XLIV Jornadas de Automática 2023

https://jautomatica.es

Analisis de un robot abierto de bajo coste para docencia de aprendizaje automático

Bes, J., Garcia-Barcos, J., Martinez-Cantin, R.*

Instituto de Investigación en Ingeniería de Aragón, Universidad de Zaragoza, C. de Mariano Esquillor Gómez, s/n, 50018 Zaragoza, España.

To cite this article: Bes, J., Garcia-Barcos, J., Martinez-Cantin, R. 2023. Analysis of a low-cost open robot for machine learning teaching.
XLIV Jornadas de Automática, 198-203. https://doi.org/10.17979/spudc.9788497498609.198

Resumen

En este trabajo estudiamos el comportamiento, ventajas e inconvenientes del robot Trifinger para la docencia de estudiantes de ingeniería y automática. El robot Trifinger es un robot de diseño abierto a nivel hardware con piezas realizadas con impresión 3D y componentes (electrónica, motores...) estándar y disponibles comercialmente. El software también es abierto e incorpora un simulador realizado en Pybullet que simplifica el prototipado de soluciones y puede usarse como una primera aproximación de los estudiantes. En concreto, en este trabajo evaluamos su potencial uso para la docencia de aprendizaje automático en general y de aprendizaje por refuerzo en particular, una línea que esta permeando todas las áreas de conocimiento y tiene gran potencial en el mundo de la automática. Evaluamos el comportamiento de varios algoritmos que están en el estado del arte del aprendizaje por refuerzo en varias tareas planteadas con el uso robot, demostrando que la plataforma es a la vez una herramienta sencilla de usar y manejar, pero a la vez desafiante y compleja incluso para el estado del arte.

Palabras clave: Aprendizaje automático, Aprendizaje por refuerzo y aprendizaje profundo en control, Robots manipuladores, Robótica inteligente, Aprendizaje automático en modelado, predicción, control y automatización.

Analysis of a low-cost open robot for machine learning teaching

Abstract

In this work we study the behavior, advantages and disadvantages of the Trifinger robot for teaching engineering and automation students. The Trifinger robot is an open design robot at hardware level with parts made with 3D printing and off-the-shelf components (electronics, motors...). The software is also open and incorporates a simulator made in Pybullet that simplifies the prototyping of solutions and can be used as a first approach for students. Specifically, in this work we evaluate its potential use for teaching machine learning in general and reinforcement learning in particular, a line that is permeating all areas of knowledge and has great potential in the world of automation. We evaluate the behavior of several algorithms that are in the state of the art of reinforcement learning in several tasks posed with the use of robots, demonstrating that the platform is both a simple tool to use and manage, but at the same time challenging and complex even for the state of the art.

Keywords: Machine learning, Reinforcement learning and deep learning in control, Robots manipulators, Intelligent robotics, Machine learning in modelling, prediction, control and automation

1. Introducción

El uso de robots como herramienta en entornos educativos son idóneos para fomentar un aprendizaje activo de los estudiantes y educar en temas de ciencia y tecnología y, más en concreto, permite enseñar conceptos de programación, control, electrónica y aprendizaje automático. Sin embargo, el uso de robots para prácticas está bastante restringido por su alto coste y la complejidad del mantenimiento, impidiendo su adopción en instituciones de distintos niveles educativos.

Los avances en impresión 3D de los últimos años han per-

*Autor para correspondencia: rmcantin@unizar.es

mitido la aparición de proyectos y plataformas robóticas basadas en impresión 3D, las cuales se basan en diseños abiertos y materiales de bajo coste que otorga un balance entre un coste económico y prestaciones del robot. A pesar de las limitaciones de los materiales, los robots impresos en 3D pueden resultar viables como plataforma de investigación y docencia. Además de las ventajas sobre el coste inicial, los diseños basados en impresión 3D tienen la ventaja de su reducido coste y complejidad en el mantenimiento. Por ejemplo, los robots empleados para prácticas docentes sufren una carga y fatiga muy elevada, repitiendo los mismos patrones continuamente, sufriendo de *errores* de programación, controladores mal diseñados, etc., lo que puede resultar en partes dañadas o golpeadas. Este tipo de robots permiten imprimir y cambiar cualquier componente en cuestión de minutos sin tener que interrumpir la realización de la práctica.

Además, el uso de robots manipuladores permiten gran versatilidad a la hora de diseñar prácticas y plantear experimentos. Por ejemplo: a) control PID sencillo del motor de una junta mientras el movimiento del resto del brazo realiza movimientos como perturbaciones controladas, b) *model predictive control* para realizar control de fuerza en el efector final, c) modelado cinemático y dinámico, d) planificación y evitación de obstáculos, e) manipulación avanzada y agarres y f) aprendizaje automático y por refuerzo. En concreto en este trabajo analizamos el robot TriFinger Wuthrich et al. (2021) un manipulador de 3 dedos articulados que permite gran flexibilidad de movimientos. El robot TriFinger está completamente abierto, tanto el diseño hardware, como el software [1].

En este trabajo nos centramos en el estudio de la plataforma para la docencia de aprendizaje automático, y más concretamente del aprendizaje por refuerzo. El aprendizaje por refuerzo es una forma de enseñar a un agente de software a tomar acciones óptimas en un entorno dado, basándose en la retroalimentación de recompensas o castigos. El estudio de aprendizaje por refuerzo puede ser útil para los estudiantes de automática, ya que les permite diseñar y optimizar sistemas complejos que se adaptan a las condiciones cambiantes del entorno. Algunas aplicaciones posibles son el control de robots (Garcia-Barcos and Martinez-Cantin, 2021), vehículos autónomos, finanzas, salud, etc. Son ampliamente conocidos los resultados recientes de AlphaGo y sus variantes (Silver et al., 2016) e incluso ha demostrado ser fundamental para el entrenamiento de grandes modelos de lenguaje (Ziegler et al., 2019). Además, el aprendizaje por refuerzo puede ayudar a los estudiantes de automática a desarrollar habilidades como la formulación de problemas, la modelización matemática, la programación, la simulación, el análisis de datos y la evaluación de resultados. Estas habilidades son fundamentales para el campo de la ingeniería de control y el aprendizaje automático.

2. Robot Trifinger

La plataforma TriFinger (Wuthrich et al., 2021) consiste en un robot con 3 dedos de 3 articulaciones similares a las extremidades de un cuadrúpedo Figura 1. Cada dedo dispone de la electrónica de control en la parte superior, permitiendo su trabajo de manera coordinada o con cada uno de los dedos por separado, dependiendo de la complejidad de la tarea a realizar o el tipo de problema planteado. Los dedos se colocan en un soporte de aluminio donde también se pueden instalar hasta 3 cámaras para poder visualizar los objetos a manipular.

La versatilidad de la plataforma viene apoyada por el uso que se ha hecho durante 3 años como plataforma del *Real Robot Challenge*[2] de la prestigiosa conferencia NeurIPS. Se planteó a los participantes que diseñaran algoritmos para resolver tareas de manipulación robótica únicamente a partir de ejemplos, conocido como aprendizaje por refuerzo offline; sin disponer de acceso al robot en el que se realizarían las pruebas finales.

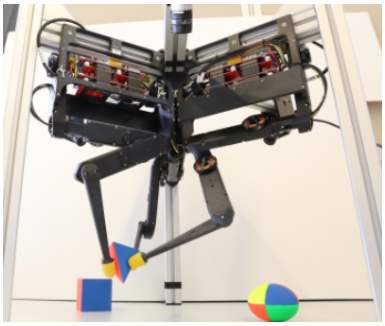


Figura 1: Plataforma Tri-Finger Wuthrich et al. (2021) realizando tareas de manipulación. En la figura se observa como cada dedo permite su instalación y control de manera independiente. En la parte superior se aprecia la cámara utilizada para percibir los objetos a manipular.

2.1. *Ventajas e inconvenientes del robot Trifinger para docencia*

El robot se ha evaluado en varias demos disponibles que comprenden desde tareas típicas de robótica de manipulación, control óptimo (fuerza) y aprendizaje por refuerzo. Al incorporar 3 cámaras, también se pueden hacer cosas interesantes de visión por computador.

El robot presenta una serie de ventajas interesantes de cara a la docencia: 1) **Precio:** Se estima un coste inicial de 5000 dolares. 2) **Fácil mantenimiento:** al ser componentes *off-the-shelf* y piezas impresas, sería fácil y rápido tener repuesto. 3) **Interfaz sencillo:** ya está pensado para docencia e investigación. Tiene un interfaz muy simple en C++ y Python (no necesita ROS[3], pero se dispone de nodos ROS para usarlo si se quiere combinar con otros nodos). 4) **Seguro:** al estar pensado para docencia, tiene implementadas bastantes restricciones de seguridad para no dañar el robot. Y en el peor de los casos, es ligero y de plástico. 5) **Simulador:** incluye un simulador en Pybullet bastante fácil de usar y con el mismo interfaz que el robot. 6) **Abierto y libre:** todo el diseño es open source/open hardware y tiene una comunidad detrás. Como se ha comentado anteriormente, ya lleva 3 años usándose como challenge en la conferencia NeurIPS. 7) **Diseño flexible y modular:** Tiene una cinemática muy sencilla por dedo, pero al trabajar con los 3 dedos en paralelo puede dar problemas interesantes de robótica.

[1] Web TriFinger: `https://sites.google.com/view/trifinger/home-page`
[2] `https://real-robot-challenge.com/en`
[3] Robot Operating System

Sin embargo, presenta algunos inconvenientes que hay que tener en cuenta: 1) **Materiales:** es plástico impreso, lo que reduce bastante la durabilidad a largo plazo, y dificulta el control-fuerza. 2) **Montaje:** Hay que comprar las piezas por separado y montarlo. A fecha de hoy no existe fabricante o comercializador que permita adquirirlo montado.

3. Simulador

Por otro lado, una alternativa todavía más asequible es el uso de simuladores. Estos son fundamentales tanto para evaluar las capacidades del robot antes de comprometerse a su compra como para facilitar el diseño de nuevos experimentos en proyectos de investigación. En el caso de la plataforma TriFinger, también esta disponible un simulador (Joshi et al., 2020) de código abierto[4]. Se trata de un entorno con físicas basado en PyBullet (Coumans and Bai, 2016), de modo que el comportamiento al interactuar con los objetos es el esperado en un entorno real. Se dispone de tres versiones distintas de la plataforma: la empleada en el concurso (TriFingerPro), la educativa (TriFingerEdu) y el prototipo inicial (TriFingerOne). Todas cuentan con tres articulaciones por dedo, reduciéndose las diferencias entre ellas a los límites de operación. Además, se cuenta con la opción de emplear los tres dedos o únicamente uno, aportando mayor riqueza y variación de los experimentos. Sin embargo, algunas de las funcionalidades disponibles están enfocadas principalmente en los robots con 3 dedos, teniendo especial relevancia la versión *TriFingerPro*. De modo que en ocasiones el usuario puede necesitar completar o extender el soporte que se da a la versión de un solo dedo articulado. Aunque no se ha llegado a emplear, también permite la integración de cámaras y con ello trabajar con sistemas basados en visión.

En cuanto a las posibilidades de sensores, es posible recibir retro-alimentación de las coordenadas articulares del robot, así como de sus velocidades. Al tratarse de un entorno simulado, se pueden conocer además las posiciones de los posibles objetos y objetivos que existan en el escenario, así como las posición de los extremos de los dedos y su distancia a los objetivos. El control de los dedos se realiza mediante una interfaz que proporciona el simulador y que ofrece un control en posición (posiciones articulares), uno en par o uno que incluye ambos.

4. Algoritmos de aprendizaje por refuerzo

Para resolver tareas de manipulación robótica es necesario aplicar técnicas de control y planificación de movimientos. No obstante, para poder resolver tareas arbitrarias, es necesario dotar al robot de capacidad de aprendizaje, puesto que es inviable la planificar todas las posibles situaciones y, en algunos casos, las tareas no son triviales. El aprendizaje por refuerzo es un conjunto de técnicas que permiten a un agente (el robot) aprender las acciones que permiten resolver una tarea únicamente mediante interacciones con el entorno (Sutton and Barto, 2018).

Formalmente, podemos modelar el problema como un Proceso de Decisiones de Markov (Markov Decision Process o MDP) el cual está definido por un espacio de estados $\mathbf{s} \in \mathcal{S}$, un espacio de acciones $\mathbf{a} \in \mathcal{A}$, una función de transición $\mathcal{P}(s_{t+1}|s_t, a_t)$ y una función de recompensa $r(\mathbf{s}, \mathbf{a})$. El objetivo es aprender una política $\pi_\theta(\mathbf{a}|\mathbf{s})$ con parámetros θ que permita escoger la acción más adecuada para cada estado. Para aprender la política, se puede maximizar la siguiente función de coste:

$$J(\theta) = E_{\tau\sim p_\theta(\tau)}\left[\sum_t \gamma^t r(\mathbf{s}_t, \mathbf{a}_t)\right], \tag{1}$$

donde τ es una trayectoria (o secuencia de estados y acciones), $p_\theta(\tau) = p(\mathbf{s}_1)\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ es la probabilidad de dicha trayectoria y γ es un factor de descuento para evitar la procrastinación del agente.

De este modo, el aprendizaje por refuerzo se centra principalmente en encontrar la solución al problema de optimización

$$\pi_\theta^* = \underset{\pi_\theta}{argmax}\ J(\pi_\theta). \tag{2}$$

En relación a la obtención del retorno esperado, típicamente se nombran dos funciones: la función de valor $V^\pi(\mathbf{s})$ y la función de acción-valor $Q^\pi(\mathbf{s}, \mathbf{a})$. La primera estima el retorno esperado en el estado $\mathbf{s}$ y actúa siguiendo la política π; mientras que en la segunda se toma primero la acción $\mathbf{a}$, escogida o no por la política π, y posteriormente se sigue la política.

$$V^\pi(\mathbf{s}) = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}\gamma^t r(\mathbf{s}_t, \mathbf{a}_t)\middle|\mathbf{s}_0 = \mathbf{s}\right] \tag{3}$$

$$Q^\pi(\mathbf{s}, \mathbf{a}) = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}\gamma^t r(\mathbf{s}_t, \mathbf{a}_t)\middle|\mathbf{s}_0 = \mathbf{s}, \mathbf{a}_0 = \mathbf{a}\right] \tag{4}$$

4.1. Búsqueda de política y actor-critic

Una forma de aproximar la la función $J(\theta)$ es mediante el muestreo,

$$J(\theta) \simeq \sum_i^N \sum_t^T \gamma^t r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}), \tag{5}$$

donde N es el número de muestras empleadas. Del mismo modo se puede estimar su gradiente

$$\nabla_\theta J(\theta) \simeq \sum_i^N \left(\sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \cdot \sum_{t'=t}^T \gamma^{t'} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'})\right). \tag{6}$$

Como se desea maximizar $J(\theta)$, y el gradiente $\nabla_\theta J(\theta)$ indica la dirección de máximo crecimiento con respecto a los parámetros θ se puede plantear un ascenso del gradiente:

Algorithm 1 REINFORCE algorithm for policy search

1: Muestrear trayectorias $\{\tau^i\}$
2: Aproximar el gradiente $\nabla_\theta J(\theta)$ según Ecuación 6
3: Actualizar los parámetros de la política $\theta \leftarrow \alpha\nabla_\theta J(\theta)$
4: Repetir pasos 1-3 hasta convergencia

Este tipo de algoritmos (Williams, 1992) basados en el uso de muestras $\mathcal{D} = \{\tau^i\} = \{(\mathbf{s}^i_{1:T}, \mathbf{a}^i_{1:T-1}, r^i_{1:T})\}$ para actualizar la

[4]Repositorio Github: `https://github.com/open-dynamic-robot-initiative/trifinger_simulation`

política se consideran como algoritmos de búsqueda de política. Estas muestras pueden ser tomadas por la propia política (*On-Policy*) o no (*Off-Policy*). En el caso de *Off-Policy*, es habitual que se incluyan muestras que se han evaluado en el pasado, almacenadas un buffer de repetición, y que por tanto corresponden a una política antigua distinta a la política actual.

Dentro de esta familia se encuentran los métodos conocidos como *Actor-Critic* (Sutton et al., 1999), en los que se entrena una red neuronal con parámetros ϕ para obtener una estimación de la función de valor $\hat{V}^{\pi}_{\phi}(\mathbf{s})$. De este modo, se tiene por un lado un *Actor*, encargado de escoger la acción óptima en cada caso, es decir, aprende la política π_θ. Por otro lado, el *Critic* es la función de valor $\hat{V}^{\pi}_{\phi}(\mathbf{s})$ que se utiliza para estimar lo bien o mal que lo está haciendo el *Actor* y así guiarlo en la dirección correcta.

Para actualizar los parámetros ϕ de la red, se puede utilizar el error de *Temporal Differences* δ_t:

$$\delta_t = r(\mathbf{s}_{t+1}, \mathbf{a}_t) + \gamma \hat{V}^{\pi}_{\phi}(\mathbf{s_t}) - \hat{V}^{\pi}_{\phi}(\mathbf{s_{t+1}}) \tag{7}$$

$$\nabla_\theta J(\theta) \simeq \sum_{i}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \cdot \delta_t \right) \tag{8}$$

Destacar el método *Proximal Policy Optimization* (PPO) (Schulman et al., 2017) el cual está basado en *Actor-Critic* pero mejora la estabilidad del entrenamiento de la política impidiendo que haya grandes cambios al actualizar la política.

4.2. *Optimalidad soft*

Aunque el aprendizaje por refuerzo se centra típicamente en la optimalidad de una política determinista, existen algoritmos que hacen uso de políticas estocásticas y que plantean una serie de ventajas durante el entrenamiento. Principalmente, permiten mejorar la exploración de manera natural, puesto que mantiene una distribución de probabilidad sobre las acciones y la acción que se toma es más aleatoria. A su vez, esto añade una convergencia más robusta, puesto que no se descartan completamente acciones alternativas, como si ocurriría con la política determinista y que podría provocar convergencias prematuras a políticas subóptimas.

Entre estos métodos destaca el *soft actor-critic* (SAC, Haarnoja et al. (2018)), una variante estocástica basada en el *actor-critic*. En SAC el actor trata de maximizar la recompensa esperada mientras que también se maximiza la entropía, es decir, el objetivo es encontrar una política estocástica que sea capaz de resolver la tarea mientras actúa lo más aleatorio posible. Esto lo consigue añadiendo a la Ecuación 4 un termino de entropía H:

$$J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\sum_t \gamma^t \left(r(\mathbf{s}_t, \mathbf{a}_t) + \alpha H(p_\theta(s)) \right) \right], \tag{9}$$

Donde $\alpha > 0$ es un coeficiente de compensación y la entropía H se calcula con respecto a la distribución de probabilidad resultante de la política estocástica $p_\theta(s)$.

5. Tareas

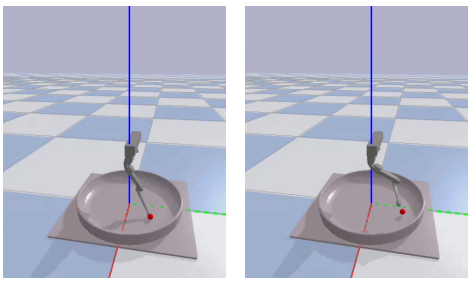
El simulador ofrece dos tareas por defecto (*Reach* y *Push*) que son compatibles con la plataforma educativa, aunque es posible definir nuevas y modificar las ya existentes. La primera consiste en alcanzar puntos en el espacio 3D con el extremo del dedo, por tanto el modelo entrenado debe ser capaz de aprender el modelo cinemático inverso del robot y de planificar los cambios en las coordenadas articulares del robot para evitar velocidades excesivas. La segunda tarea es más compleja, ya que implica mover un cubo desde una posición inicial conocida a una posición final también conocida.

5.1. *Tarea 1: Reach*

En esta tarea, el agente dispone de n_{steps} pasos para llevar el extremo del dedo de la posición actual, $\vec{x}_t^{\,end\ effector}$, a la posición deseada $\vec{x}^{\,goal}$. La recompensa r_t asociada a cada instante de tiempo t en esta tarea es la distancia euclidea d_t entre el extremo del robot ($\vec{x}_t^{\,end\ effector}$) y la localización 3D que debe alcanzar ($\vec{x}^{\,goal}$), con signo negativo.

$$r_t = -\|\vec{x}^{\,goal} - \vec{x}_t^{\,end\ effector}\| = -d_t. \tag{10}$$

Cada episodio acaba si se supera ese límite temporal o si por el contrario se completa la tarea, $d_t < d_{threshold}$. De esta forma se favorecen dos comportamientos por parte del robot. En primer lugar, acercarse a la posición objetivo, ya que incrementa la recompensa (en valor absoluto). Por otro lado, dado que la recompensa de un paso nunca es positiva, cuanto antes se complete la tarea con éxito, mayor debería ser la recompensa final obtenida. En la 5.1 puede observarse el robot intentando alcanzar distintos objetivos, que son representados mediante una esfera roja.



(a) (b)

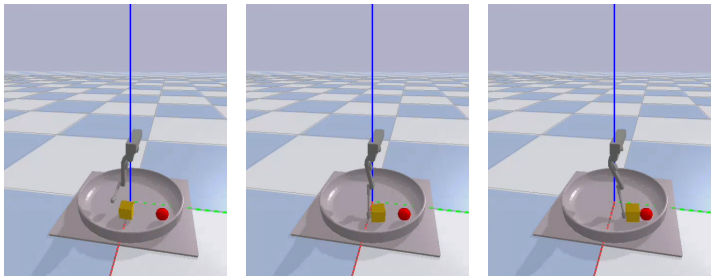
Figura 2: Ejemplos de configuraciones articulares del robot durante la simulación de la tarea *Reach*, donde el robot debe tocar con su extremo el objetivo (esfera roja).

5.2. *Tarea 2: Push*

Al igual que en la anterior tarea, se disponen de n_{steps} pasos para completarla y la recompensa en cada instante es también una distancia, pero en este caso la que hay de la posición actual del cubo $\vec{x}_t^{\,cubo}$ a la objetivo $\vec{x}^{\,goal}$,

$$r_t = -\|\vec{x}^{\,goal} - \vec{x}_t^{\,cubo}\| = -d_t. \tag{11}$$

Siguiendo el mismo enfoque que para la tarea *Reach*, el agente logra mayor recompensa si logra cumplir la tarea antes de tiempo, ya que suma menos recompensas negativas. Sin embargo, hay otra forma de finalizar esta tarea, consiste en mover el cubo fuera del alcance del robot. En este caso, se penaliza con una recompensa muy negativa para impedir que durante el entrenamiento se vea reforzado este comportamiento, ya que una finalización prematura evita sumar recompensas negativas.



(a) Situación inicial. (b) Contacto con el cubo. (c) Empujando el cubo.

Figura 3: Ejemplos de configuraciones adoptadas por el robot y evolución del entorno durante la simulación de la tarea *Push*. El cubo amarillo debe ser movido desde su posición inicial hasta el objetivo, representado por la esfera roja

En la 3(a) puede observarse la situación inicial y en 3(b), 3(c) pasos intermedios ejecutados durante esta tarea. La posición objetivo del cubo es la esfera roja.

6. Experimentos

Se han realizado pruebas en el simulador con la plataforma educativa en las dos tareas disponibles para ella por defecto en el entorno. Inicialmente, se ha comprobado el funcionamiento del simulador y la posibilidad de emplear los algoritmos de aprendizaje por refuerzo libres de modelo implementados en la librería *Stable Baselines 3* (Raffin et al., 2021).En concreto se han probado los algoritmos *PPO* (Schulman et al., 2017) y *SAC* (Haarnoja et al., 2018). Como el objetivo era verificar el funcionamiento la plataforma y su compatibilidad con librerías externas y listas para usar se ha optado por simplificar los problemas a resolver, ejecutando en escenarios con un único dedo.

Para las dos tareas se han entrenado 5 instancias de cada uno de los modelos planteados para poder analizar la validez de las propuestas y paliar posibles irregularidades debidas a las aleatoriedades presentes en el proceso. En cada entrenamiento se han ejecutado 250000 pasos, con una duración máxima de 300 pasos para cada episodio.

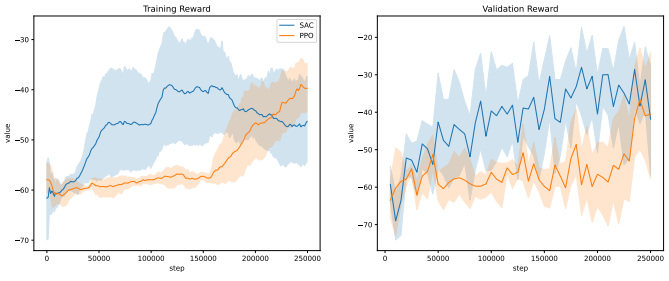


Figura 4: Comparación de recompensas medias obtenidas a lo largo del entrenamiento y en los episodios de validación intermedios en la tarea *Reach*.

6.1. Tarea 1: Reach

En esta tarea el robot parte siempre de la misma configuración articular inicial, mientras que el objetivo se escoge de forma aleatoria dentro de su rango de operación. En este escenario se ha entrenado un modelo de SAC empleando los hiperparámetros planteados en la Tabla 1. La *tasa de aprendizaje* es el tamaño del paso tomado en la dirección del gradiente, el *tamaño de Batch* el número de muestras empleadas en cada paso de entrenamiento, los *pasos de gradiente* el número de pasos tomados en cada *rollout* y está relacionado con temas de eficiencia. Las acciones se muestrean del *buffer de repetición* y la *frecuencia de entrenamiento* indica el número de pasos de simulación transcurridos entre dos pasos de entrenamiento. Posteriormente se han repetido los entrenamientos con PPO, con los hiperparámetros que ofrece *Stable Baselines 3* por defecto para este algoritmo, a excepción de los que se mencionan en la Tabla 1. Las recompensas obtenidas durante ambos entrenamientos se pueden ver en Figura 4. Puede observarse que los modelos basados en SAC parecen estar aprendiendo la tarea con mayor facilidad al inicio del entrenamiento, ya que alcanzan una mayor recompensa rápidamente. Sin embargo, durante los últimos 100000 episodios, la recompensa recibida por los modelos PPO experimenta un fuerte crecimiento, obteniendo un valor más elevado al finalizar el entrenamiento.

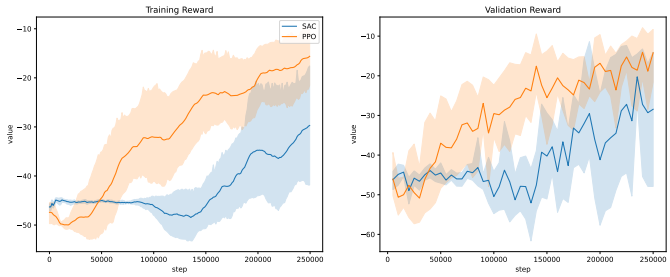


Figura 5: Comparación de recompensas medias obtenidas a lo largo del entrenamiento y en los episodios de validación intermedios en la tarea *Push*.

6.2. Tarea 2: Push

Inicialmente se ha planteado un escenario en el que tanto la posición inicial como la objetivo son fijas. Bajo esas circunstancias se ha entrenado un modelo de *SAC* con los hiperparámetros de la Tabla 1. Durante el entrenamiento se han ejecutado 5 episodios de evaluación cada 5000 pasos en el entorno de aprendizaje. En cuanto al entrenamiento de los modelos PPO, se han vuelto a emplear los parámetros de la tarea anterior, Tabla 1. La evolución de las recompensas de ambos modelos se puede comparar en Figura 5. En esta segunda tarea PPO obtiene mejores resultados que SAC durante casi todo el entrenamiento.

Hiperparámetro	Reach (SAC)	Push (SAC)	Reach — Push (PPO)
Tasa de aprendizaje	0,005	0,0005	0,005
Factor de Descuento (γ)	0,95	0,95	0,95
Tamaño de Batch	1024	1024	1024
Pasos de gradiente	10	10	-
Tamaño del Buffer *de Repetición*	$3 \cdot 10^6$	$3 \cdot 10^6$	-
Frecuencia de Entrenamiento	64 pasos	64 pasos	-
Arquitectura Red	Perceptrón multicapa	Perceptrón multicapa	Perceptrón multicapa
Capas Red	[256, 256, 256, 256]	[256, 256, 256, 256]	[256, 256, 256, 256]

Tabla 1: Hiperparámetros empleados en los entrenamiento de los diferentes modelos y tareas.

7. Lecciones Aprendidas

La pruebas realizadas con el simulador han permitido experimentar con las posibilidades que ofrece no solo en el plano de la investigación, sino también en la faceta docente. Se trata de un entorno con un motor de físicas realista. Además, es compatible con la interfaz empleada por la librería *Gym* (Brockman et al., 2016), facilitando su integración con librerías ya existentes de aprendizaje reforzado o incluso la implementación de algoritmos presentes en la literatura durante la realización de prácticas de laboratorio. Las tareas ofrecidas por defecto son lo suficientemente sencillas para ser resueltas por algoritmos basados en modelo en alrededor de 250000 pasos de simulación, teniendo en cuenta que los episodios se han limitado a 300 pasos, se han ejecutado aproximadamente 800 episodios para cada entrenamiento. En términos de tiempo real, esto supone 20 minutos en el caso de PPO y $1,5$ horas en el caso de SAC. Otro aspecto positivo es la flexibilidad que caracteriza a este entorno, siendo posible la implementación de nuevas tareas propias o la modificación de las ya existentes.

Si bien tanto el propio entorno como las librerías empleadas ofrecen las ventajas mencionadas anteriormente, también presentan ciertas complicaciones. En primer lugar, la librería *Gym* ha sufrido importantes cambios con el tiempo, llegando al punto de ser incompatible la interfaz empleada en las más recientes con la empleada en versiones anteriores. Esto supone un problema al tratar de trabajar con múltiples librerías que empleen distintas versiones de la misma. Aunque se ha podido solventar mediante el uso de *wrappers* intermedios y la modificación del código implicado, esto dificulta su puesta a punto en las aulas ya que puede suponer dedicar una gran cantidad de tiempo por parte del alumnado a tareas no relacionadas con el temario. Sin embargo, este problema sería común a la mayoría de entornos y librerías basados en *Gym*, que suele ser la gran mayoría de los empleados en docencia. Otra complicación encontrada de naturaleza similar a la anterior es la existencia de funciones en la interfaz ofrecida por la librería *trifinger_simulation* que no cumplen con las especificaciones y restricciones impuestas aunque en la documentación del código aseguren hacerlo. Si bien puede corregirse la programación de dichas funciones, entorpece su puesta a punto y complica la detección de errores propios, ya que el propio simulador se convierte en una fuente de ellos.

8. Conclusiones

El uso de robots abiertos supone una herramienta interesante para la docencia de automatización y aprendizaje automático. Este tipo de plataformas resultan muy atractivas para los estudiantes y permiten trabajar a varios niveles de complejidad fácilmente. En este caso, el robot Trifinger permite disponer de un sistema completo de bajo coste y fácil mantenimiento al estar construido con piezas de impresión 3D, componentes estándar y software libre. En este artículo hemos analizado con detalle el uso del simulador disponible para la docencia de aprendizaje por refuerzo, mostrando como ofrece un abanico de posibilidades que permiten desde ensayos sencillos a experimentos que ponen a prueba los métodos del estado del arte. Sin embargo, la plataforma también tiene sus limitaciones tanto a nivel de hardware como el software al carecer del grado de madurez de otras plataformas comerciales, pero con un coste muy elevado.

Agradecimientos

Este trabajo ha sido financiado en parte por los proyectos: PID2021-125209OB-I00 y TED2021-131150B-I00.

Referencias

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W., 2016. Openai gym. arXiv preprint arXiv:1606.01540.

Coumans, E., Bai, Y., 2016. Pybullet, a python module for physics simulation for games, robotics and machine learning.

Garcia-Barcos, J., Martinez-Cantin, R., 2021. Robust policy search for robot navigation. IEEE Robotics and Automation Letters 6 (2), 2389–2396.

Haarnoja, T., Zhou, A., Abbeel, P., Levine, S., 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.

Joshi, S., Widmaier, F., Agrawal, V., Wüthrich, M., 2020.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., Dormann, N., 2021. Stable-baselines3: Reliable reinforcement learning implementations. Journal of Machine Learning Research 22 (268), 1–8.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017. Proximal policy optimization algorithms. CoRR abs/1707.06347.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al., 2016. Mastering the game of go with deep neural networks and tree search. nature 529 (7587), 484–489.

Sutton, R. S., Barto, A. G., 2018. Reinforcement learning: An introduction. MIT press.

Sutton, R. S., McAllester, D., Singh, S., Mansour, Y., 1999. Policy gradient methods for reinforcement learning with function approximation. Advances in neural information processing systems 12.

Williams, R. J., 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Reinforcement learning, 5–32.

Wuthrich, M., Widmaier, F., Grimminger, F., Joshi, S., Agrawal, V., Hammoud, B., Khadiv, M., Bogdanovic, M., Berenz, V., Viereck, J., Naveau, M., Righetti, L., Schölkopf, B., Bauer, S., 16–18 Nov 2021. Trifinger: An open-source robot for learning dexterity. In: Kober, J., Ramos, F., Tomlin, C. (Eds.), Proceedings of the 2020 Conference on Robot Learning. Vol. 155 of Proceedings of Machine Learning Research. PMLR, pp. 1871–1882.

Ziegler, D. M., Stiennon, N., Wu, J., Brown, T. B., Radford, A., Amodei, D., Christiano, P., Irving, G., 2019. Fine-tuning language models from human preferences. arXiv preprint arXiv:1909.08593.