del SPI2.

Sabiendo que vamos a utilizar el SPI2, el siguiente paso sería determinar el DMA que vamos a utilizar. Pues resulta que es el DMA1 y los canales correspondientes 4 y 5, esto se puede observar en la figura 45 donde aparece la tabla con los periféricos que pueden emplear la memoria de acceso directo con los canales correspondientes.

Peripherals	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
ADC1	ADC1						
SPI/I²S		SPI1_RX	SPI1_TX	SPI2/I2S2_RX	SPI2/I2S2_TX		
USART		USART3_TX	USART3_RX	USART1_TX	USART1_RX	USART2_RX	USART2_TX
I²C				I2C2_TX	I2C2_RX	I2C1_TX	I2C1_RX
TIM1		TIM1_CH1	TIM1_CH2	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	
TIM2	TIM2_CH3	TIM2_UP			TIM2_CH1		TIM2_CH2 TIM2_CH4
TIM3		TIM3_CH3	TIM3_CH4 TIM3_UP			TIM3_CH1 TIM3_TRIG	
TIM4	TIM4_CH1			TIM4_CH2	TIM4_CH3		TIM4_UP

Figura 45. Resumen de los canales del DMA1.

Para la configuración como siempre empezamos habilitando el reloj para los periféricos, en este caso el puerto B y la DMA1:

```
/* Enable GPIO clocks */
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB | RCC_AHB1Periph_GPIOB
| RCC_AHB1Periph_GPIOB, ENABLE);

  /* Enable DMA clock */
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA1, ENABLE);
```

Código 33. Habilitación del reloj de los periféricos.

A continuación, indicamos los pines del puerto correspondiente que se emplearan para la comunicación SPI, así como otras configuraciones.

```
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13|GPIO_Pin_14|GPIO_Pin_15;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_DOWN;
  GPIO_Init(GPIOB, &GPIO_InitStructure);
```

Código 34. Configuración de los pines de la comunicación SPI.

Para ello también, los pines se deben configuran en modo alternativo, indicando que se trata de SPI.

```
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource13, GPIO_AF_SPI2);
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource14, GPIO_AF_SPI2);
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource15, GPIO_AF_SPI2);
```

Código 35. Modo alternativo (SPI).

Una vez configurados los pines, paso seguido, debe configurar la comunicación SPI. Entre las posibles opciones, podemos indicar si se trata de comunicación *full-duplex*, lo que viene siendo bidireccional, o *simplex*, en una dirección en la que una estación siempre actúa como fuente y la otra siempre como receptor. También existe la *half-duplex* es una conexión en la que los datos fluyen en una u otra dirección, pero no las dos al mismo tiempo. Otra posible opción es indicar el tamaño del dato que se envía (8 o 16 bits). La polaridad del reloj, el orden de los datos, es decir, si primero se envían los bits más significativos MSB o lo menos significativos LSB.

Para determinar la velocidad de la comunicación que limitaremos con un *prescaler*, debemos conocer la velocidad del reloj del periférico SPI. Todo esto, en forma de código se realiza de la siguiente manera:

```
SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;
SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low;
SPI_InitStructure.SPI_CPHA = SPI_CPHA_1Edge;
SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;
SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_8;
SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;
SPI_InitStructure.SPI_CRCPolynomial = 7;
SPI_InitStructure.SPI_Mode = SPI_Mode_Slave;
SPI_Init(SPI2, &SPI_InitStructure);
```

Código 36. Configuración de la comunicación SPI.

El siguiente paso es configurar la DMA. Para ello necesitaremos tener dos canales DMA habilitados al mismo tiempo: uno para recibir bytes de SPI y almacenarlos en un buffer de memoria, y uno para enviar bytes de un segundo *buffer* a SPI.

Como ya comentamos, el dispositivo STM32F4 incorporan dos controladores DMA, ofreciendo hasta 16 *streams* en total (ocho por controlador), cada uno dedicado a gestionar solicitudes de acceso a memoria de uno o más periféricos.

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	SPI3_RX	-	SPI3_RX	SPI2_RX	SPI2_TX	SPI3_TX	-	SPI3_TX

Figura 46. Configuración disponible de los flujos / canales DMA frente a peticiones periféricas.

De la tabla 46, vemos que el canal que debemos utilizar para el caso del SPI2 es el cero y los *streams* de recepción y transmisión son tres y cuatro respectivamente. También existe la posibilidad de utilizar el stream 5 para la transmisión, pero no es el caso. En el código 47 se muestran todos los pasos seguidos para la correcta configuración de la STM32F4, no obstante, no se han describirán todos los pasos.

Observe que se utiliza un buffer normal, el puntero de escritura apunta a la siguiente ubicación disponible del búfer que es donde se escribirá. Se incrementa cuando los datos se colocan en el búfer. El puntero de lectura apunta a la siguiente ubicación del búfer a leer. Se incrementa cuando se extraen datos del búfer.

```
    /* Configure DMA Initialization Structure */
  DMA_InitStructure.DMA_BufferSize = BUFFERSIZE ;
  DMA_InitStructure.DMA_FIFOMode = DMA_FIFOMode_Disable ;
  DMA_InitStructure.DMA_FIFOThreshold =
DMA_FIFOThreshold_1QuarterFull ;
  DMA_InitStructure.DMA_MemoryBurst = DMA_MemoryBurst_Single ;
  DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Byte;
  DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
  DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
  DMA_InitStructure.DMA_PeripheralBaseAddr =(uint32_t)(&(SPI2->DR));
  DMA_InitStructure.DMA_PeripheralBurst =
DMA_PeripheralBurst_Single;
  DMA_InitStructure.DMA_PeripheralDataSize =
DMA_PeripheralDataSize_Byte;
  DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
  DMA_InitStructure.DMA_Priority = DMA_Priority_High;
  /* Configure TX DMA */
  DMA_InitStructure.DMA_Channel_0 = DMA_Channel_0 ;
  DMA_InitStructure.DMA_DIR = DMA_DIR_MemoryToPeripheral ;
  DMA_InitStructure.DMA_Memory0BaseAddr =(uint32_t)SPI_DataTx ;
  DMA_Init(DMA1_Stream4, &DMA_InitStructure);
  /* Configure RX DMA */
  DMA_InitStructure.DMA_Channel_0 = DMA_Channel_0 ;
  DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralToMemory ;
  DMA_InitStructure.DMA_Memory0BaseAddr =(uint32_t)SPI_DataRx ;
  DMA_Init(DMA1_Stream3, &DMA_InitStructure);
```

Código 37. Configuración de la DMA.

El último paso es habilitar la comunicación SPI, el DMA1 y el acceso directo a memoria del periférico.

```
  /* Enable DMA SPI TX Stream */
  DMA_Cmd(DMA1_Stream4,ENABLE);

  /* Enable DMA SPI RX Stream */
  DMA_Cmd(DMA1_Stream3,ENABLE);

  /* Enable SPI DMA TX Requsts */
  SPI_I2S_DMACmd(SPI2, SPI_I2S_DMAReq_Tx, ENABLE);

  /* Enable SPI DMA RX Requsts */
  SPI_I2S_DMACmd(SPI2, SPI_I2S_DMAReq_Rx, ENABLE);

  /* Enable the SPI peripheral */
  SPI_Cmd(SPI2, ENABLE);
```

Código 38. Habilitación SPI y acceso directo a la memoria.

Para terminar, nótese que hay dos variables DataRx y DataTx, el primero contiene los datos recibidos y el otro los datos que se van a enviar.

5.2.7. Interpretación de los datos recibidos (SPI)

Los datos que llegan de la Raspberry Pi se almacenan en el vector SPI_DataRx[BUFFERSIZE] que es de tipo uint8_t. Tal y como veremos en el apartado 5.3.2 los datos que enviaremos de la Raspberry son de tipo *float* convertidos a *signed int* de 32 bits y posteriormente divididos en cuatro de 8 bits. Esto significa que, cada cuatro datos que recibimos se corresponden con el valor de un parámetro. Para ello se ha implementado la siguiente función.

```
float convertFour8bitToFloat(uint8_t a, uint8_t b, uint8_t c,
uint8_t d){
      int32_t aux;

      aux = ((a << 24) | (b << 16) | (c << 8) | (d & 0x000000ff));

      return aux/1000.0;
}
```

Código 39. Función para convertir cuatro 8 bits a float.

El siguiente paso sería interpretar todos los datos transmitidos. Los datos son las ganancias K y Kv y la secuencia v. Estos datos son necesarios para implementar el algoritmo LQR tal y como vimos en su implementación. Este proceso se describe en la función `void spi_receive(LQR_t *lqr)`. También se utiliza el manejador EXTI3_IRQHandler que es el que determina cuando se han transmitido los datos de cada trama y pueden ser leídos.

```
for(z=0;z<spi_iter;z++){
  // Espera de recepción de datos (CS interupt)
  for(i=0;i<25;i++){
    for(k=0;k<8;k++){
      if (spi_max_iter<=((25*z)+i)){  // si se alcanza tamaño max.
        break;
      }

      lqr->K[0][k][(25*z)+i] = convertFour8bitToFloat(
      SPI_DataRx[0+(20*k)+(160*i)+offset+(z*4000)],
      SPI_DataRx[1+(20*k)+(160*i)+offset+(z*4000)],
      SPI_DataRx[2+(20*k)+(160*i)+offset+(z*4000)],
      SPI_DataRx[3+(20*k)+(160*i)+offset+(z*4000)]);

      lqr->K[1][k][(25*z)+i] = convertFour8bitToFloat(
      SPI_DataRx[4+(20*k)+(160*i)+offset+(z*4000)],
      SPI_DataRx[5+(20*k)+(160*i)+offset+(z*4000)],
      SPI_DataRx[6+(20*k)+(160*i)+offset+(z*4000)],
      SPI_DataRx[7+(20*k)+(160*i)+offset+(z*4000)]);

      // Lo mismo para lqr->Kv[0][k][(25*z)+i],
      lqr->Kv[1][k][(25*z)+i] y lqr->v[k][(25*z)+i]
    }
  }
}
```

Código 40. Interpretación de los datos recibidos (SPI).

5.2.8. Parada controlada

La parada controlada como su nombre indica trata de la parada del movimiento de la máquina, en este caso de los dos eslabones, por ejemplo, mediante la reducción de la señal de mando hasta cero desde que la orden de parada ha sido reconocida por el dispositivo de mando. Pero manteniendo la energía en los accionadores durante el proceso de parada. Consiguientemente para que la parada sea controlada se ha optado por llevar la velocidad de los motores a cero empleando un regulador PI. Donde la referencia de velocidad es generada por la siguiente función.

```
float vel_reference(float vel_alcanzada, float acel, float dt){

	if(vel_alcanzada>V_offset){
		vel_alcanzada = vel_alcanzada - acel*dt;
	}
	if(vel_alcanzada<=V_offset && vel_alcanzada>=-V_offset){
		vel_alcanzada = 0;
	}
	if(vel_alcanzada<-V_offset){
		vel_alcanzada = vel_alcanzada + acel*dt;
	}
	return vel_alcanzada;
}
```

Código 41. Generación de la referencia de velocidad.

Como se puede observar la velocidad aumenta o disminuye en función de la aceleración o desaceleración deseada que pasamos como parámetro a la función. También se ha establecido un rango o zona muerta en la que la velocidad simplemente es nula.

La forma de implementar el controlador PI es muy parecida a la del PD solo cambia la acción derivativa por la acción integrativa y la posición angular por la velocidad angular.

```
float PI_action(float vel_encoder, float vel_ref, float dt){

	/* Calculamos el error */
	vel_error = vel_ref - vel_encoder;
	/* Calculamos la acción del proporcional */
	vel_P = vel_error * Kp;
	/* Calculamos la acción integrativa */
	vel_I = vel_error_last + (vel_error * Ki * dt);
	/* Obtenemos la acción del PI */
	PI = vel_P  + vel_I;
	/* Nos aseguramos de que no sobrepase un límite (SATURACIÓN)*/
	if(PI>PI_max)PI = PI_max;
	else if(PI<(PI_min))PI = PI_min;

	/* Almaenamos el error atual, que utilizaremos en el siguiente
cálculo */
	vel_error_last = vel_error;

	return PI;
}
```

Código 42. Controlador PI.

Quedaría definir cuándo y cómo se produce la parada controlada, esto ocurre cuando alguno de los finales de carrera es presionado.

Cuando el pulsador es accionado, el código principal se ve interrumpido para atender la interrupción producida con el manejador correspondiente. Por ejemplo, el pin PB0 tiene el manejador `EXTI0_IRQHandler` que es el que se muestra a continuación. Es importante señalar que todos los pines 0 de cualquier puerto, emplean el mismo manejador, esto es un dato importante a tener en cuenta a la hora de diseñar el programa.

```
void EXTI1_IRQHandler(void) {
    if (EXTI_GetITStatus(EXTI_Line1) != RESET) {

        if(homing.state==1){
                homing.B1=1;
        }
        else{
                paradacontrolada();
        }
        /* Clear interrupt flag */
        EXTI_ClearITPendingBit(EXTI_Line1);
    }
}
```

Código 43. Manejador de interrupciones EXTI0_IRQHandler.

Así pues, en concreto, lo que sucede cuando se produce la interrupción, es la llamada a la función `paradacontrolada()`, esta a su vez implementa un bucle de control muy parecido al bucle principal que vimos anteriormente, con todas las tareas de lectura de sensores, estimación de velocidades y cálculo de la acción de control. Sólo que ahora la idea es controlar la velocidad llevándola a cero radianes por segundo, con esto obviamente se consigue inmovilizar al brazo. Observe que existe una condición en la que no es posible realizar la parada controlada. Esto ocurre cuando se realiza el homing, en este caso, la velocidad angular a la que se mueven los eslabones es relativamente lenta por lo tanto se podría asumir que parar directamente los motores no ocasionaría ningún problema.

A continuación, se muestra parte del código dónde se muestra como se genera la señal de la acción de control.

```
/ Eslabón 1
DAC->DHR12R1 = escalado(PI_action(encoder1->velEncoder_rad, V1_ref,
tsampling));

/ Eslabón 2
DAC->DHR12R2 = escalado(PI_action(encoder2->velEncoder_rad, V2_ref,
tsampling));
```

Código 44. Acción de control PI para la parada controlada.

Los parámetros que debemos pasar son la velocidad angular en el estado actual y la velocidad de referencia generada (rampa decreciente), así como el tiempo de muestreo.

5.2.9. Homing

El proceso de *homing* consiste en llevar al robot a su posición inicial, para ello se debe iniciar un proceso en el que los eslabones se muevan hasta llegar a esa posición a una velocidad relativamente lenta, sabemos que esto sucede cuando los finales de carrera son presionados y se produce una interrupción. En ese instante se toman lecturas de los encoders y las posiciones servirán de referencia, equivalentes al *offset* que hay que restar. El proceso se realizará por separados, es decir empezando con el eslabón 1 y llevándolo a su posición y después el mismo procedimiento para el segundo.

Los finales de carrera están conectados a una resistencia de *pull-up*, de esta manera se establece un estado lógico en un pin o entrada del microcontrolador cuando se encuentra en estado reposo. Las resistencias de *pull-up* o *pull-down* se puede conectar externamente o en algunos casos los mismos microcontroladores cuentan con esos circuitos y se pueden configurar mediante código, como es el caso de la STM32F4. De modo que los 4 pines PB0, PB1, PB2 y PB3 están configurados con resistencias de *pull-up* y con interrupciones habilitadas para detectar flancos de bajada.

En cuanto a la implementación del *homing*, primero se ha definido una estructura llamada homing_t con varios elementos, el primero determina el estado, normalmente asignaremos 1, cuando estamos ejecutando el *homing*, y 0 cuando se haya terminado. Asimismo, tenemos el estado de los cuatro finales de carrera, de forma similar, cuando no están presionados el estado es 0 y cuando lo son, el estado es 1. Finalmente tenemos dos variables para determinar cuándo el encoder ha dado una vuelta entera.

```
typedef struct{
      uint8_t state, B0, B1, B2, B3, Index1, Index2;
}homing_t;
```

Código 45. Estructura homing_t

Cuando creamos una estructura de tipo homing_t, hacemos que sea volátil y global para que una vez creada pueda ser accedida por distintos módulos. Es decir, la creamos en el fichero “homing.c” y en “homing.h” ponemos que es *extern*. De esta forma, cuando se produce una interrupción y se ejecute el manejador que tenga asignado dicha interrupción, y que está en el fichero “stm32f4xx_it.c”, podremos modificar el parámetro que queramos de la estructura. Esto se puede ver por ejemplo en el código 32, donde se hace la asignación homing.B1 = 1.

Volátil indica que un campo puede ser modificado por varios subprocesos que se ejecutan al mismo tiempo. Los campos declarados como volátil no están sujetos a optimizaciones del compilador que dan por hecho el acceso por parte de un único subproceso. Esto garantiza que el valor más actualizado esté presente en el campo en todo momento.

Todo el procedimiento de homing se describe en la función homing() que se muestra a continuación.

```
volatile homing_t homing;
void Homing(void){
	uint16_t i;
	homing.state = 1;
	NVIC_DisableIRQ(EXTI4_IRQn); // deshabilitamos interrupciones
	NVIC_DisableIRQ(EXTI9_5_IRQn); // de los Índices
	// Reseteamos valor del switch
	homing.B0 = 0;  homing.B1 = 0;  homing.B2 = 0;
	homing.B3 = 0;  homing.Index1=0; homing.Index2=0;
	// Buscamos el final de carrera 1
	while(homing.B0==0)
	{
		DAC->DHR12R1 = escalado(2.9)*sentido_motor1;
	}

	Delayms(1000);
	NVIC_EnableIRQ(EXTI4_IRQn);

	while(homing.Index1e==0){
		DAC->DHR12R1 = escalado(2.1)*sentido_motor1;
	}

	DAC->DHR12R1 = escalado(2.5);
	NVIC_DisableIRQ(EXTI4_IRQn);

	Delayms(1000);
	NVIC_EnableIRQ(EXTI9_5_IRQn);

	while(homing.Index1m==0){
		DAC->DHR12R1 = escalado(2.9)*sentido_motor1;
	}

	DAC->DHR12R1 = escalado(2.5);
	NVIC_DisableIRQ(EXTI9_5_IRQn);

	Delayms(2000);

	// Buscamos el final de carrera 2
	// …  MISMO PROCEDIMIENTO (SEGUNDO ESLABÓN)
	homing.state=0; // MODO HOMING OFF
}
```

Código 46. Función que implementa el homing.

Observe que las interrupciones que producen los índices del encoder sólo están habilitadas cuando estamos en modo homing. De esta manera se evita interrumpir el código principal cuando estamos en modo de funcionamiento normal, es decir cuando estamos en el bucle principal.

También observe que una vez situado en la posición inicial, se realza un movimiento en sentido contrario hasta determinar el primer índice del encoder, es decir, el del encoder que está del lado del elemento. Después se busca el segundo índice que está del lado del motor, con esto se puede obtener el desfase producido por el *harmonic drive*. De forma similar esto también es necesario por una parte para calcular el offset y por otra para así poder implementar una condición de que nunca se sobrepase esta posición contraria.

```
void EXTI4_IRQHandler(void) {

    if (EXTI_GetITStatus(EXTI_Line4) != RESET) {

            homing.Index1e=1;
            // paramos motor y registramos la posición
            DAC->DHR12R1 = escalado(2.5);
            encoder1.offset_index1e = TIM_GetCounter(TIM4);
            encoder3.offset_index1e = TIM_GetCounter(TIM8);


        // Clear interrupt flag //
        EXTI_ClearITPendingBit(EXTI_Line4);
    }
}
```

Código 47. Manejador de la interrupción del índice del encoder 1.

```
void EXTI9_5_IRQHandler(void) {

    if (EXTI_GetITStatus(EXTI_Line5) != RESET) {

            homing.Index1m=1;
            // paramos motor y registramos la posición
            DAC->DHR12R1 = escalado(2.5);
            encoder1.offset_index1m = TIM_GetCounter(TIM4);
            encoder3.offset_index1m = TIM_GetCounter(TIM8);

        // Clear interrupt flag
        EXTI_ClearITPendingBit(EXTI_Line5);
    }
}
```

Código 48. Manejador de la interrupción del índice del encoder 2.

En el código 47 y 48, tenemos el manejador de la interrupción del índice de los encoders del primer motor y elemento, para el segundo caso es similar. Observe que, es aquí donde se almacena la lectura del encoder tanto del lado del elemento como del lado del motor. Estos datos se corresponden con el offset y siempre y cuando se realice una lectura del encoder se tendrá que restar dicho offset. Esto se puede encontrar en el bucle principal o en la parada controlada y se realiza como se muestra a continuación.

```
// Lectura de la posición del encoder en pulsos por revolución
// Lectura del encoder 1 - q1
encoder1.posEncoder_PPR = TIM_GetCounter(TIM4) - encoder1.offset;
// Lectura del encoder 2 - q2
encoder2.posEncoder_PPR = TIM_GetCounter(TIM1) - encoder2.offset;
// Lectura del encoder 1 - theta1
encoder3.posEncoder_PPR = TIM_GetCounter(TIM8) - encoder3.offset;
// Lectura del encoder 2 - theta2
encoder4.posEncoder_PPR = TIM_GetCounter(TIM3) - encoder4.offset;
```

Código 49. Lectura de los encoder sin offset.

5.3. Sistema embebido con Linux: Raspberry PI

Raspberry Pi es un computador de placa reducida, computador de placa única o computador de placa simple (SBC) de bajo costo desarrollado en Reino Unido por la Fundación Raspberry Pi, con el objetivo de estimular la enseñanza de ciencias de la computación en las escuelas.

Cuenta con una capacidad de memoria RAM (*Random Access Memory* - memoria de acceso aleatorio-) de 512 MB, puerto ethernet, arquitectura ARM, puerto HDMI, puertos USB y 17 pines del GPIO que pueden ser usados como entradas o salidas digitales a 3,3V a 20mA. Además, existen pines para propósitos específicos como lo es la comunicación RS232 o I2C. Una desventaja de la Raspberry es que no cuenta con ADC (conversor análogo- digital) ni con DAC (conversor digital- análogo) aunque es verdad que se podría utilizar uno externo. Dicho módulo puede ser un integrado o una tarjeta con la capacidad de transmitirle las lecturas analógicas usando un canal de comunicación serie como por ejemplo I2C o RS232. Sin embargo, en este proyecto se comunicará con la STM32F4 que es la que se encargara de realizar la mayor parte de tareas.

Figura 47. Raspberry Pi 3 Modelo B.

El sistema operativo empleado por esta plataforma es Raspbian es de libre uso basado en Debian optimizado para el hardware de Raspberry Pi. Para controlar procesos la utilización de tarjetas como lo es la Raspberry Pi no es muy frecuente, la mayoría de aplicaciones de estas tarjetas de desarrollo corresponden a la utilización de esta como ordenador para funciones muy específicas o control de procesos simples. En nuestro caso se encargaría de resolver varias operaciones matriciales.

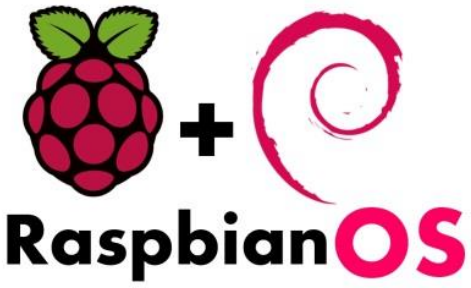


Figura 48. Sistema operativo utilizado por la Raspberry.

5.3.1. Lenguaje de programación Python

Este lenguaje no necesita compilador, es de alto nivel y solo requiere un intérprete por lo tanto cuando se ejecutan las líneas de código se hace directamente y sin necesidad de generar ejecutables. Otra característica muy importante es que destaca su facilidad de uso, sencillez y limpieza, además Python es orientado a objetos.

Figura 49. Logotipo de Python.

El motivo principal de utilizar Python es que existen muchas bibliotecas disponibles para extender la funcionalidad básica de Python a cualquier campo. Una de ellas es scipy, que cuenta con el paquete llamado NumPy que le agrega mayor soporte para vectores y matrices, constituyendo una biblioteca de funciones matemáticas de alto nivel para operar con esos vectores o matrices. Que para el caso de la implementación del LQR es necesario realizar varias operaciones con matrices y hacerlo en la STM32F4 con programación en C sería mucho más costoso.

Por otro lado, se podría implementar una aplicación de supervisión, en la que esté presente toda la información obtenida de los sensores, así como un historial o seguimiento de todas las actividades realizadas. En Python sería mucho más fácil que en la mayoría de lenguajes de programación.

Además, con efectos académicos se profundiza en nuevas materias como es el uso de un lenguaje distinto de los principales que se estudian en la facultad como es C, C++ y Java.

5.3.2. Diseño del LQR con seguimiento de trayectoria

Para diseñar el LQR con seguimiento de trayectoria, lógicamente, es necesario generar la trayectoria, para ello se ha creado una función llapada tapez[5] que es la misma que se implementó en la STM32F4 pero esta vez en Python. Para generar las trayectorias simplemente debemos llamar a la función y pasar los parámetros de posición inicial, posición final, la velocidad y aceleración máximas, así como el periodo de muestreo y el tiempo extra.

```
textra = 0.2
(qref1,vref1,aref1,t1)=trapez(0, 1.04, 1.8, 1.95, 0.01, textra);
(qref2,vref2,aref2,t2)=trapez(0, 0.35, 1, 1, 0.01, 0.48);
```

Código 50. Generación de las trayectorias en Python.

[5] La implementación de esta función se encuentra en el fichero LQR_trajectory.py y está en Anexos.

El siguiente paso es discretizar el modelo continuo, se podría pasar directamente el modelo discreto, pero siempre es más preciso discretizar directamente el continuo. Gracias a la librería scipy, esto se reduce simplemente a la llamada de la función cont2discrete. En forma de código sería:

```
from scipy.signal import cont2discrete as c2d

A, B, C, D, dt = c2d((Ac, Bc, Cc, Dc), dt_requested, method='zoh')
```

Código 51. Discretización del modelo en Python.

Antes de obtener el algoritmo LQR debemos definir las matrices Q, R y S y los estados iniciales al igual que lo hacíamos en la implementación en Matlab. Para este paso haremos uso de la librería numpy que nos permite crear matrices de una forma sencilla.

```
Q = np.diag([90000,90000,90000,90000,90000,90000,90000,90000])
P = np.diag([90000,90000,90000,90000,90000,90000,90000,90000])
R = matrix([[0.0001 ,0],[0 ,0.001]])
x0=matrix([[qref1[0],qref2[0],qref1[0],qref2[0],vref1[0],vref2[0],vr
ef1[0],vref2[0]]])
```

Código 52. Definición de las matrices Q, R y S y los estados iniciales en Python.

También de forma similar, tenemos que crear las matrices vacías para las ganancias, algunas de ellas son de tres dimensiones ya que en cada iteración se tiene una matriz.

```
k=len(qref1)
S=C.T*P*C
v = np.zeros((8, k))
x = np.zeros((8, k))
u = np.zeros((k, 2, 2))
K = np.zeros((k-1,2,8))
Kv = np.zeros((k-1,2,8))
```

Código 53. Creación vacía delas matrices de ganancias del LQR en Python.

Ahora ya si podemos implementar el algoritmo LQR y resolver de forma recursiva hacia atrás en el tiempo la ecuación diferencial de Riccati y la secuencia auxiliar vk.

```
for k in range(N-1, 0, -1):
    K[k,:,:] = np.dot(inv(R+np.dot(np.dot(B.T,S),B)),
    np.dot(np.dot(B.T,S),A))
    Kv[k,:,:] = np.dot(inv(R+np.dot(np.dot(B.T,S),B)),B.T)
    S = np.dot(np.dot(A.T,S),(A-np.dot(B,K[k,:,:])))+
    np.dot(np.dot(C.T,Q),C)
    v[:,[k]] = np.dot((A-np.dot(B,K[k,:,:])).T,v[:,[k+1]]) +
    np.dot(np.dot(C,Q),(matrix([[qref1[k],qref2[k],qref1[k],
    qref2[k],vref1[k],vref2[k],vref1[k],vref2[k]]])).T)
```

Código 54. Obtención de las ganancias del LQR en Python.

Sin olvidar x_0 inicial y la condición final de v_N:

```
xd = matrix([[qref1[N],qref2[N],qref1[N],qref2[N],vref1[N],vref2[N]
,vref1[N],vref2[N]]]).T
v[:,[N]]=C.T*P*xd
x[:,[0]] = x0.T
```

Código 55. x0 inicial y condición final de Vn.

Ahora ya podemos enviar las ganancias K y Kv con la comunicación SPI y realizar el control con la STM32F4. Para verificar que se ha realizado bien podemos simular para el modelo lineal al igual que se hizo en la implementación con Matlab.

```
for k in range(0, N):
  x[:,[k+1]] = np.dot(A,x[:,[k]])-np.dot(np.dot(B,K[k,:,:]),
  x[:,[k]])+np.dot(np.dot(B,Kv[k,:,:]),v[:,[k+1]])
  u[k,:,:] = np.dot(-K[k,:,:],x[:,[k+1]])+np.dot(Kv[k,:,:],v[:,k+1])
```

Código 56. Simulación en Python.

De modo que si representamos la trayectoria seguida de posición con la trayectoria generada se ve que se está realizando correctamente el seguimiento.

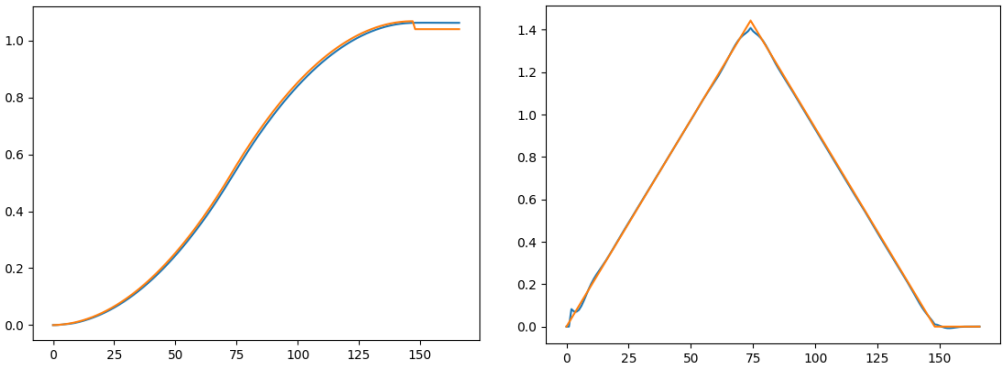


Figura 50. Resultado de la simulación, seguimiento de la trayectoria de posición y velocidad.

5.3.3. Comunicación SPI con la STM32F4

El primer paso sería establecer conexión entre la STM32F4 y la Raspberry para ello se utilizará la comunicación SPI. En el apartado 5.2.7 vimos la configuración para el caso de la STM32F4, donde se configuro en modo esclavo. Para el caso de la Raspberry la configuraremos en modo maestro. Además, por lo visto, la configuración en modo esclavo en este dispositivo resulta ser difícil por las limitaciones del chip BCM2835 y actualmente no hay mucha información sobre si hay algún sistema operativo o soporte de controladores para este modo.

Para configurar la comunicación SPI en la Raspberry Pi haremos uso de la librería spidev y en concreto la versión para Python. Siguiendo el manual[6] que suministran los desarrolladores, esto se debe realizar de la siguiente manera:

[6] Manual spidev: http://tightdev.net/SpiDev_Doc.pdf

```
spi = spidev.SpiDev() # create spi object
spi.open(0, 1) # open spi port 0, device (CS) 1
spi.max_speed_hz = 3500000
spi.bits_per_word = 8
spi.mode = 1
```

Código 57. Configuración SPI en la Raspberry PI con la librería spidev.

La configuración es muy simple, en primer lugar, se debe crear el objeto al que asignaremos algunos parámetros como el SPI utilizado, el pin de habilitación del esclavo, la velocidad de la comunicación, el tamaño de los datos a transmitir y el modo. De modo que se ha utilizado el SPI0 y el *chip selector* uno (CS1) . En cuanto a la velocidad se ha fijado en 3.5 MHz que es más que suficiente. El modo uno hace referencia a los parámetros CPOL y CPHA, es decir, el primero si el selector es el Maestro o el Esclavo y el segundo indica la polaridad del reloj. Así pues, el selector es el Maestro y la polaridad es activo en *high*.

Una de las principales desventajas de la librería para Python es que el tamaño máximo de la trama a transmitir está restringido en 4096 bytes y esto supone un problema porque normalmente el número de datos (ganancias K, Kv y secuencia v) que vamos a transmitir es superior a 20000 bytes. Lógicamente eso depende del periodo del muestreo y el tiempo de simulación.

Otro aspecto importante es que todos los datos que se quieren transmitir son de tipo *float*, por lo tanto, se han convertido a *signed int* de 32 bits y se ha dividido en cuatro de 8 bits que es lo permitido a la hora de transmitir. Todo esto se realiza en la siguiente función.

```
def convertFloatToFour8bit(x):
    x = int(round(x*1000))      # Para considerar 3 decimales
    a = (x >> 24) & 0x000000ff
    b = (x >> 16) & 0x000000ff
    c = (x >> 8) & 0x000000ff
    d = (x & 0x000000ff)
    return a,b,c,d
```

Código 58. Convertir float a cuatro de 8 bits.

Para enviar los datos emplearemos la función spi.xfer([values]) que se caracteriza por liberar y reactivar el CS entre bloques. Para enviar los datos, primero los agruparemos todos en un vector. De modo que, de cada parámetro obtenemos 4 bytes, al tener 5 parámetros obtendremos 20 bytes por estado, y teniendo en cuenta que las ganancias K, Kv y la secuencia v tienen 8 parámetros para cada estado, se obtienen 160 bytes por iteración; cada iteración es un instante de tiempo.

```
for i in range(0, N-1):
   for j in range(0, 8):
     data1, data2, data3, data4 = convertFloatToFour8bit(K[i,0,j])
     data5, data6, data7, data8 = convertFloatToFour8bit(K[i,1,j])
     data9,data10,data11,data12 = convertFloatToFour8bit(Kv[i,0,j])
     data13,data14,data15,data16 = convertFloatToFour8bit(Kv[i,1,j])
     data17, data18, data19, data20 = convertFloatToFour8bit(v[j,i])
     Data.append(data1)
     Data.append(data2)
     Data.append(data3)
     Data.append(data4)
     Data.append(data5)
     Data.append(data6)
     Data.append(data7)
     Data.append(data8)
     Data.append(data9)
     Data.append(data10)
     Data.append(data11)
     Data.append(data12)
     Data.append(data13)
     Data.append(data14)
     Data.append(data15)
     Data.append(data16)
     Data.append(data17)
     Data.append(data18)
     Data.append(data19)
     Data.append(data20)
```

Código 59. Agrupación de datos.

Evidentemente, luego, para interpretar los datos se debe saber el orden en el que se han enviado. En el apartado 5.2.7 se describe como los datos son recibidos e interpretados.

Finalmente, se muestra el código dónde el vector de datos es dividido en tramas de 4000 bytes por la limitación que comentamos. Para la última trama esta podría ser menor a 4000 y esto se ha tenido en cuenta, tal y como se puede observar a continuación.

```
if ((len(Data)/4000) > spi_tx):
    spi_tx = spi_tx + 1

for i in range(0, spi_tx):
    if (4000+4000*i) < len(Data):
        spi.xfer(Data[(4000*i):(4000+4000*i)])
    else:
        spi.xfer(Data[(4000*i):len(Data)])
    time.sleep(0.5)
```

Código 60. Envío de datos por SPI.

CONCLUSIONES

Se han implementado cuatro formas de controlar el brazo robot, los dos primeros basados en el control proporcional derivativo y los dos segundos basado en el regulador lineal cuadrático. Se han analizado las diferencias y las ventajas de cada uno, de modo que se ha llegado a la conclusión de que el control PD es suficiente para controlar el brazo robot. Su implementación es sencilla, sin embargo, al estar ante un proceso de múltiples entradas, que en este caso son dos, es necesario utilizar dos PDs independientes para cada entrada. Cabe destacar que, para garantizar la estabilidad, en la implementación del PD es necesario realimentar la velocidad angular del lado del motor pudiendo emplear la posición angular, tanto del lado del motor como del lado del elemento. En cuanto a los resultados obtenidos, se ha visto que con un buen ajuste de las ganancias los resultados suelen ser buenos. Además, añadir la compensación de los efectos de gravedad supone un mejor seguimiento de la trayectoria, de modo que, se suele alcanzar la posición final deseada.

Por su parte el controlador optimo cuadrático permite encontrar un buen desempeño en sistemas multivariables, se ha conseguido una reducción considerable en las oscilaciones y además permite el manejo del ahorro energético. Se ha estudiado la importancia de las matrices Q, R y S y se ha llegado a la conclusión de que su selección adecuada es fundamental para la obtención de la respuesta deseada. Hay que aclarar que Q y S afectan directamente al seguimiento de la trayectoria y R a la optimización de la acción de control. Ahora bien, para conseguir un control servible, la matriz S debe ser suficientemente alta en comparación con R, en caso contrario no se llega a alcanzar la referencia o posición final. Teniendo en cuenta lo anterior, variando Q se ha podido manipular el seguimiento de trayectoria y de esta forma el manejo del ahorro energético.

Para el ahorro energético, al estar ante un control por corriente y sabiendo que existe una relación de proporcionalidad entre el par y la intensidad, se ha utilizado el par como factor que determina el ahorro. En las baterías es normal el uso del miliamperio hora (mAh), que es la milésima parte del Ah, o lo que es lo mismo 3,6 culombios. Esto indica la máxima carga eléctrica que es capaz de almacenar la batería. A más carga eléctrica almacenada, más tiempo tardará en descargarse.

De modo que se compara la acción de control de un perfecto seguimiento de la trayectoria con otra trayectoria que alcanza la referencia, pero no sigue bien la referencia. Para ello, se utiliza el valor medio cuadrático del par y de ahí obtenemos el ahorro de consumo del uno con respecto del otro.

Por otro lado, con el regulador lineal cuadrático iterativo se tiene en cuenta la dinámica variante del modelo, por lo tanto, es una mejora sustancial al tratarse de control de plantas no lineales. Sin embargo, no hay que olvidar que al ser un algoritmo basado en el principio del máximo de Pontryagin en general se encuentran soluciones localmente óptimas, así pues, problemas complejos de control pueden presentar muchos mínimos locales. Hay que decir, que no se ha podido probar en la planta real, pero a partir de las simulaciones se ha observado que es un buen optimizador de trayectorias.

También decir que el iLQR se ha implementado para llevar el sistema a un punto deseado, es decir no se realiza seguimiento de trayectoria. Por lo tanto, para algunas aplicaciones donde es obligatorio seguir una trayectoria determinada esta parte sería necesaria. Un ejemplo es en los robots industriales, por ejemplo en ABB cuando se utilizan los comandos moveJ y moveL, para el primer caso la precisión no es importante y para el segundo sí lo es. A pesar de ello, esta parte no se ha implementado y se propone como un futuro trabajo.

Cambiando de tema, para llevar a cabo el control a la planta real se han utilizado dos dispositivos, el microcontrolador STM32F4 y el sistema embebido Raspberry Pi. Hay que recalcar que la Raspberry Pi es un mini PC y normalmente no suele utilizarse para controlar sistemas ya que el tema de tiempo real no lo lleva muy bien. Existen sistemas operativos que dan soporte al procesamiento de datos en tiempo real pero este tema no se ha tratado. Una alternativa seria utilizar la BeagleBone que tiene dos microcontroladores integrados llamados PRUs, pudiendo lograr un control rápido, determinista y en tiempo real de los pines y dispositivos de E / S.

Volviendo a la Raspberry Pi, decir que se ha utilizado para calcular offline las ganancias y la secuencia auxiliar necesarios para implementar el algoritmo LQR con seguimiento de trayectoria. Al disponer de SO, se pueden emplear distintos lenguajes de programación; para nuestro caso, ha sido el uso del lenguaje Python, siendo un lenguaje de alto nivel que dispone de una infinidad de extensiones y librerías. Gracias a ello, la implementación del LQR se ha reducido en escribir un par de líneas de código.

Más allá, como trabajos futuros, las aplicaciones que se pueden desarrollar son infinitas; asimismo unido al concepto de IOT lo que viene siendo la interconexión digital de objetos cotidianos con internet, permitirá al usuario disponer de un seguimiento de todos los aspectos relacionados con el brazo robot desde cualquier dispositivo.

Dicho lo anterior, el verdadero responsable de realizar el control es el microcontrolador STM32F4 que además es el que interactúa con todos los sensores y actuadores. En él se han implementado los dos controles, el PD y el PD con compensación de gravedad. Para realizar el control LQR es necesario disponer de todas las ganancias en cada instante siendo algo que ya hemos realizado con la Raspberry Pi. De modo que, para que esto sea posible se ha tenido que establecer una comunicación entre los dos dispositivos. La idea principal fue utilizar el bus CAN, sin embargo, las Raspberry Pi no dispone de ello y como consecuencia, sería necesario adquirir un módulo CAN. Esto no se ha hecho ya que finalmente se ha optado por emplear la comunicación SPI. Además, para no emplear la CPU se ha conectado el periférico directamente a la DMA, almacenando todos los datos en un buffer. Durante el proceso de envió es muy importante saber cómo se encapsula la información antes de ser enviada, para poder interpretarla en el receptor.

Otro aspecto muy importante es el tema de seguridad, de modo que el brazo robot debe estar dotado de mecanismo que en caso de emergencia o un estado indeseado pueda ser inmovilizado. Esto, por una parte, se debe realizar con un *software* y por otro disponer del *hardware* adecuado.

Finalmente decir que desafortunadamente, no se ha podido probar el control en la planta real por falta de tiempo. Además, al ser un proyecto en el que participan varias personas, ha habido retrasos en los objetivos inicialmente propuestos, de modo que, no se han cumplido muchas de las tareas en las fechas concretadas. Sin embargo, la mayor parte de las funcionalidades se han probado y en principio todo está implementado y listo para ser usado una vez este montado el brazo robótico.

BIBLIOGRAFÍA

Control Automático 2. (s.f.). En N. Básico, *Introduccion al Control Óptimo.* 2001. Obtenido de http://www.eng.newcastle.edu.au/~jhb519/teaching/caut2/clases/Cap9.pdf

DeWolf, T. (10 de 09 de 2015). *LINEAR-QUADRATIC REGULATION FOR NON-LINEAR SYSTEMS USING FINITE DIFFERENCES*. Obtenido de studywolf: https://studywolf.wordpress.com/2015/11/10/linear-quadratic-regulation-for-non-linear-systems-using-finite-differences/

DeWolf, T. (03 de 02 de 2016). *THE ITERATIVE LINEAR QUADRATIC REGULATOR ALGORITHM*. Obtenido de studywolf: https://studywolf.wordpress.com/2016/02/03/the-iterative-linear-quadratic-regulator-method/

Differential Dynamic Programming Solver iLQR. (2015). Obtenido de https://github.com/flforget/DDP

Emanuel Todorov, W. L. (January 2004). *Iterative Linear Quadratic Regulator Design for Nonlinear Biological Movement Systems.*

Fajardo, D. F. (2010). *Simulación de Sistemas Dinámicos Mediante.* Universidad de Narino, Departamento de Matemáticas. Obtenido de file:///C:/Users/Aleks/Downloads/Dialnet-SimulacionDeSistemasDinamicosMedianteDiscretizacio-3714810.pdf

Frank L. Lewis, D. V. (s.f.). *Optimal Control* (Third Edition ed.). (I. John Wiley & Sons, Ed.)

Khatib, P. O. (22 de 07 de 2008). *ntroduction to Robotics (CS223A).* Obtenido de https://www.youtube.com/watch?v=0yD3uBshJB0&list=PL64324A3B147B5578

Mantz, R. J. (2003). *Introducción al control óptimo.* Universidad Nacional de La Plata. Obtenido de https://catedra.ing.unlp.edu.ar/electrotecnia/controlm/electronica/archivos/apuntes/c_optimo.pdf

Messner, P. B. (s.f.). *Inverted Pendulum: State-Space Methods for Controller Design*. Obtenido de http://ctms.engin.umich.edu: http://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum&section=ControlStateSpace

Oussama Khatib, B. S. (2008). *Springer Handbook of Robotics, Chapter 13: Robots with Flexible Elements.*

Rojo, J. F. (2016). *DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR PROGRAMABLE DE PATRONES DE MOVIMIENTO PARA MOTORES DE CORRIENTE CONTINUA CON APLICACIÓN EN ROBOTS Y MÁQUINAS CNC.* Valencia: 06.

Seron, M. M. (1996). *Sistemas No Lineales.* Prentice Hall.

Tatiana Poveda Galvis, J. D. (2016). Diseño e implementación de un control óptimo LQR con la tarjeta Raspberry Pi. Obtenido de http://repository.udistrital.edu.co/bitstream/11349/3372/1/PovedaGalvisLeidyTatiana2016.pdf

Tedrake, R. (14 de 07 de 2010). *Underactuated Robotics.* Obtenido de https://www.youtube.com/watch?v=xwgIkdBQku4

Wen, A. W. (1997). Trajectory Tracking Control of a Car-Trailer System. Obtenido de https://pdfs.semanticscholar.org/d2d6/8ca2a98239d582c6fb16c9f93cd5edfea6d6.pdf

ANEXOS

A. Generación de trayectoria

Para realizar el control con seguimiento de trayectoria dijimos que existen distintos tipos de movimiento para sistemas robotizados, entre los cuales, el más utilizado es el movimiento punto a punto y concretamente el correspondiente a patrones o perfiles de movimiento en forma de curva trapezoidal o de curva en S.

En el contexto del movimiento punto a punto, una curva en S consta de 7 fases distintas de movimiento. En la fase I la carga comienza a moverse desde el reposo a una aceleración linealmente creciente hasta un valor máximo. En la fase II, la carga acelera de forma constante manteniendo este valor máximo. En la fase III comienza una deceleración lineal en la que la aceleración llega a 0, instante en el que la velocidad alcanza su valor máximo. La velocidad se mantiene constante durante la fase IV, y después la carga decelera de forma simétrica a las tres primeras fases en las fases V, VI y VII.

Los perfiles trapezoidales, por el contrario, tienen solo 3 fases correspondientes a fase I, aceleración constante, fase II, velocidad constante y fase III, deceleración constante.

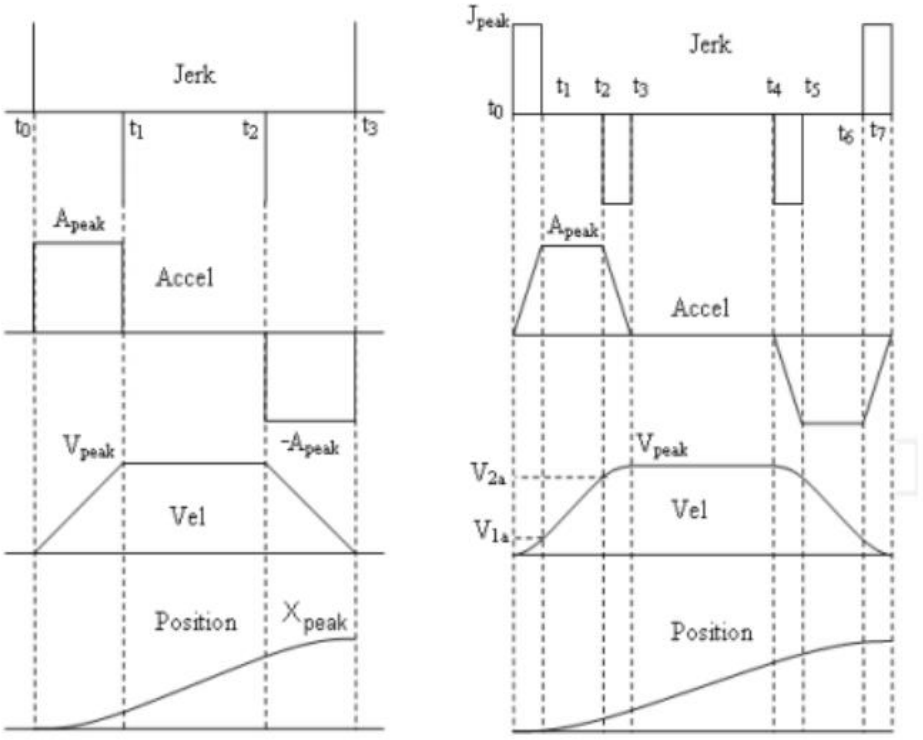


Figura 51. Curva en S trapezoidal (Izquierda) y curva en S polinomial de tercer orden (derecha).

Esta diferencia de fases marca la diferencia entre ambos perfiles, resultando las fases adicionales presentes en el perfil en S en unas transiciones entre periodos de aceleración y no aceleración más suave. Por el contrario, las transiciones del perfil trapezoidal son instantáneas.

Las características de movimiento asociadas a las variaciones de la aceleración se denominan “jerk”. Se define como la variación de la aceleración respecto al tiempo, si el intervalo es lo suficientemente pequeño se traduce en su derivada. En los perfiles trapezoidales, el jerk es infinito en las transiciones, lo que implica una discontinuidad transmitida a la carga como un rebote. En los perfiles en S, el jerk es constante, y la energía asociada al cambio de la aceleración se libera a tiempo, eliminando ese enganche del caso anterior.

Así pues, variar la aceleración de forma gradual posee varias ventajas, una de las cuales es la menor oscilación de la carga. Para una carga dada, a mayor jerk, mayor la vibración indeseada, pudiendo estropear completamente el control y la precisión del mismo si se alcanza el estado de resonancia.

Como los perfiles trapezoidales funcionan con aceleración y deceleración máximas son, desde el punto de vista de la ejecución, más rápidos que los perfiles en S. Sin embargo, si la aproximación todo o nada causa un incremento en el tiempo de establecimiento, esta ventaja podría perderse. Afortunadamente, esto puede arreglarse e incluso optimizarse el rendimiento añadiendo una pequeña transición entre periodos de reposo y aceleración.

En cualquier caso, la meta de cualquier perfil de movimiento es alcanzar las características del sistema para la aplicación deseada. Los perfiles de curvas trapezoidales y en S funcionan bien cuando la curva de respuesta del par del sistema es lo bastante plana; esto sucede cuando el par de salida no varía mucho respecto al rango de velocidad esperado. Esto es cierto para la mayoría de sistemas basados en servomotores de corriente continua, ya sean con o sin escobillas.

A.1. Perfil de movimiento trapezoidal

Como se ha indicado antes, consiste en tres etapas: una aceleración, una velocidad máxima y una deceleración. La posición se define a partir de polinomios de segundo orden por lo que a veces se le llama polinomial de segundo orden.

$$a = \begin{cases} A_{peak}, & t_0 \le t \le t_1 \\ -A_{peak}, & t_2 \le t \le t_3 \end{cases}$$

Las matemáticas detrás de los perfiles de curvas trapezoidales son sencillas. Existen dos formas que se pueden utilizar: la forma continua (forma de ecuaciones físicas) y la forma discreta, utilizada en la mayoría de sistemas de móviles que utilizan microprocesadores para generar los nuevos parámetros de movimiento en cada cuenta del reloj.

Forma continua:

$$p(t) = p_o + v_o t + \frac{1}{2} a t^2$$

$$v(t) = v_o + at$$

$$a(t) = a$$

$$p_{decel} = \frac{v^2}{2a}$$

Forma discreta

$$p(k) = p_{k-1} + v_k$$

$$v(k) = v_{k-1} + a$$

donde po y vo son la posición y velocidad iniciales; pk y vk son la posición y velocidad en el instante k y a es la aceleración.

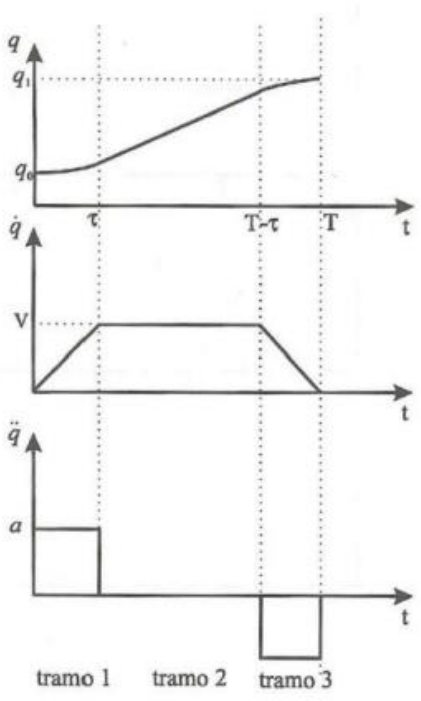


Figura 52. Ejemplos de perfil trapezoidal.

A.2. Perfil de movimiento curva en S. Aproximación de orden n

De forma similar al planteamiento de las curvas de tercer orden, las de cuarto poseen 16 segmentos o fases que definir. Su planteamiento llamando M3 al perfil de la curva de tercer orden es:

$$M_4 = \begin{cases} M_3, & t_0 \le t \le t_7 \\ -M_3, & t_8 \le t \le t_{15} \end{cases}$$

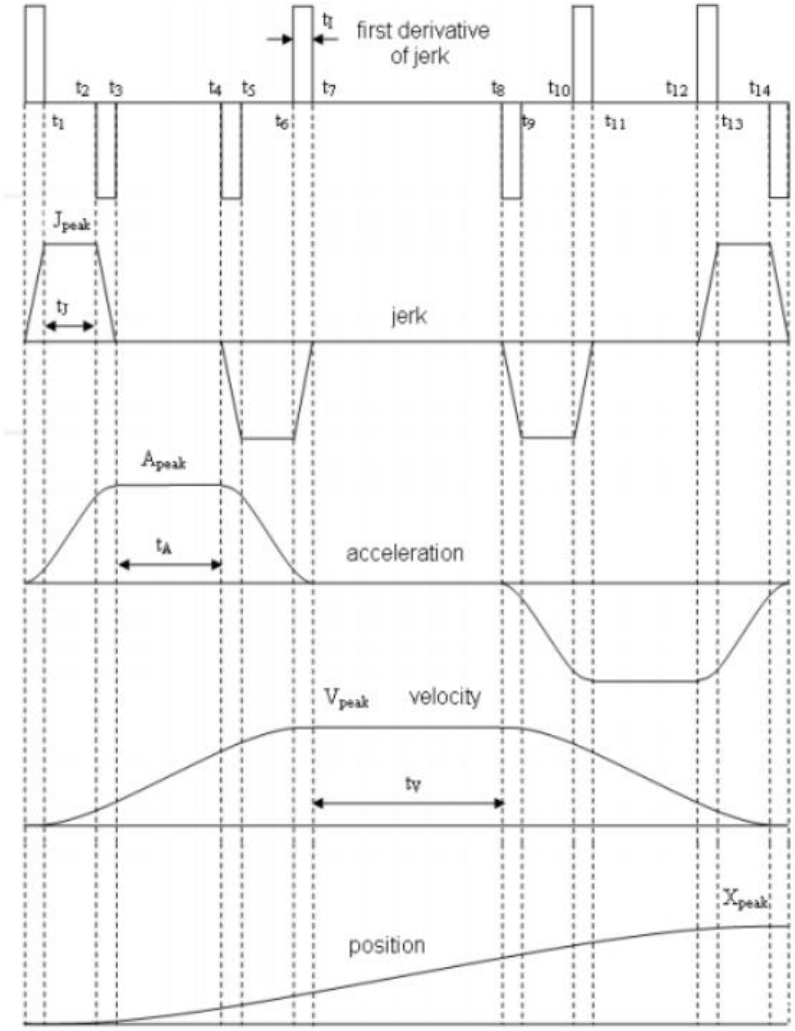


Figura 53. Perfil de curva de cuarto orden

Y en general, para una curva polinomial de orden N tendremos 2^N fases, en las que la aceleración tendrá la forma por tramos de la velocidad del perfil anterior:

$$M_n = \begin{cases} M_{n-1}, & t_0 \le t \le t_{2^{n-1}-1} \\ -M_{n-1}, & t_{2^{n-1}} \le t \le t_{2^n-1} \end{cases}$$

A.3. Código creado en Python para obtener la trapezoidal de 2º orden

```
def trapez(qr1, qr2, qpr, qsec, tm, textra):
    s=qr2-qr1;
    sign = np.sign(s);
    global t1, t2, tfin

    if (abs(s) > (qpr*qpr/qsec)):
        t1 = (qpr/qsec)
        t2 = (abs(s)/qpr)
        tfin = t2+t1
    else:
        t1 = math.sqrt(abs(s)/qsec)
        tfin = 2.0 * t1
        t2=0

    position = []
    velocity = []
    acceleration = []
    tiempo = []
    position.append(qr1)
    velocity.append(0)
    acceleration.append(0)

    print t1
    print t2
    print tfin
    for time in range(0, int((tfin+textra)/tm)):
        #print time
        #print t1
        tiempo.append(time*tm)

        if (time < t1/tm):
            #print ('E1')
            acceleration.append(sign*qsec)
            velocity.append(velocity[time] +
acceleration[time+1]*tm)
            position.append(position[time] + velocity[time+1]*tm +
0.5*acceleration[time+1]*tm*tm)

        elif ((time >= t1/tm)&(time < t2/tm)):
            #print ('E2')
            acceleration.append(0)
            velocity.append(sign*qpr)
            position.append(position[time] + velocity[time+1]*tm)
        elif ((time >=t2/tm)&(time <=tfin/tm)):
            #print ('E3')
            acceleration.append(-sign*qsec)
            velocity.append(velocity[time] +
acceleration[time+1]*tm)
            position.append(position[time] + velocity[time+1]*tm +
0.5*acceleration[time+1]*tm*tm)
        else:
            #print ('E4')
            acceleration.append(0)
            velocity.append(0)
            position.append(qr2)

    return (position, velocity, acceleration, tiempo)
```

Código 61. Generación de la trayectoria trapezoidal 2º orden en Python.

B. Cinemática inversa

En la Figura 45 se muestra la representación gráfica de lo que se necesita para utilizar este método, el cual consiste en conocer los valores reales de las coordenadas de x e y del efector final del robot, para que de esta forma se obtengan los valores de β y α, por ejemplo para obtener q_1 que representa a θ_1 se obtiene mediante la resta de $\beta - \alpha$ ec. (A.3), y por ultimo para obtener q_2 se requiere conocer el valor del coseno de θ_2, este valor se obtiene mediante la ec. (A.1). Al conocer este datose puede saber el valor de θ_2 mediante la ec. (A.2)

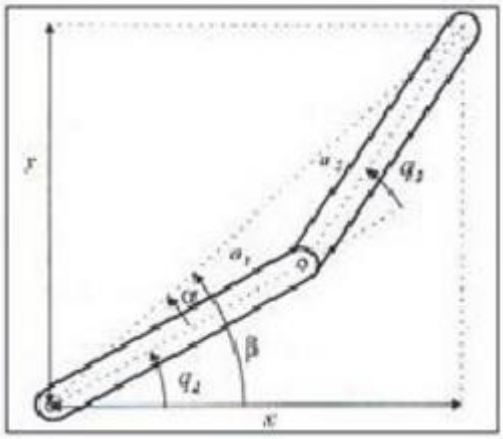


Figura 54. Representación gráfica del Robot de 2 gdl.

$$D = cos\theta_2 = \frac{(x^2 + y^2 - a_1^2 - a_2^2)}{2a_1a_2} \quad \text{(A.1)}$$

$$\theta_2 = \tan^{-1}\left(\frac{\sqrt[2]{1 - D^2}}{D^2}\right) \quad \text{(A.2)}$$

$$\theta_2 = \tan^{-1}\left(\frac{y}{x}\right) - \tan^{-1}\left(\frac{a_2 \sin\theta_2}{a_1 + a_2 \cos\theta_2}\right) \quad \text{(A.3)}$$

C. Resultado de las matrices jacobianas A y B

A continuación, se muestra el código implementado en Matlab para resolver y obtener todas las derivadas parciales. En primer lugar tenemos las derivadas parciales de $\dot{x}_5$:

```
    %% Derivadas parciales columna 5 matriz A
%     df5_dx1 = diff(x_dot(1,:),q1)
    df5_dx1 = (24*((49*cos(q1)*sin(q2))/4 +
(49*cos(q2)*sin(q1))/4)*(3*cos(q2) + 2))/(45*cos(q2)^2 - 124) -
(48*((294*sin(q1))/5 + (49*cos(q1)*sin(q2))/4 +
(49*cos(q2)*sin(q1))/4 - 10000))/(45*cos(q2)^2 - 124);

%     df5_dx2 = diff(x_dot(1,:),q2)
    df5_dx2 = 12000/(45*cos(q2)^2 - 124) -
(48*((49*cos(q1)*sin(q2))/4 + (49*cos(q2)*sin(q1))/4 +
(5*q_prim2*cos(q2)*(2*q_prim1 + q_prim2))/8))/(45*cos(q2)^2 - 124) -
(24*(3*cos(q2) + 2)*((5*cos(q2)*q_prim1^2)/8 -
(49*cos(q1)*sin(q2))/4 - (49*cos(q2)*sin(q1))/4 +
10000))/(45*cos(q2)^2 - 124) + (72*sin(q2)*((5*sin(q2)*q_prim1^2)/8
+ 10000*q2 - 10000*theta2 + (49*cos(q1)*cos(q2))/4 -
(49*sin(q1)*sin(q2))/4))/(45*cos(q2)^2 - 124) +
(4320*cos(q2)*sin(q2)*(10000*q1 - 10000*theta1 + (294*cos(q1))/5 +
(49*cos(q1)*cos(q2))/4 - (49*sin(q1)*sin(q2))/4 -
(5*q_prim2*sin(q2)*(2*q_prim1 + q_prim2))/8))/(45*cos(q2)^2 - 124)^2
+ (108*cos(q2)*sin(q2)*(10000*q2 - 10000*theta2 +
tau2))/(45*cos(q2)^2 - 124)^2 - (2160*cos(q2)*sin(q2)*(3*cos(q2) +
2)*((5*sin(q2)*q_prim1^2)/8 + 10000*q2 - 10000*theta2 +
(49*cos(q1)*cos(q2))/4 - (49*sin(q1)*sin(q2))/4))/(45*cos(q2)^2 -
124)^2;

%     df5_dx3 = diff(x_dot(1,:),theta1)
    df5_dx3 = -480000/(45*cos(q2)^2 - 124);

%     df5_dx4 = diff(x_dot(1,:),theta2)
    df5_dx4 = (240000*(3*cos(q2) + 2))/(45*cos(q2)^2 - 124) -
12000/(45*cos(q2)^2 - 124);

%     df5_dx5 = diff(x_dot(1,:),q_prim1)
    df5_dx5 = - (60*q_prim2*sin(q2))/(45*cos(q2)^2 - 124) -
(30*q_prim1*sin(q2)*(3*cos(q2) + 2))/(45*cos(q2)^2 - 124);

%     df5_dx6 = diff(x_dot(1,:),q_prim2)
    df5_dx6 = -(48*((5*q_prim2*sin(q2))/8 + (5*sin(q2)*(2*q_prim1 +
q_prim2))/8))/(45*cos(q2)^2 - 124);

%     df5_dx7 = diff(x_dot(1,:),q_sec1)
    df5_dx7 = 0;

%     df5_dx8 = diff(x_dot(1,:),q_sec2)
    df5_dx8 = 0;

%     df5_du1 = diff(x_dot(1,:),tau1)
    df5_du1 = 0;

%     df5_du2 = diff(x_dot(1,:),tau2)
    df5_du2 = 6/(5*(45*cos(q2)^2 - 124));
```

A continuación se presentan las derivadas parciales de $\dot{x}_6$:

```
%% Derivadas parciales columna 6 matriz A
%     df6_dx1 = diff(x_dot(2,:),q1)
    df6_dx1 = (24*(3*cos(q2) + 2)*((294*sin(q1))/5 +
(49*cos(q1)*sin(q2))/4 + (49*cos(q2)*sin(q1))/4 -
10000))/(45*cos(q2)^2 - 124) - (144*((49*cos(q1)*sin(q2))/4 +
(49*cos(q2)*sin(q1))/4)*(5*cos(q2) + 12))/(5*(45*cos(q2)^2 - 124));

%     df6_dx2 = diff(x_dot(2,:),q2)   --- ESTO NO LO HACE BIÉN
    df6_dx2 = (9*sin(q2)*(10000*q2 + tau2 -
10000*theta2))/(5*(45*cos(q2)^2 - 124)) - (6000*(3*cos(q2) +
2))/(45*cos(q2)^2 - 124) + (24*(3*cos(q2) +
2)*((49*cos(q1)*sin(q2))/4 + (49*cos(q2)*sin(q1))/4 +
(5*q_prim2*cos(q2)*(2*q_prim1 + q_prim2))/8))/(45*cos(q2)^2 - 124) +
(72*sin(q2)*(10000*q1 - 10000*theta1 + (294*cos(q1))/5 +
(49*cos(q1)*cos(q2))/4 - (49*sin(q1)*sin(q2))/4 -
(5*q_prim2*sin(q2)*(2*q_prim1 + q_prim2))/8))/(45*cos(q2)^2 - 124) +
(144*(5*cos(q2) + 12)*((5*cos(q2)*q_prim1^2)/8 -
(49*cos(q1)*sin(q2))/4 - (49*cos(q2)*sin(q1))/4 +
10000))/(5*(45*cos(q2)^2 - 124)) -
(144*sin(q2)*((5*sin(q2)*q_prim1^2)/8 + 10000*q2 - 10000*theta2 +
(49*cos(q1)*cos(q2))/4 - (49*sin(q1)*sin(q2))/4))/(45*cos(q2)^2 -
124) + (2592*cos(q2)*sin(q2)*(5*cos(q2) +
12)*((5*sin(q2)*q_prim1^2)/8 + 10000*q2 - 10000*theta2 +
(49*cos(q1)*cos(q2))/4 - (49*sin(q1)*sin(q2))/4))/(45*cos(q2)^2 -
124)^2 - (54*cos(q2)*sin(q2)*(3*cos(q2) + 2)*(10000*q2 + tau2 -
10000*theta2))/(45*cos(q2)^2 - 124)^2 -
(2160*cos(q2)*sin(q2)*(3*cos(q2) + 2)*(10000*q1 - 10000*theta1 +
(294*cos(q1))/5 + (49*cos(q1)*cos(q2))/4 - (49*sin(q1)*sin(q2))/4 -
(5*q_prim2*sin(q2)*(2*q_prim1 + q_prim2))/8))/(45*cos(q2)^2 -
124)^2;

%     df6_dx3 = diff(x_dot(2,:),theta1)
    df6_dx3 = (240000*(3*cos(q2) + 2))/(45*cos(q2)^2 - 124);

%     df5_dx4 = diff(x_dot(2,:),theta2)
    df6_dx4 = (6000*(3*cos(q2) + 2))/(45*cos(q2)^2 - 124) -
(288000*(5*cos(q2) + 12))/(45*cos(q2)^2 - 124);

%     df6_dx5 = diff(x_dot(2,:),q_prim1)
    df6_dx5 = (30*q_prim2*sin(q2)*(3*cos(q2) + 2))/(45*cos(q2)^2 -
124) + (36*q_prim1*sin(q2)*(5*cos(q2) + 12))/(45*cos(q2)^2 - 124);

%     df6_dx6 = diff(x_dot(2,:),q_prim2)
    df6_dx6 = (24*(3*cos(q2) + 2)*((5*q_prim2*sin(q2))/8 +
(5*sin(q2)*(2*q_prim1 + q_prim2))/8))/(45*cos(q2)^2 - 124);

%     df6_dx7 = diff(x_dot(2,:),q_sec1)
    df6_dx7 = 0;

%     df6_dx8 = diff(x_dot(2,:),q_sec2)
    df6_dx8 = 0;

%     df6_du1 = diff(x_dot(2,:),tau1)
    df6_du1 = 0;

%     df6_du2 = diff(x_dot(2,:),tau2)
    df6_du2 = -(3*(3*cos(q2) + 2))/(5*(45*cos(q2)^2 - 124));
```

A continuación se presentan las derivadas parciales de $\dot{x}_8$

```
%% Derivadas parciales columna 8 matriz A
%     df8_dx1 = diff(x_dot(4,:),q1)
    df8_dx1 = (6*((294*sin(q1))/5 + (49*cos(q1)*sin(q2))/4 +
(49*cos(q2)*sin(q1))/4 - 10000))/(5*(45*cos(q2)^2 - 124)) -
(3*((49*cos(q1)*sin(q2))/4 + (49*cos(q2)*sin(q1))/4)*(3*cos(q2) +
2))/(5*(45*cos(q2)^2 - 124));

%     df8_dx2 = diff(x_dot(4,:),q2)
    df8_dx2 = (6*((49*cos(q1)*sin(q2))/4 + (49*cos(q2)*sin(q1))/4 +
(5*q_prim2*cos(q2)*(2*q_prim1 + q_prim2))/8))/(5*(45*cos(q2)^2 -
124)) + (900*(1200*cos(q2)^2 - 3307))/(45*cos(q2)^2 - 124) +
(3*(3*cos(q2) + 2)*((5*cos(q2)*q_prim1^2)/8 - (49*cos(q1)*sin(q2))/4
- (49*cos(q2)*sin(q1))/4 + 10000))/(5*(45*cos(q2)^2 - 124)) -
(9*sin(q2)*((5*sin(q2)*q_prim1^2)/8 + 10000*q2 - 10000*theta2 +
(49*cos(q1)*cos(q2))/4 - (49*sin(q1)*sin(q2))/4))/(5*(45*cos(q2)^2 -
124)) - (108*cos(q2)*sin(q2)*(10000*q1 - 10000*theta1 +
(294*cos(q1))/5 + (49*cos(q1)*cos(q2))/4 - (49*sin(q1)*sin(q2))/4 -
(5*q_prim2*sin(q2)*(2*q_prim1 + q_prim2))/8))/(45*cos(q2)^2 - 124)^2
- (216*cos(q2)*sin(q2)*(10000*q2 + tau2 -
10000*theta2))/(45*cos(q2)^2 - 124) + (54*cos(q2)*sin(q2)*(3*cos(q2)
+ 2)*((5*sin(q2)*q_prim1^2)/8 + 10000*q2 - 10000*theta2 +
(49*cos(q1)*cos(q2))/4 - (49*sin(q1)*sin(q2))/4))/(45*cos(q2)^2 -
124)^2 + (81*cos(q2)*sin(q2)*(1200*cos(q2)^2 - 3307)*(10000*q2 +
tau2 - 10000*theta2))/(10*(45*cos(q2)^2 - 124)^2);

%     df8_dx3 = diff(x_dot(4,:),theta1)
    df8_dx3 = 12000/(45*cos(q2)^2 - 124);

%     df8_dx4 = diff(x_dot(4,:),theta2)
    df8_dx4 = - (6000*(3*cos(q2) + 2))/(45*cos(q2)^2 - 124) -
(900*(1200*cos(q2)^2 - 3307))/(45*cos(q2)^2 - 124);

%     df8_dx5 = diff(x_dot(4,:),q_prim1)
    df8_dx5 = (3*q_prim2*sin(q2))/(2*(45*cos(q2)^2 - 124)) +
(3*q_prim1*sin(q2)*(3*cos(q2) + 2))/(4*(45*cos(q2)^2 - 124));

%     df8_dx7 = diff(x_dot(4,:),q_prim2)
    df8_dx6 = (6*((5*q_prim2*sin(q2))/8 + (5*sin(q2)*(2*q_prim1 +
q_prim2))/8))/(5*(45*cos(q2)^2 - 124));

%     df8_dx7 = diff(x_dot(4,:),q_sec1)
    df8_dx7 = 0;

%     df8_dx8 = diff(x_dot(4,:),q_sec2)
    df8_dx8 = 0;

%     df8_du1 = diff(x_dot(4,:),tau1)
    df8_du1 = 0;

%     df8_du2 = diff(x_dot(4,:),tau2)
    df8_du2 = (9*(1200*cos(q2)^2 - 3307))/(100*(45*cos(q2)^2 -
124));
```

A continuación se presentan las derivadas parciales de $\dot{x}_7$:

```
%% Derivadas parciales columna 7 matriz A
%     df7_dx1 = diff(x_dot(3,:),q1)
    df7_dx1 = 7500;

%     df7_dx2 = diff(x_dot(3,:),q2)
    df7_dx2 = 0;

%     df7_dx3 = diff(x_dot(3,:),theta1)
    df7_dx3 = -7500;

%     df7_dx4 = diff(x_dot(3,:),theta2)
    df7_dx4 = 0;

%     df7_dx5 = diff(x_dot(3,:),q_prim1)
    df7_dx5 = 0;

%     df6_dx7 = diff(x_dot(3,:),q_prim2)
    df7_dx6 = 0;

%     df7_dx7 = diff(x_dot(3,:),q_sec1)
    df7_dx7 = 0;

%     df7_dx8 = diff(x_dot(3,:),q_sec2)
    df7_dx8 = 0;

%     df7_du1 = diff(x_dot(3,:),tau1)
    df7_du1 = 3/4;

%     df7_du2 = diff(x_dot(3,:),tau2)
    df7_du2 = 0;
```

Y finalmente las matrices A y B sustituyendo todas las derivadas parciales serían:

```
A = [0 0 0 0 1 0 0 0;
        0 0 0 0 0 1 0 0;
        0 0 0 0 0 0 1 0;
        0 0 0 0 0 0 0 1;
        df5_dx1 df5_dx2 df5_dx3 df5_dx4 df5_dx5 df5_dx6 0 0;
        df6_dx1 df6_dx2 df6_dx3 df6_dx4 df6_dx5 df6_dx6 0 0;
        df7_dx1 0 df7_dx3 0 0 0 0 0;
        df8_dx1 df8_dx2 df8_dx3 df8_dx4 df8_dx5 df8_dx6 0 0];

    B = [0 0; 0 0; 0 0; 0 0; 0 df5_du2; 0 df6_du2; df7_du1 0; 0
df8_du2];
```