XXXVIII Jornadas de Automática
https://doi.org/10.17979/spudc.9788497497749.0360

Aprendizaje por Refuerzo para sistemas lineales discretos con dinámica desconocida: Simulación y Aplicación a un Sistema Electromecánico

Henry Díaz, Leopoldo Armesto, Antonio Sala
{hendia@posgrado larmesto@idf asala@isa}.upv.es
Universitat Politècnica de València,
C/Camino de Vera s/n, 46022, Valencia, España

Resumen

El aprendizaje por refuerzo es una técnica que se utiliza en la búsqueda de soluciones en sistemas de decisión secuencial. Una gran parte de los algoritmos usados en el aprendizaje por refuerzo se fundamentan en la programación dinámica, se considera que el aprendizaje por refuerzo es una extensión de la programación dinámica que proporciona soluciones sin la necesidad de conocer el modelo de comportamiento del sistema. Estas técnicas combinan algunas características del control óptimo y control adaptativo para el diseño de controladores realimentados. Se describen los algoritmos básicos del aprendizaje por refuerzo para la implementación de soluciones en sistemas discretos deterministas. Finalmente, se realizaron pruebas prácticas de la implementación del algoritmo de aprendizaje Q-Learning en un péndulo de un grado de libertad, con el objetivo de verificar si el algoritmo de aprendizaje converge y proporciona un controlador estabilizante.

Palabras clave Aprendizaje por refuerzo, Q-*Learning*, control óptimo, control adaptativo óptimo, programación dinámica .

1. Introducción

El aprendizaje por refuerzo (RL, *reinforcement learning*) es un conjunto de técnicas para resolver problemas de decisión secuenciales, en los cuales las decisiones son aplicadas al sistema con el objetivo de obtener una respuesta deseada[12][16]. Este tipo de problemas secuenciales aparecen en una amplia variedad de campos entre los que podemos mencionar el control automático, inteligencia artificial, robótica, control de procesos, entre otras[10]. El aprendizaje por refuerzo a diferencia de las técnicas de programación dinámica(DP, *dynamic programming*) no requiere del conocimiento del modelo[2][3]. Debido a los múltiples orígenes del RL es común encontrar en la literatura los mismos conceptos con diferentes definiciones, por ejemplo: programación neuro-dinámica[4], programación dinámica aproximada[10], programación dinámica adaptativa[9] entre otros.

Los principales elementos y su interacción del problema a resolver en la DP y RL son representados en la Figura 1. Se muestra la manera en la que un agente interactúa con el entorno mediante tres señales: una señal de estado del entorno, una señal de acción que permite al agente influenciar el estado del entorno y una señal escalar de recompensa, la cual proporciona al agente información sobre la calidad de la acción que acaba de realizar en el estado actual. En cada instante temporal, el agente recibe una medida del estado y realiza una acción. Como consecuencia de la acción realizada se produce una transición del entorno a un nuevo estado. Además se genera una señal de recompensa que evalúa la calidad de dicha transición. Entonces el agente recibe el nuevo estado y el ciclo completo se repite [12].

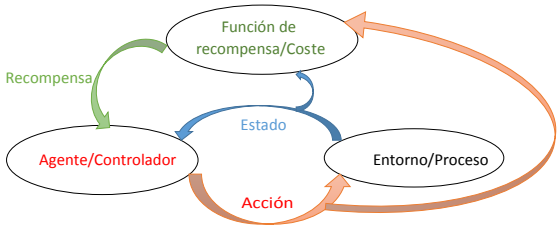

Figura 1: Elementos de la DP y RL y su flujo de interacción[5].

El agente selecciona la acción realizada en cada estado de acuerdo a una *política*. La política es una función que mapea los estados a acciones. El objetivo del agente es aprender una política que maximice la cantidad total de recompensa recibida, es decir, la recompensa acumulada a largo plazo[12][4][5].

Considerando las características que presentan los controladores proporcionados por las técnicas de RL se analizan e implementan algoritmos para sistemas discretos lineales de los cuales se desconoce su comportamiento dinámico[6][7].

El paper está estructurado de la siguiente forma: en la sección 2 se describen los conceptos básicos de la programación dinámica y el aprendizaje

por refuerzo. La sección 3 describe el aprendizaje por refuerzo para sistemas lineales discretos. En la sección 4 se realiza la simulación del aprendizaje en sistemas lineales. En la sección 5 se describe la aplicación del algoritmo de aprendizaje Q-*Learning* a un péndulo de un grado de libertad y se finaliza con la sección 6 de conclusiones.

2. Programación dinámica y aprendizaje por refuerzo

La programación dinámica es una parte fundamental de la teoría de control óptimo. En un problema de control óptimo, el objetivo es desarrollar un controlador que minimice una medida del comportamiento de un sistema dinámico a lo largo del tiempo [4], esta medida de comportamiento es evaluada con un índice de costo o función de valor y este índice o valor puede ser definido en términos de objetivos de optimalidad [8].

El RL tiene sus orígenes en el campo de la inteligencia artificial y se encuentra inspirado en los mecanismos de aprendizaje biológico. Específicamente, tiene sus raíces en el condicionamiento operante entre las diferentes formas que un individuo puede responder ante una misma situación, aquellas que estén acompañadas de una satisfacción(refuerzo positivo), estarán más firmemente conectadas a dicha situación de repetirse.

Los algoritmos DP para encontrar una política óptima requieren de un modelo MDP *Markov decision processes* incluyendo la dinámica de transición y la función de refuerzo[4], en general muchos problemas de decisión toman como marco de referencia los MDP incluidos los sistemas de control realimentados [7], un estudio detallado y amplio sobre MDP se puede consultar en [11].

Los algoritmos de RL son libres de modelo [12][4] lo que les hace muy útiles cuando la obtención del modelo de un proceso es demasiado dificultosa o muy costosa de ser implementada. Estos algoritmos usan datos obtenidos del proceso, estos datos pueden ser un conjunto de trayectorias, una simple trayectoria o un conjunto de muestras, lo que implica trabajar con un número limitado de datos que provienen proceso. Mientras los algoritmos de DP pueden usar el modelo para obtener cualquier número de muestras de transición de cualquier par estado-acción.

2.1. Criterios de optimalidad

El objetivo de los algoritmos de DP/RL es encontrar una política que maximice la recompensa obtenida por el agente a lo largo del tiempo. Escoger entre los criterios de optimalidad está relacionado con el problema del aprendizaje. La mayor parte de los algoritmos de DP y RL emplean el criterio de optimalidad de horizonte infinito descontado (ecuación 1) debido a que posee propiedades teóricas que lo hacen más adecuado para el análisis matemático[4].

$$\sum_{t=0}^{\infty} \gamma^t r_t \tag{1}$$

Donde r_t es el refuerzo instantáneo y $\gamma \in [0, 1)$ es el factor de descuento.

2.2. Función de valor y ecuaciones de Bellman

Las funciones de valor son el punto de unión entre el sistema y el criterio de optimalidad. Una *función de valor* es una estimación de la bondad que supone para un agente estar en un determinado estado cuando se sigue una política fija. Existen dos tipos de funciones de valor: función V, que estima la bondad de estar en un estado, y función Q que estima la bondad de realizar una acción en un estado. Usando el modelo de horizonte infinito con descuento la función de valor puede ser expresada así:

$$V^{\pi}(x) = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{2}$$

Una función de valor estado-acción similar:

$$Q^{\pi}(x, u) = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{3}$$

Una de las características fundamentales de las funciones de valor es que satisfacen ciertas propiedades recursivas. Para cualquier política π y cualquier estado x la expresión en la ecuación 2 puede ser definida recursivamente en términos de la llamada *ecuación de Bellman*[2].

$$\begin{aligned} V^{\pi}(x) &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... \\ &= r_t + \gamma V^{\pi}(x_{t+1}) \end{aligned} \tag{4}$$

La meta buscada es encontrar la *mejor* política, por ejemplo la que reciba el mayor retorno. Esto significa maximizar la ecuación 2 para todos los estados $x \in X$. Una *política óptima*, denotada π^*, es tal que $V^{\pi^*}(x) \geq V^{\pi}(x)$ para todo $x \in X$ y todas las políticas π. Se puede demostrar que la solución óptima $V^* = V^{\pi^*}$ satisface las siguiente ecuación:

$$V^*(x) = \max_u [r(x, u) + \gamma V^*(x')] \tag{5}$$

Esta ecuación se denomina la *Ecuación de optimalidad de Bellman*. Y establece que el valor de un estado bajo una política óptima debe ser igual

al retorno esperado para la mejor acción en ese estado. Para seleccionar una acción óptima dada la función de valor óptima V^* se puede aplicar la siguiente regla:

$$\pi^*(x) = \underset{u}{\text{argmáx}}[r(x,u) + \gamma V^\pi(x')] \qquad (6)$$

La denominación de esta política es *política voraz (greedy policy)*, se denota $\pi_{greedy}(V)$. Esta política selecciona la mejor acción usando la función de valor V. Análogamente el valor óptimo estado-acción es:

$$Q^*(x,u) = r(x,u) + \gamma \underset{u'}{\text{máx}}\, Q^*(x',u') \qquad (7)$$

Las *Q-funciones* son muy útiles debido a que hacen innecesaria la suma ponderada sobre las diferentes alternativas usando la función de transición. Esa es la razón por la cual en el enfoque libre de modelo donde no se conoce la función de transición ni la función de recompensa son aprendidas en lugar de las V-funciones. La relación entre Q^* y V^* esta dada por:

$$V^*(x) = \underset{u}{\text{máx}}\, Q^*(x,u) \qquad (8)$$

La selección de la acción óptima esta dada por:

$$\pi^*(x) = \underset{u}{\text{argmáx}}\, Q^*(x,u) \qquad (9)$$

Es decir, la mejor acción es la acción que tiene la mayor utilidad esperada sobre la base de posibles estados próximos resultantes de tomar esa acción. Los algoritmos de DP y RL de a acuerdo a como se obtienen la política óptima se clasifican en algoritmos de Iteración de función de valor (VI, *Value Iteration*), buscan el valor óptimo de la función de valor, que consiste en el máximo refuerzo de cada estado o de cada par estado-acción. Algoritmos de Iteración de política (PI, *Policy Iteration*), evalúan las políticas a través de construir sus funciones de valor (en lugar de la función de valor óptima), y utilizan estas funciones de valor para hallar nuevas y mejores políticas.

Hay varios métodos para la implementación de los algoritmos VI y PI. Los tres prinicipales son: cálculo exacto, métodos de Monte Carlo y aprendizaje por diferencias temporales(TD, *Temporal difference*)[4][12][10]. Los dos últimos métodos pueden ser implementados sin el conocimiento de la dinámica del sistema, el método de diferencias temporales es el que se toma como referencia en las secciones posteriores.

2.2.1. *Temporal difference*

Temporal difference hace referencia a una familia de métodos para estimar, o predecir, la función V de una política fija, aunque como veremos en secciones posteriores, el concepto del aprendizaje TD puede ser extendido al caso de funciones Q[12]. En los métodos TD la función V se estima en base a otras estimaciones previas, técnica que recibe el nombre de *bootstraping*[12]. Cada vez que el agente realiza una acción el algoritmo TD utiliza la recompensa generada y la estimación actual de V para realizar una nueva estimación de acuerdo a la expresión:

$$\begin{aligned} V_{k+1}(x_k) = V_k(x_k) + \\ \alpha_k \left[r_{k+1} + \gamma V_k(x_{k+1}) - V_k(x_k) \right] \end{aligned} \qquad (10)$$

donde $\alpha_k \in [0,1]$ es la secuencia de tasas de aprendizaje que determina la cantidad con la que se actualiza el valor del estado x_k. El término entre corchetes, se conoce como diferencia temporal y da nombre al método, es la diferencia entre la nueva estimación de la función V, $r_{k+1} + \gamma V_k(x_{k+1})$, y la estimación en el instante temporal anterior, $V_k(x_k)$.

3. Aprendizaje por refuerzo y control adaptativo óptimo para sistemas lineales en tiempo discreto

El análisis físico de los sistemas utilizando por ejemplo la mecánica lagragiana o la mecánica hamiltoniana que son una reformulación de la mecánica clásica proporcionan descripciones de los sistemas en términos de ecuaciones diferenciales ordinarias no lineales. Discretizando obtenemos una representación de los sistemas en ecuaciones en diferencias[8].

Considerando un sistema discreto representado por las siguiente ecuación en diferencias

$$x_{k+1} = f(x_k) + g(x_k)u_k \qquad (11)$$

donde el estado $x_k \in R^n$ y la acción de control $u_k \in R^m$. Una política de control esta definida como una función del espacio de estados al espacio de control $h(.) : R^n \to R^m$. Lo que significa que por cada estado se define una acción de control dada por

$$u_k = h(x_k) \qquad (12)$$

Una política es simplemente un controlador realimentado. A partir de definir una función costo se obtiene la función de valor[7][6][8].

$$V^h(x_k) = \sum_{i=k}^{\infty} \gamma^{i-k}(x_i^T H_x x_i + u_i^T H_u u_i) \qquad (13)$$

con un factor de descuento $0 < \gamma \leq 1$, $H_x \in \mathbb{R}^{n_x \times n_x}$ y $H_u \in \mathbb{R}^{n_u \times n_u}$, son las matrices de pesos de la función de costo cuadrático y $u_k = h(x_k)$ una política de control realimentada. El costo de cada etapa

$$r(x_k, u_k) = x_k^T H_x x_k + u_k^T H_u u_k \qquad (14)$$

es considerado como cuadrático en u_k para simplificar el desarrollo, pero puede ser cualquier función de control definida positiva. Se asume que el sistema es estabilizante en un conjunto $\Omega \in R^n$, lo que significa que existe una política de control $u_k = h(x_k)$ que el sistema en lazo cerrado $x_{k+1} = f(x_k) + g(x_k)h(x_k)$ es asintóticamente estable en Ω. Una política se dice que es *admisible* si esta es estabilizante y proporciona un costo finito $V^h(x_k)$ para la trayectorias en Ω[8][1]. Para sistemas determinísticos discretos, el valor óptimo esta dado por la ecuación de optimalidad de Bellman

$$V^*(x_k) = \min_{h(.)}(r(x_k, h(x_k)) + \gamma V^*(x_{k+1})) \quad (15)$$

Que es justamente la ecuación Hamilton-Jacobi-Bellman(HJB)[8] en tiempo discreto. Y tenemos que la política óptima es

$$h^*(x_k) = \operatorname{argmin}_{h(.)}(r(x_k, h(x_k)) + \gamma V^*(x_{k+1}))) \quad (16)$$

Para el regulador lineal cuadrático para sistemas discretos(DT LQR) tenemos,

$$x_{k+1} = Ax_k + Bu_k \quad (17)$$

$$V^h(x_k) = \frac{1}{2}\sum_{i=k}^{\infty} \gamma^{i-k}(x_i^T H_x x_i + u_i^T H_u u_i) \quad (18)$$

Notar que desde el punto de vista de los sistemas de control el objetivo que busca el aprendizaje por refuerzo es encontrar una política óptima que minimice el coste acumulado.

Iteración de política(PI) usando aprendizaje por diferencias temporales[7]

Inicialización

Seleccionar cualquier política de control admisible $h_0(x_k)$.

Hacer para $j = 0$ hasta converger

Evaluación de Política

$$V_{j+1}(x_k) = r(x_k, h_j(x_k)) + \gamma V_{j+1}(x_{k+1}) \quad (19)$$

Mejora de Política

$$h_{j+1}(x_k) = \operatorname{argmin}_{h(.)}(r(x_k, h(x_k)) + \gamma V_{j+1}(x_{k+1})) \quad (20)$$

o

$$h_{j+1}(x_k) = -\frac{\gamma}{2}R^{-1}g^T(x_k)\nabla V_{j+1}(x_{k+1}) \quad (21)$$

donde $\nabla V(x) = \delta V(x)/\delta x$ es el gradiente de la función de valor , interpretado aquí como un vector columna. En el método de iteración de la función de valor se realiza de manera similar, pero el procedimiento de evaluación de la política se realiza de la siguiente forma.

Iteración de la función de valor(VI) usando aprendizaje por diferencias temporales[7]
Actualización de la función valor en cada paso

Actualización del valor usando

$$V_{j+1}(x_k) = r(x_k, h_j(x_k)) + \gamma V_j(x_{k+1}) \quad (22)$$

En VI se puede seleccionar cualquier política de control inicial $h_0(x_k)$, no necesariamente admisible o estabilizante.

3.1. Aproximación de la función de valor

Para implementaciones prácticas de PI y VI para sistemas dinámicos con infinitos espacios de estado y de acciones es aproximar la función de valor usando una estructura de un aproximador adecuado en términos de parámetros desconocidos[6]. Así, los parámetros desconocidos son ajustados en línea exactamente como en un sistema de identificación. Esta idea de la *aproximación de la función de valor* (VFA) fue usada por Werbos[15][14] y llamada programación dinámica aproximada(ADP) o programación dinámica adaptativa. Esta fue usada por Bertsekas y Tsitsiklis[4] y la llamó programación neurodinámica.

En el caso del LQR es conocido que el valor es cuadrático en el estado para alguna matriz *kernel* P[8].

$$V(x_k) = \frac{1}{2}x_k^T P x_k = \frac{1}{2}(vec(P))^T(x_k \otimes x_k) \equiv \overline{p}^T\phi(x_k) \quad (23)$$

El producto de Kronecker $\otimes$ permite escribir esta forma cuadrática como una lineal en vector de parámetros $\overline{p} = vec(P)$, que se forma apilando la columnas de la matriz P[7]. El vector $\phi(x_k) = \overline{x}_k = x_k \otimes x_k$ es el vector polinomial cuadrático que contiene todos los posibles pares de productos de n componentes de x_k. Notar que P es simétrica y tiene solamente $n(n+1)/2$ elementos independientes, removiendo los términos redundantes en $x_k \otimes x_k$ para definir un conjunto de base cuadrática $\phi(x_k)$ con $n(n+1)/2$ elementos independientes[7][6].

Se asume que la ecuación de Bellman tiene una solución local suave.

$$V(x) = \sum_{i=1}^{\infty} w_i\varphi_i(x) \equiv W^t\phi(x) + \varepsilon_L(x) \quad (24)$$

donde el vector de base $\phi(x) = [\varphi_1(x) \ \varphi_2(x) \ ... \ \varphi_L(x)] : R^n \to R^L$ y $\varepsilon_L(x)$ converge uniformemente a cero mientras el número de términos $L \to \infty$[7][5][3].

3.2. Control adaptativo óptimo en sistemas lineales discretos con dinámica desconocida

El método de RL Q-*learning* proporciona un algoritmo de control adaptativo que converge en línea a la solución de control óptima para sistemas en los que se desconoce completamente su dinámica. Este método resuelve la ecuación de Bellman y las ecuación HJB en tiempo real a través de la medición de datos a lo largo de las trayectorias del sistema, sin conocer la dinámica $f(x_k), g(xk)$[7].

Q-*learning*[13] es un método simple de RL que trabaja para sistemas desconocidos, esto es, para sistemas los cuales se desconoce completamente su dinámica. Q-*learning* aprende la función Q usado el método de diferencias temporales(TD) y realizando una acción u_k y midiendo en cada etapa el resultado del conjunto de datos de experiencia (x_k, x_{k+1}, r_k) consistentemente en el estado actual, el estado próximo y el costo resultante[8].

El algortimo Q-*Learning* puede ser fácilmente desarrollado para sistemas dinámicos discretos usando aproximaciones de la función Q, en [8] se desarrolla y se muestra las principales ecuaciones para el Q-*Learning* en sistemas discretos.

Tomando como referencia [7], tenemos que para un sistema no lineal la función Q es parametrizada como

$$Q(x,u) = W^T\phi(z)$$

para algún vector de parámetros desconocido W y un conjunto de vectores base $\phi(z)$. Para un DT LQR, $\phi(z)$ es un conjunto base cuadrático formado por componentes de estado y entrada. Por lo tanto, el error TD es

$$e_k = -W^T\phi(z_k) + r(x_k,u_k) + \gamma W^T\phi(z_{k+1}) \quad (25)$$

sobre el cual los algoritmos PI y VI pueden basarse. Considerando el algoritmo PI el paso de evaluación de una función Q es

$$W_{j+1}^T(\phi(z_k) - \gamma\phi(z_{k+1})) = r(x_k, h_j(x_k)) \quad (26)$$

y el paso de mejora de la política es

$$h_{j+1}(x_k) = \underset{u}{\text{argmín}}(W_{j+1}^T\phi(x_k,u)), x \in X \quad (27)$$

Q-*learning* usando VI esta dado por

$$W_{j+1}^T\phi(z_k) = r(x_k, h_j(x_k)) - \gamma W_j^T\phi(z_{k+1}) \quad (28)$$

y la ecuación 27. Estas ecuaciones no requieren conocimiento de la dinámica $f(.), g(.)$.

Para implementaciones en línea, para resolver la ecuación 26 se puede usar LS por lotes o RLS para el vector de parámetros W_{j+1} obteniendo el vector de regresión $\phi(z_k) - \gamma\phi(z_{k+1})$, o en 28 usando el vector de regresión $\phi(z_k)$. Los datos observados en cada instante de tiempo son $(z_k, z_{k+1}, r(x_k,u_k))$ con $z_k \equiv [x_k^T u_k^T]^T$, con $u_{k+1} = h_j(x_{k+1})$ y $h_j(x_k)$ la política actual. Se debe agregar ruido de exploración a la entrada de control para obtener una excitación persistente.

Después de la convergencia de los parámetros de la función Q, la actualización de la acción es realizada. Esto se realiza fácilmente sin conocer la dinámica del sistema debido a que la función Q contiene u_k como uno de sus argumentos así que $\partial(W_{j+1}^T\phi(x_k,u))/\partial u$ puede ser explícitamente calculada.

$$\frac{\partial Q(x,u)}{\partial u} = (\frac{\partial z}{\partial u})^T(\frac{\partial\phi(z)}{\partial z})^T$$

$$W = [0_{m,n} \quad I_m]\nabla\phi^T W$$

donde $0_{m,n} \in R^{m\times n}$ es una matriz de ceros. El vector de base $\phi(z) = z \otimes z \in R^{n+m^2}$ es el vector polinomial cuadrático que contiene todos lo posibles pares de productos de los $n+m$ componentes de z. Se define $N = n+m$, entonces

$$\nabla\phi^T = \frac{\partial\phi^T}{\partial z} = (I_N \otimes z + z \otimes I_N)^T \in R^{N\times N^2} \quad (29)$$

4. Simulaciones implementadas

Las simulaciones realizadas en esta sección consideran sistemas discretos lineales en los cuales la dinámica del sistema no es conocida.

4.1. Aprendizaje por refuerzo para un Sistema lineal discreto de segundo orden usando PI(*Policy Iteration*).

En esta simulación se muestra el uso del algoritmo PI para resolver la DT ARE sin conocer la dinámica del sistema, es decir, para el caso del LQR se desconoce las matrices A y B del sistema. Las matrices solo se usan para generar las trayectorias y adquirir los datos que el algoritmo requiere. Por lo tanto, el modelo del sistema a considerar aquí es $x_{k+1} = Ax_k + Bu_k$, donde

$$A = \begin{bmatrix} 0.9039 & -0.1903 \\ 0.0095 & 0.9990 \end{bmatrix}, B = \begin{bmatrix} 0.0095 \\ 0 \end{bmatrix}$$

que representa un modelo discretizado de un sistema sencillo masa-muelle-amortiguado Figura 2. La solución de DT ARE con los pesos de la función de coste $H_x = diag(1\ 1)$, $H_u = 1$ y $\gamma = 1$ es

$$P_{DARE} = \begin{bmatrix} 5.7529 & 2.5143 \\ 2.5143 & 130.338 \end{bmatrix}$$

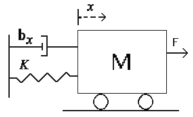

Figura 2: Sistema de masa, muelle y amortiguador

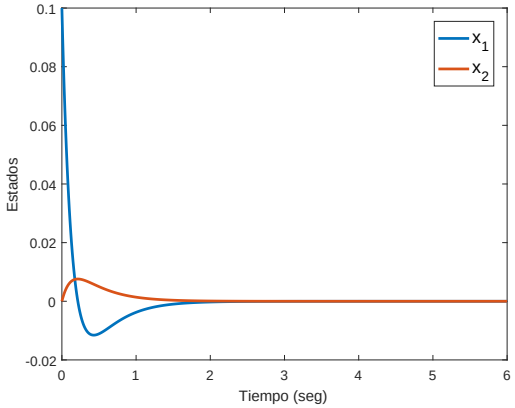

Figura 3: Variables de estado del sistema simulado

Definimos la aproximación de la función de valor considerando un modelo de coste cuadrático en la acción de control:

$$Q(x_k, u_k) = W^T \phi(x_k, u_k)$$

$$\phi(x_k, u_k) = [x_{k1}^2 \quad x_{k1}x_{k2} \quad x_{k1}u_k \quad x_{k2}^2 \quad x_{k2}u_k \quad u_k^2]^T$$

$$W = [w_1 \quad w_2 \quad w_3 \quad w_4 \quad w_5 \quad w_6]^T$$

La acción de control óptima es:

$$u_k^* = -\frac{1}{2} w_6^{-1} [w_3 \quad w_5][x_{k1} \quad x_{k2}]^T$$

La implementación online de PI se realizó usando el método de mínimos cuadrados recursivos (*Recursive-Least-Squares,RLS*). Las trayectorias de los estados se muestran en la Figura 3, donde se evidencia como los estados son regulados a cero como es deseable. Los valores finales de la matriz P son:

$$P_{RL} = \begin{bmatrix} 5.7529 & 2.5143 \\ 2.5143 & 130.3380 \end{bmatrix}$$

El algoritmo implementado es un algoritmo adaptativo de control que identifica la función Q a través de técnicas RLS. Para su implementación no se requiere de las matrices de la dinámica del sistema(A, B). El algoritmo efectivamente resuelve la ecuación algebraica de Riccati en línea a tiempo real usando datos $(x_k, u_k, x_{k+1}, u_{k+1})$ medidos en tiempo real en cada instante k. Es necesario agregar ruido de exploración a la señal de control para garantizar una excitación persistente para lograr la convergencia usando RLS.

4.2. Aprendizaje por refuerzo para un sistema lineal discreto usando VI(*Value Iteration*)[7].

En esta simulación se muestra el uso del algoritmo VI para resolver la DT ARE sin conocer la dinámica del sistema. Un modelo lineal puede ser usado par representar el sistema dinámico alrededor de un punto de operación específico con una carga de valor constante. El problema se incrementa con el hecho de que los parámetros de la planta no son conocidos, mientras lo que se busca es una solución de control óptima. El modelo del sistema a considerar aquí es $\dot{x} = Ax + Bu$. Los parámetros del sistema en tiempo continuo se seleccionar aleatoriamente con rango de operación específicos, tenemos

$$A = \begin{bmatrix} -0.0665 & 8 & 0 & 0 \\ 0 & -3.663 & 3.663 & 0 \\ -6.8681 & 0 & -13.7363 & -13.7363 \\ 0.6 & 0 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 0 \\ 13.7355 \\ 0 \end{bmatrix}$$

El sistema se ha discretizado con un periodo de muestreo de $T = 0.01s$ y los pesos de la función de costo: $H_x = I$, $H_u = I$, y $\gamma = 1$. La solución es:

$$P_{DARE} = \begin{bmatrix} 0.4805 & 0.4772 & 0.0604 & 0.4771 \\ 0.4772 & 0.7892 & 0.1240 & 0.3855 \\ 0.0604 & 0.1240 & 0.0567 & 0.0304 \\ 0.4771 & 0.3855 & 0.0304 & 2.3509 \end{bmatrix}$$

La implementación de VI se realizó usando mínimos cuadrados por lotes, es así que la ecuación de Riccati es resuelta con los datos $(x_k, x_{k+1}, r(x_k, u_k))$. La aproximación de la función de valor es:

$$Q(x_k, u_k) = W^T \phi(x_k, u_k)$$

$$\phi(x_k, u_k) = [x_{k1}^2, x_{k1}x_{k2}, x_{k1}x_{k3}, x_{k1}x_{k4}, x_{k1}u_k, x_{k2}^2, x_{k2}x_{k3}, x_{k2}x_{k4}, x_{k2}u_k, x_{k3}^2, x_{k3}x_{k4}, x_{k3}u_k, x_{k4}^2, x_{k4}u_k, u_k^2]^T$$

$$W = [w_1 \quad w_2 \quad w_3 \quad ... \quad w_{14} \quad w_{15}]^T$$

La acción de control óptima es:

$$u_k^* = -\frac{1}{2} w_{15}^{-1} [w_5 \quad w_9 \quad w_{12} \quad w_{14}][x_{k1} \quad x_{k2} \quad x_{k3} \quad x_{k4}]^T$$

Las trayectorias de los estados del sistemas se muestran en la Figura 4, donde se observa que los estados son regulados a cero. Los valores finales de los parámetros estimados para P son

$$P_{RL} = \begin{bmatrix} 0.4753 & 0.4770 & 0.0602 & 0.4769 \\ 0.4770 & 0.7837 & 0.1238 & 0.3852 \\ 0.0602 & 0.1238 & 0.0513 & 0.0302 \\ 0.4769 & 0.3852 & 0.0302 & 2.3457 \end{bmatrix}$$

La Figura 5 muestra la convergencia de P_k a sus valores óptimos P^* durante el proceso de aprendizaje. Para su implementación no se requiere de las matrices de la dinámica del sistema(A, B). El

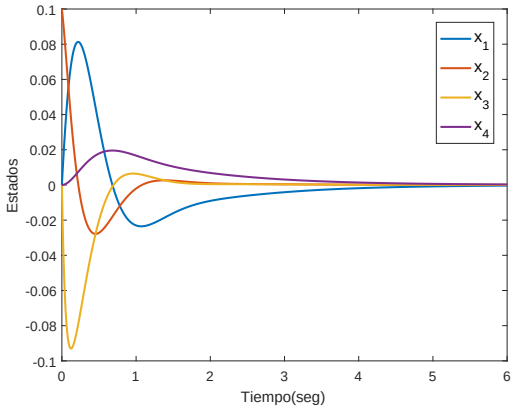

Figura 4: Variables de estado del sistema simulado

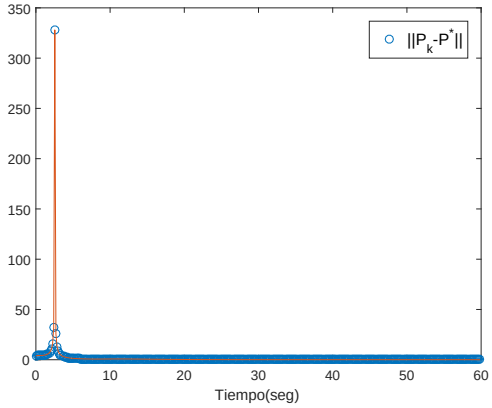

Figura 5: Convergencia de P_k a su valor óptimo P^*

algoritmo efectivamente resuelve la ecuación algebraica de Riccati en línea a tiempo real usando datos $(x_k, u_k, x_{k+1}, u_{k+1})$ medidos en tiempo real en cada instante k. Es necesario agregar ruido de exploración a la señal de control para garantizar una excitación persistente para lograr la convergencia usando LS.

5. Experimentación real: Péndulo de un grado de libertad(1DoF)

En esta sección se describe la implementación práctica del algoritmo de aprendizaje Q-*Learning* en un péndulo de un grado de libertad, con el objetivo de verificar si el algoritmo de aprendizaje nos proporciona una ganancia estabilizante, considerando que no se ha tomado en cuenta para la función de coste las no linealidades existentes en experimentos prácticos. Las implementaciones reales de los algoritmos se han hecho sobre un banco de experimentos mostrado en la Figura 6. El banco de pruebas esta formado por un péndulo de 1DoF cuyo actuador es un motor DC, el péndulo posee un sensor de posición de efecto hall en el primer eslabón. El experimento diseñado se ha definido

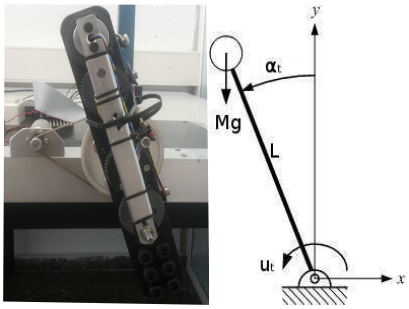

Figura 6: Sistema implementado para pruebas de aprendizaje Q-*Learning*.

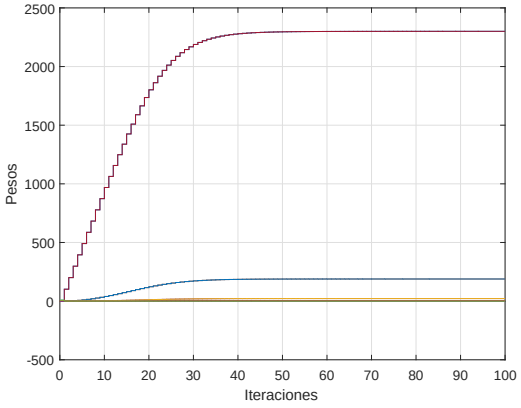

Figura 7: Convergencia de la matriz de parámetros del aprendizaje usando VI.

como la búsqueda del controlador con realimentación del estado para una trayectoria del péndulo desde una posición inicial de $-\frac{\pi}{3}[rad]$ hasta su posición arriba $0[rad]$. En la etapa de exploración se adquirieron 3000 muestras a un periodo de adquisición de $T = 10ms$. Los datos obtenidos en la exploración ingresan al algoritmo de aprendizaje implementado Q-*Learning*, el factor de descuento es $\gamma = 0.98$ y las matrices de ponderación del índice de coste usadas son:

$$H_x = \begin{bmatrix} 100 & 0 \\ 0 & 0.1 \end{bmatrix}, H_u = 0.1$$

La aproximación de la función de valor es $Q(x_k, u_k) = W^T \phi(x_k, u_k)$ y la Figura 7 muestra la convergencia del vector de pesos W. La Figura 8 muestra la evolución de la posición del péndulo usando el controlador aprendido y comparándolo con un controlador proporcional-derivativo(PD).

6. Conclusiones

El aprendizaje por refuerzo proporcionan soluciones a problemas de decisión secuencial, los cuales aparecen en una amplia variedad de campos entre ellos el de los sistemas de control. Gran parte de los algoritmos de aprendizaje por refuerzo se basan en las técnicas de programación dinámica, y la

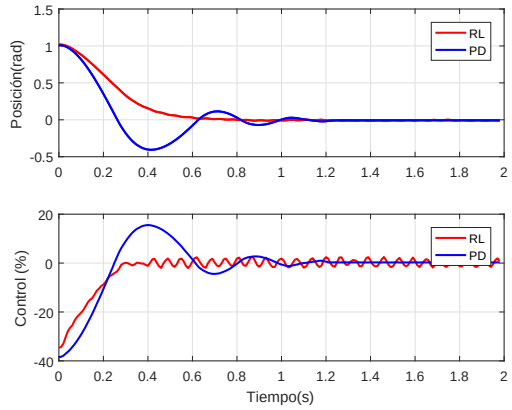

Figura 8: Evolución de la posición del sistema y la acción de control del controlador Q-*Learning* y de un controlador PD implementado desde la condición inicial $x_0 = [-\frac{\pi}{3} \quad 0]^T$.

diferencia fundamental yace en que la DP requiere de un modelo de comportamiento mientras que RL no requiere del conocimiento del modelo para proporcionar una solución, específicamente en el caso de los sistemas de control proporcionan un controlador estabilizante y óptimo. El aprendizaje por refuerzo nos permite diseñar controladores adaptativos que convergen a soluciones óptimas usando los datos medidos a lo largo de las trayectorias del sistema. Las simulaciones también permite concluir que la técnica de aprendizaje Q-*Learning* resuelve la ecuación de *Riccati*, en nuestro caso en las simulaciones de manera *online* y en el experimento real de manera *offline*, sin el conocimiento del comportamiento dinámico del sistema, simplemente observando los datos medidos(exploración) a lo largo de las trayectorias del sistema.

7. Agradecimientos

Agradecemos al Ministerio de Economía de España,la Unión Europea DPI2016-81002-R (AEI/FEDER, UE), y al Gobierno de Ecuador (Beca SENESCYT).

Referencias

[1] Asma Al-Tamimi, Frank L Lewis, and Murad Abu-Khalaf. Model-free q-learning designs for linear discrete-time zero-sum games with application to h-infinity control. *Automatica*, 43(3):473–481, 2007.

[2] Richard Bellman. *Dynamic programming*. Courier Corporation, 2013.

[3] Dimitri P Bertsekas and Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena Scientific Belmont, MA, 1995.

[4] Dimitri P Bertsekas and Bertsekas. *Neuro-Dynamic Programming*, volume 1. Athena Scientific, 1996.

[5] Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement learning and dynamic programming using function approximators*, volume 39. CRC press, 2010.

[6] Frank L Lewis and Draguna Vrabie. Reinforcement learning and adaptive dynamic programming for feedback control. *Circuits and Systems Magazine, IEEE*, 9(3):32–50, 2009.

[7] Frank L Lewis, Draguna Vrabie, and Kyriakos G Vamvoudakis. Reinforcement learning and feedback control: Using natural decision methods to design optimal adaptive controllers. *Control Systems, IEEE*, 32(6):76–105, 2012.

[8] Vrabie D. L. Lewis, F. L. and V. L. Syrmos. *Optimal control*. John Wiley & Sons, 2012.

[9] John J Murray, Chadwick J Cox, George G Lendaris, and Richard Saeks. Adaptive dynamic programming. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 32(2):140–153, 2002.

[10] Warren B Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*, volume 703. John Wiley & Sons, 2011.

[11] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[12] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[13] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[14] Paul J Werbos. A menu of designs for reinforcement learning over time. *Neural networks for control*, pages 67–95, 1990.

[15] Paul J Werbos. Approximate dynamic programming for real-time control and neural modeling. *Handbook of intelligent control: Neural, fuzzy, and adaptive approaches*, 15:493–525, 1992.

[16] Marco Wiering and Martijn van Otterlo. *Reinforcement Learning: State-of-the-art*, volume 12. Springer Science & Business Media, 2012.