CEA Comité Español de Automática
XLVII JORNADAS DE AUTOMÁTICA CÓRDOBA 2026
UNIVERSIDADE DA CORUÑA

Jornadas de Automática

Una aproximación preliminar al despliegue asíncrono de software de aprendizaje por refuerzo en sistemas físicos

Adrián Bañuls-Arias[a,*], Diego Caruana-Montes[a], Ana Cruz-Martín[a], Vicente Arévalo-Espejo[a], Cipriano Galindo[a], Juan-Manuel Gandarias[a]

[a] *UnCORE Team, Instituto Universitario de Investigación en Ingeniería Mecatrónica y Sistemas Ciberfísicos (IMECH.UMA). Dpto. de Ingeniería de Sistemas y Automática, Universidad de Málaga, Arquitecto Francisco Peñalosa, nº 6, 29071, Málaga, España.*

To cite this article: Bañuls-Arias, A., Caruana-Montes, D., Cruz-Martín, A., Arévalo-Espejo, V., Galindo, C., Gandarias, J.M. 2026. A preliminary approach for the asynchronous deployment of reinforcement learning software on physical systems. Jornadas de Automática, 47. https://doi.org/10.17979/ja-cea.2026.47.13751

Resumen

El aprendizaje por refuerzo (RL) suele asumir tiempos de paso fijos en el bucle de control, ignorando las discrepancias temporales entre simulación y realidad, así como el efecto de las latencias existentes en dicho software. Esto puede degradar el rendimiento al transferir la simulación a la realidad (*sim-to-real gap*). Este trabajo presenta una solución desacoplada que permite una ejecución de RL asíncrona, agnóstica al algoritmo y con tratamiento explícito del tiempo. Al aislar el algoritmo del agente en bucles independientes, se superan las limitaciones de los marcos de desarrollo síncronos tradicionales. Esto facilita la integración con robots físicos y simuladores, permitiendo implantar estrategias de RL con tiempo de paso variable. Nuestra herramienta es compatible con bibliotecas como *Stable-Baselines3*, simuladores como *MuJoCo* y robots como el Panda de Franka Emika. Una validación experimental preliminar ha demostrado su efectividad con dicho robot manipulador, y la versión inicial del software ha sido puesta a disposición de la comunidad en GitHub: `https://github.com/uncore-team/rl_spin_decoupler`.

Palabras clave: Manipuladores robóticos, Aprendizaje por refuerzo, Tiempo de paso variable, Despliegue asíncrono, Arquitectura software distribuida.

A preliminary approach for the asynchronous deployment of reinforcement learning software on physical systems.

Abstract

Reinforcement Learning (RL) usually assumes fixed timesteps, ignoring temporal discrepancies between simulation and reality and the effects of system latencies. These factors can degrade performance in sim-to-real transfer. This work introduces a fully decoupled solution enabling asynchronous, algorithm-agnostic, and time-aware RL deployment. Our approach isolates the algorithm from the agent into independent loops, overcoming the limitations of traditional synchronous frameworks. This facilitates integration with physical robots and closed-source simulators, enabling variable-time RL strategies. Our tool is compatible with libraries like Stable-Baselines3, simulators such as MuJoCo, and robots like the Franka Emika Panda. A preliminary experimental validation has demonstrated its effectiveness with such a robotic manipulator, and an initial version of the software has been made available to the community on GitHub: `https://github.com/uncore-team/rl_spin_decoupler`.

Keywords: Robotic manipulators, Reinforcement Learning, Asynchronous Deployment, Variable timestep, Distributed Software Architecture.

1. Introducción

El aprendizaje por refuerzo (RL), y específicamente el RL profundo (DRL), ha surgido como paradigma fundamental para la toma de decisiones autónoma en sistemas robóticos Le et al. (2024); Al Mahmud et al. (2024); Prasuna and Potturu (2024). El RL ofrece una alternativa robusta a los métodos tradicionales de control al permitir que los agentes aprendan

*Autor para correspondencia: `adrianb_arias@uma.es`

comportamientos óptimos mediante la interacción con el entorno. Durante la última década, la integración del RL en la robótica ha impulsado avances significativos en tareas como la manipulación Elguea-Aguinaco et al. (2023) o la navegación autónoma Zhu and Zhang (2021), entre muchas otras.

Sin embargo, la mayor parte de la investigación existente sigue limitada a entornos simulados, con una transferencia reducida a robots reales. Esta brecha se debe, en parte, al uso generalizado de arquitecturas software síncronas que heredan la estructura secuencial del formalismo subyacente: los Procesos de Decisión de Markov (MDP) Puterman (2014). En dicho enfoque, el código del algoritmo de RL y el del agente físico se modelan entrelazados en un único bucle software que corre en un hilo de ejecución también único; además, los MDP ignoran los retrasos existentes, a menudo estocásticos, que aparecen tanto en el RL como en el agente, y en las interacciones entre ambos. Descartar la dinámica temporal provoca que las implementaciones literales sobre MDPs resulten poco representativas de las condiciones del mundo real, induciendo variabilidad en los resultados del aprendizaje y en su ejecución en distintas máquinas, así como agravando problemas de inestabilidad e ineficiencia de muestreo en el bucle de control Dulac-Arnold et al. (2019); Ibarz et al. (2021).

Existen varios *frameworks* de desarrollo para RL ampliamente utilizados por la comunidad, como *Gymnasium* Towers et al. (2024) y *Stable-Baselines3* Raffin et al. (2021). Aunque estos *frameworks* proporcionan interfaces estandarizadas e implementaciones fiables de los algoritmos, lo que facilita la reproducibilidad, dependen de tiempos de paso fijos y ejecución síncrona, lo que los hace susceptibles a los problemas mencionados. Hay por tanto una demanda de una arquitectura de software específica que permita el despliegue de RL asíncrono con un procedimiento de desacoplamiento entre algoritmo de aprendizaje y agente que facilite el análisis de los efectos temporales en el despliegue del sistema (ver Fig. 1).

Hasta la fecha, existen pocas aproximaciones de este tipo. En Yuan and Mahmood (2022) se presenta una arquitectura asíncrona que considera las interacciones RL-agente, pero requiere modificaciones en el algoritmo de RL e implica una carga computacional relevante al realizar inferencias repetidas en el lado del agente, algo poco práctico en robots de pequeño tamaño o con poca capacidad de cómputo a bordo. Otro trabajo relevante, Mahmood et al. (2018), adopta una arquitectura desacoplada para examinar el impacto de las latencias software, pero su análisis de las duraciones de acción sigue siendo superficial al centrarse exclusivamente en las recompensas resultantes sin detallar los procesos temporales involucrados.

Aquí proponemos una solución de software minimalista y agnóstica al algoritmo: *RL-Spin-Decoupler*, que logra un despliegue de RL totalmente desacoplado y asíncrono. La solución implementa patrones de software *Adapter* Gamma et al. (1994) que centralizan las comunicaciones entre los bucles de RL y del agente. A diferencia de trabajos previos, este enfoque no altera el algoritmo de aprendizaje, el espacio de acción o la estructura de la política, haciéndolo compatible con herramientas como las ya mencionadas (p.ej., *Stable-Baselines3* Raffin et al. (2021)), simuladores abiertos como *MuJoCo* Todorov et al. (2012) y plataformas de hardware cerradas como *Franka Emika* Haddadin (2024). Al aislar completamente la ejecución del algoritmo de RL del bucle del agente, nuestra herramienta permite manejar tiempos de paso variables y garantiza la monitorización de las latencias. Hasta donde sabemos, esta es la primera solución de código abierto que permite un desacoplamiento arquitectónico orientado al RL en robótica con tratamiento explícito del tiempo.

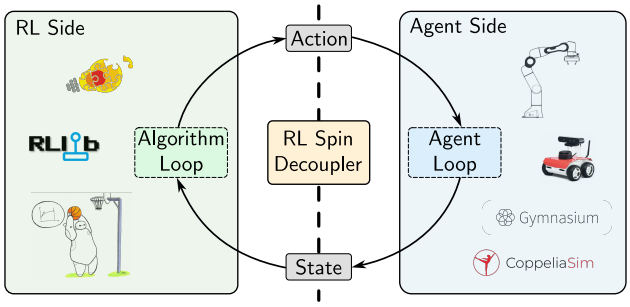

Figura 1: *RL-Spin-Decoupler* permite un despliegue de RL totalmente desacoplado y asíncrono tanto en robots simulados como físicos.

El resto del artículo se organiza así: la sección 2 presenta el planteamiento del problema; la sección 3 describe el diseño del *RL-Spin-Decoupler*; la sección 4 detalla un caso de uso con un manipulador Franka Emika simulado y en real; y, por último, la sección 5 que recoge las conclusiones del trabajo.

2. RL con tratamiento explícito del tiempo

El despliegue de RL en sistemas robóticos físicos introduce desafíos a menudo ignorados. Dado que la mayoría de las implementaciones actuales asumen tiempos de paso fijos y ejecución síncrona tanto del algoritmo de aprendizaje como de los procesos de observación y ejecución de acciones en el agente, se omiten las discrepancias temporales entre simulación y realidad (*sim-to-real gap*). Además, integrar el algoritmo de RL y el bucle de control del agente en un mismo hilo de CPU puede ser inviable en plataformas robóticas o simuladores físicamente realistas como *CoppeliaSim* Robotics (2024), que proporcionan códigos cerrados.

Incluso cuando la fusión síncrona basada en una implementación literal de un MDP es factible, las latencias de comunicación, sensorimotoras y de cómputo asociado al propio RL provocan que el tiempo efectivo durante el que se ha ejecutado la última acción (denominado aquí LAT, *last action time*) difiera del valor nominal establecido por el propio algoritmo de aprendizaje. Esta desviación altera los efectos de las recompensas, estados y acciones previamente definidos, exacerbando la brecha *sim-to-real* al transferir las políticas a robots reales Salvato et al. (2021); Zhao et al. (2020).

El LAT puede diferir significativamente según diversas situaciones, de las que las más relevantes son:

- *Ideal*: El agente final es simulado y opera más rápido que el tiempo real. Si los cálculos de RL finalizan antes del tiempo de paso nominal, o si durante dichos cálculos la simulación está detenida, el sistema se comporta de forma muy cercana a un MDP puro, sin grandes efectos de la dinámica temporal que ocurriría en una implantación real, la cual está siendo ignorada.

- *Aproximado*: El sistema físico o simulado cumple el paso nominal excepto por retrasos estocásticos debidos a

latencias en el software. Existen ciertos límites para ejecutar procesos adicionales (como la repetición de experiencias) sin desviarse de la duración nominal.

- *Estricto o de tiempo real*: Requiere un acoplamiento síncrono en hardware con capacidades deterministas. Minimiza latencias y garantiza el cumplimiento del LAT nominal si el RL es suficientemente rápido.
- *Sobrecargado*: El tiempo de cómputo excede el nominal frecuentemente. Esto puede ocurrir en arquitecturas síncronas o agentes reales, invalidando los resultados de simulación y deteriorando severamente el rendimiento.

Esta diversidad de escenarios subraya la importancia de monitorizar los tiempos para comprender su impacto en la convergencia y la seguridad del RL. Como ventaja adicional de un despliegue asíncrono y desacoplado como el propuesto, dichos datos se pueden recopilar con mayor precisión. Los principales retardos que pueden monitorizarse son:

- **De comunicación**: En sistemas de computación distribuida a bordo o cuando el aprendizaje se ejecuta en máquinas externas potentes Mahmood et al. (2018).
- **De selección de acción**: Dependiente de la complejidad de la política de comportamiento (*behaviour policy*) y de la capacidad de cómputo y multiprocesamiento.
- **De actualización**: Tiempo necesario para actualizar el modelo. Se podría evitar este retraso modificando el software de RL para permitir una asincronía total, pero eso añadiría el coste de transferencia de modelos entre agente y RL y el de hacer inferencias locales.
- **Del sistema**: Retrasos subyacentes, como la respuesta de los actuadores, los ciclos de muestreo de los sensores o los períodos de control impuestos por el agente.

3. Ejecución asíncrona del software robótico de RL

En este artículo presentamos una versión preliminar de una herramienta desarrollada en Python Van Rossum and Drake (2009) que integra las implementaciones del agente y del RL sin modificar los algoritmos o controladores preexistentes. La comunicación entre ambos componentes se realiza mediante *sockets TCP* Postel (1981) de forma transparente, lo que garantiza un desacoplamiento fiable durante el entrenamiento y la explotación, facilitando además la conexión con robots físicos que utilicen esta tecnología.

Este diseño hace que ambos lados (agente / RL) tengan bucles de control desacoplados y ejecutando a distintas frecuencias, de manera que, asíncronamente, el segundo envía acciones al primero y el primero devuelve observaciones. Nótese que, mientras el agente no recibe una nueva acción, sigue ejecutando la anterior. Esto contrasta con otros enfoques, como Yuan and Mahmood (2022), donde el agente puede inferir nuevas acciones en ese caso. Nuestra aproximación: (i) evita alterar la estructura interna del algoritmo de aprendizaje, (ii) reduce la complejidad computacional y de memoria al no tener que compartir el modelo aprendido hacia el agente, y (iii) facilita el análisis de los retardos. El inconveniente es que las acciones ejecutadas pueden extenderse más allá de sus tiempos nominales; si los períodos de control son cortos, la política no debería verse afectada significativamente por este motivo.

La versión preliminar del *RL-Spin-Decoupler* se estructura principalmente en dos clases, `RLSide` y `AgentSide`, que son las encargadas de implementar el patrón *Adapter* y de encapsular la interacción entre el algoritmo de RL y el agente. Con este fin, la clase `RLSide` ofrece métodos para que el algoritmo de aprendizaje solicite observaciones o envíe acciones:

- `resetGetObs()`: Ordena el reinicio del agente y su entorno, esperando hasta recibir la primera observación.
- `stepSendActGetObs(action)`: Envía una acción al agente y recupera la observación resultante, el tiempo medido en el agente (simulado o real) y el tiempo real de ejecución si difiere del anterior.
- `stepExpFinished()`: Indica la finalización del proceso y señala el cierre de la comunicación al agente.

La clase `AgentSide`, por otra parte, ofrece métodos para el bucle de ejecución del agente (simulador o robot físico):

- `readWhatToDo()`: lee instrucciones del lado de RL y devuelve los siguientes comandos al agente: `RESET_SEND_OBS` - reiniciar episodio y enviar observación inicial; `REC_ACTION_SEND_OBS` - ejecutar acción y enviar la observación resultante al RL; y `FINISH` - finalizar experimento y cerrar comunicaciones.
- `stepSendLastActDur()`: reporta el tiempo de ejecución real de la última acción (LAT) según el reloj del agente y el real. Esta información es clave para el tratamiento explícito del tiempo, permitiendo cuantificar la discrepancia respecto al tiempo de paso nominal.

En caso de usar relojes distintos por ejecución distribuida o simulación asíncrona, los problemas de sincronización pueden gestionarse mediante métodos especializados (véase, por ejemplo, Fernández-Madrigal et al. (2020)).

4. Caso de uso: Manipulador Franka Emika

En esta sección se presenta un caso de estudio que emplea *RL-Spin-Decoupler* en un robot manipulador con el algoritmo de aprendizaje *Soft Actor-Critic (SAC)* Haarnoja et al. (2019).

4.1. Implementación y Configuración temporal

La Fig. 2 detalla el despliegue software en el sistema Franka Emika. En este caso se usan las bibliotecas *Gymnasium* Towers et al. (2024), *Stable-Baselines3* Raffin et al. (2021), *dm_robotics_panda* Elsner (2023) y el simulador físicamente realista *MuJoCo* Todorov et al. (2012). Nótese que *dm_robotics_panda* proporciona una API válida tanto para simular el robot como para controlarlo físicamente. Para facilitar la integración asíncrona y el tratamiento explícito del tiempo, se definen tres parámetros:

- `PHYSICS_TIMESTEP` ($0.002s$): Período de simulación del motor físicamente realista *MuJoCo*.
- `CONTROL_TIMESTEP` ($0.05s$): Período de acción del agente *dm_robotics_panda* sobre el robot simulado o físico; debe ser mayor que `PHYSICS_TIMESTEP`.

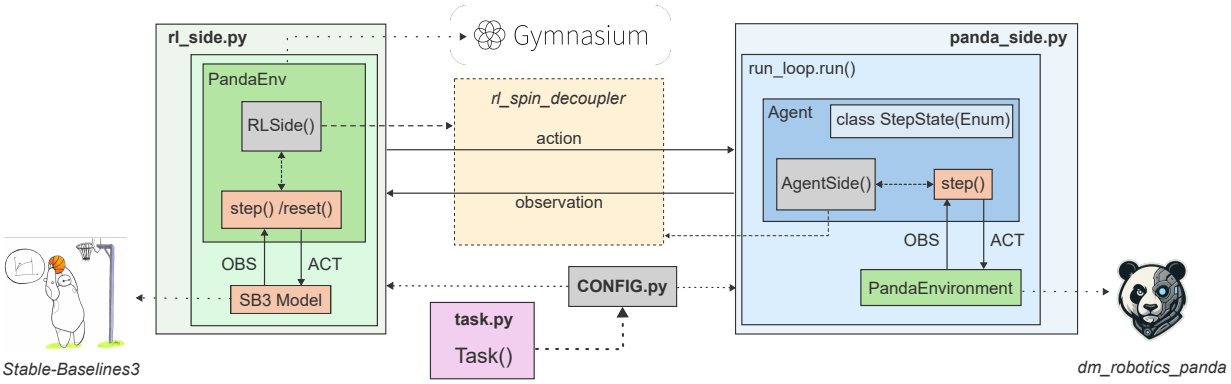

Figura 2: Arquitectura de software de RL asíncrona para el robot Franka Emika utilizando *RL-Spin-Decoupler*.

- `RL_TIMESTEP` ($0.1s$): Período de toma de decisiones del RL cuando recibe una nueva observación. Al ser el mayor, el agente puede tener que extender la última acción durante varios ciclos de control hasta recibir una nueva.

La arquitectura se organiza en cuatro archivos especializados: `task.py` define la tarea de aprendizaje; `CONFIG.py` centraliza los parámetros; `panda_side.py` gestiona el entorno del robot mediante la clase `AgentSide`; y `rl_side.py` ejecuta el algoritmo SAC empleando la clase `RLSide`. El software de esta integración esta libremente disponible en el repositorio de GitHub `https://github.com/uncore-team/franka_rl`.

4.2. *Validación experimental y Resultados*

Para la validación de la propuesta presentada, se ha diseñado un experimento en el que el manipulador debe alcanzar un objetivo lo más rápido posible. Los espacios de observación ($\boldsymbol{O}$) y acción ($\boldsymbol{\mathcal{A}}$) se definen en las ecuaciones 1 y 2, respectivamente.

$$\begin{aligned}\boldsymbol{O} = \{\, &(\dot{\mathbf{x}}_{EF}, \mathbf{x}_{EF}, d, f) \in \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R} \times \mathbb{R}, \\ &\dot{\mathbf{x}} \in [-\dot{x}_{\max}, \dot{x}_{\max}], \mathbf{x} \in [-x_{\max}, x_{\max}], \\ &d \in [0, d_{\max}], f \in [0, f_{\max}] \,\}\end{aligned} \tag{1}$$

$$\boldsymbol{\mathcal{A}} = \left\{ \dot{\mathbf{x}}^d_{EF} \in \mathbb{R}^3, \dot{\mathbf{x}}^d_{EF} \in [-\dot{x}_{\max}, \dot{x}_{\max}]^3 \right\}, \tag{2}$$

donde $(\dot{\mathbf{x}}_{EF}, \mathbf{x}_{EF})$ representan la velocidad y la posición actuales del efector final (EF), respectivamente; d denota la distancia euclídea entre la posición del EF y el punto objetivo; f es la magnitud de la fuerza medida en el EF; y $(\dot{\mathbf{x}}^d_{EF}, \mathbf{x}^d_{EF})$ constituyen la velocidad y la posición deseadas del EF, respectivamente. Asimismo, $d_{\max} = 3$ m corresponde a la distancia máxima medible al objetivo; $x_{\max} = 1.5$ m es la distancia máxima medible al objetivo a lo largo de cada eje; $\dot{x}_{\max} = 0.5$ m/s representa la velocidad máxima permitida del EF en cada eje; y $f_{\max} = 100$ N es la magnitud de fuerza máxima medible.

La función de recompensa $\mathcal{R}$ (Ec. 3) es diferenciable, está normalizada ($K = 100$), y penaliza la distancia recorrida ($k_d = 89$), la fuerza ejercida ($k_f = 5$), la velocidad del efector final ($k_v = 5$) y el tiempo empleado en alcanzar el objetivo ($k_t = 1$).

$$\mathcal{R} = -\frac{1}{K}\left(k_d \cdot \frac{d}{d_{max}} + k_f \cdot \frac{f}{f_{max}} + k_v \cdot \frac{\|\dot{\mathbf{x}}_{EF_{t-1}} - \dot{\mathbf{x}}_{EF_t}\|}{\sqrt{3} \cdot \dot{x}_{max}} + k_t\right) \tag{3}$$

El entrenamiento se llevó a cambo en simulación (50.000 episodios) utilizando la infraestructura de hardware del IMECH.UMA IMECH.UMA (2025). Posteriormente, las pruebas de explotación consistieron en desplazar el efector final –tanto en simulación como en hardware real– a cuatro posiciones dispuestas en forma de cuadrado en el plano YZ de la base del manipulador.

Los resultados mostrados en la Fig. 3 confirman la naturaleza asíncrona entre el agente y el algoritmo de RL: la posición y la velocidad deseadas, proporcionadas por `rl_side` con un *timestep* de 0.1 s, presentan una menor densidad de puntos que los valores reales, suministrados por `panda_side` con un *timestep* para la simulación de la física de 0.002 s. Asimismo, se observa que en el escenario simulado –en el cual se desarrolló el aprendizaje– el comportamiento real se ajusta con mayor precisión a las consignas deseadas, particularmente en los estados estacionarios de posición y en el perfil de velocidad.

La Fig. 4 muestra la evolución del LAT (*Last Action Time* o tiempo de la última acción) a lo largo de un experimento realizado en el entorno simulado durante la explotación de la política obtenida. La línea horizontal magenta, situada en 0.1 s, indica el *timestep* de RL nominal establecido para dicho experimento. A partir de esta gráfica, se concluye que el LAT real es, efectivamente, superior al *timestep* nominal fijado y que, además, presenta una fluctuación temporal (*jitter*) significativa.

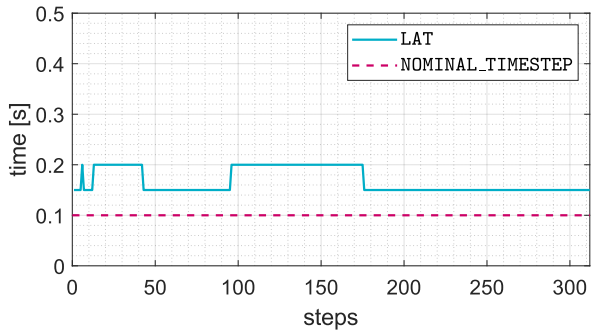

Figura 4: Evolución del LAT por paso para el modelo entrenado con *timestep* de RL nominal de 0.1 s durante la ejecución de la política aprendida en el entorno de simulación.

Las pruebas experimentales recopiladas en esta sección demuestran que *RL-Spin-Decoupler* constituye una aproxima-

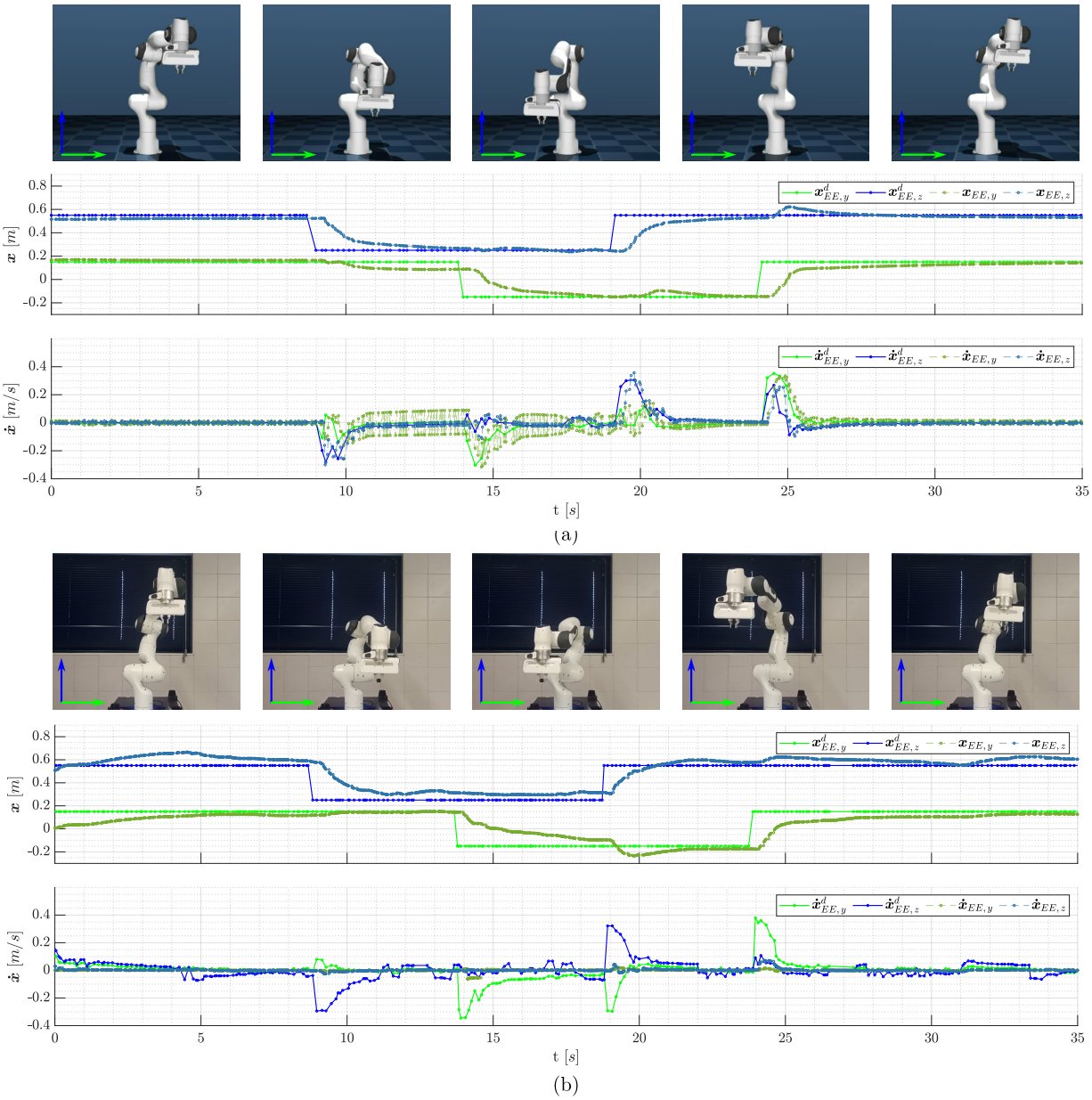

Figura 3: Explotación de la política aprendida en el caso de uso: (a) en un entorno simulado y (b) en un entorno real. Ambas figuras recogen la posición deseada (x^d_{EE}) y real (x_{EE}) del efector final (gráfico superior) y la velocidad deseada ($\dot{x}^d_{EE}$) y real ($\dot{x}_{EE}$) (gráfico inferior) de un experimento. Nótese que, dado que la tarea se realiza en el plano YZ con respecto al sistema de referencia de la base del manipulador, solo se han representado los valores para dichos ejes.

ción válida para ejecutar eficazmente tareas de RL asíncronas en sistemas tanto físicos como simulados sin modificar el software de control ni el algoritmo de aprendizaje; asimismo, dicha herramienta proporciona datos detallados para el análisis y el tratamiento explícito del factor temporal.

5. Conclusiones

Este artículo ha presentado la versión preliminar del *RL-Spin-Decoupler*, un marco de software minimalista, de código abierto y agnóstico respecto al algoritmo de RL, diseñado para permitir una ejecución asíncrona con tratamiento explícito del tiempo en sistemas robóticos. A diferencia de las arquitecturas síncronas convencionales, que asumen tiempos de paso fijos, nuestro enfoque emplea patrones software *Adapter* para lograr un diseño totalmente desacoplado. Esto permite que el algoritmo de RL y el agente operen de forma independiente sin necesidad de modificar los controladores robóticos o las implementaciones de RL preexistentes.

RL-Spin-Decoupler ofrece tres ventajas estratégicas para la investigación: (i) proporciona un mecanismo práctico para integrar RL en robots físicos y simuladores cerrados, facilitando la transferencia *sim-to-real*; (ii) permite la monitorización detallada de la dinámica temporal (como el LAT y latencias de comunicación), factores críticos para analizar la eficiencia de muestreo, la estabilidad y el rendimiento general, así como el *sim-to-real gap*; y (iii) garantiza compatibilidad con ecosistemas ampliamente adoptados como *Stable-Baselines3*, *MuJoCo*, *CoppeliaSim* y el robot *Franka Emika Panda*, asegurando su facilidad de adopción.

La efectividad del software se ha validado mediante un experimento con un manipulador Franka Emika. Los resulta-

dos confirman la capacidad de la herramienta para preservar la ejecución de la tarea en entornos simulados y reales sin alterar el software existente. Las diferencias en las tasas de muestreo entre los bucles del RL y del agente ratifican la naturaleza asíncrona del despliegue.

Como trabajo futuro, se planea delimitar y evaluar explícitamente cada componente temporal para analizar cómo las latencias influyen en la estabilidad del entrenamiento. Asimismo, *RL-Spin-Decoupler* servirá de base para desarrollar algoritmos avanzados que consideren explícitamente duraciones de acción variables y retrasos estocásticos.

Agradecimientos

Este trabajo ha sido financiado por el proyecto TYRELL (PID2023-147392NB-I00), por MICIU/AEI/10.13039/501100011033 y por los Fondos Europeos de Desarrollo Regional (FEDER); ha sido asimismo soportado técnicamente por el IMECH.UMA (PPRO-IUI-2023-02).

Referencias

Al Mahmud, S., Kamarulariffin, A., Ibrahim, A. M., Mohideen, A. J. H., 08 2024. Advancements and challenges in mobile robot navigation: A comprehensive review of algorithms and potential for self-learning approaches. Journal of Intelligent & Robotic Systems 110 (3), 120.
URL: https://doi.org/10.1007/s10846-024-02149-5
DOI: 10.1007/s10846-024-02149-5

Dulac-Arnold, G., Mankowitz, D. J., Hester, T., 2019. Challenges of real-world reinforcement learning. ArXiv abs/1904.12901.

Elguea-Aguinaco, Í., Serrano-Muñoz, A., Chrysostomou, D., Inziarte-Hidalgo, I., Bøgh, S., Arana-Arexolaleiba, N., 2023. A review on reinforcement learning for contact-rich robotic manipulation tasks. Robotics and Computer-Integrated Manufacturing 81, 102517.

Elsner, J., 2023. Taming the panda with python: A powerful duo for seamless robotics programming and integration. SoftwareX 24, 101532.
URL: https://www.sciencedirect.com/science/article/pii/S2352711023002285
DOI: 10.1016/j.softx.2023.101532

Fernández-Madrigal, J.-A., Navarro, A., Asenjo, R., Cruz-Martín, A., 2020. Characterization, statistical analysis and method selection in the two-clocks synchronization problem for pairwise interconnected sensors. Sensors 20 (17).
URL: https://www.mdpi.com/1424-8220/20/17/4808
DOI: 10.3390/s20174808

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1994. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., Levine, S., 2019. Soft actor-critic algorithms and applications.
URL: https://arxiv.org/abs/1812.05905

Haddadin, S., 2024. The franka emika robot: A standard platform in robotics research. IEEE Robotics & Automation Magazine.

Ibarz, J., Tan, J., Finn, C., Kalakrishnan, M., Pastor, P., Levine, S., 2021. How to train your robot with deep reinforcement learning: lessons we have learned. The International Journal of Robotics Research 40, 698 – 721.

IMECH.UMA, 2025. Instituto Universitario de Investigación en Ingeniería Mecatrónica y Sistemas Ciberfísicos, Universidad de Málaga, España. https://www.imech-uma.es/, accedido el 14 de julio de 2025.

Le, H., Saeedvand, S., Hsu, C.-C., 11 2024. A comprehensive review of mobile robot navigation using deep reinforcement learning algorithms in crowded environments. Journal of Intelligent & Robotic Systems 110 (4), 158.
URL: https://doi.org/10.1007/s10846-024-02198-w
DOI: 10.1007/s10846-024-02198-w

Mahmood, A. R., Korenkevych, D., Komer, B., Bergstra, J., 2018. Setting up a reinforcement learning task with a real-world robot. 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 4635–4640.
URL: https://api.semanticscholar.org/CorpusID:3971262

Postel, J., 1981. Transmission control protocol. RFC 793, accessed: 2025-05-22.
URL: https://www.rfc-editor.org/rfc/rfc793

Prasuna, R. G., Potturu, S. R., 08 2024. Deep reinforcement learning in mobile robotics – a concise review. Multimedia Tools and Applications 83 (28), 70815–70836.

Puterman, M. L., 2014. Markov Decision Processes: Discrete Stochastic Dynamic Programming, 1st Edition. John Wiley & Sons.

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.
URL: http://jmlr.org/papers/v22/20-1364.html

Robotics, C., 2024. Coppeliasim. https://www.coppeliarobotics.com/, accessed: 2025-05-08.

Salvato, E., Fenu, G., Medvet, E., Pellegrino, F. A., 2021. Crossing the reality gap: a survey on sim-to-real transferability of robot controllers in reinforcement learning. IEEE Access PP, 1–1.
URL: https://api.semanticscholar.org/CorpusID:243882665

Todorov, E., Erez, T., Tassa, Y., 2012. Mujoco: A physics engine for model-based control. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 5026–5033.
DOI: 10.1109/IROS.2012.6386109

Towers, M., Kwiatkowski, A., Terry, J., Balis, J. U., De Cola, G., Deleu, T., Goulão, M., Kallinteris, A., Krimmel, M., KG, A., et al., 2024. Gymnasium: A standard interface for reinforcement learning environments. arXiv preprint arXiv:2407.17032.

Van Rossum, G., Drake, F. L., 2009. Python 3 Reference Manual. CreateSpace, Scotts Valley, CA.

Yuan, Y., Mahmood, A. R., 2022. Asynchronous reinforcement learning for real-time control of physical robots.
URL: https://arxiv.org/abs/2203.12759

Zhao, W., Queralta, J. P., Westerlund, T., 2020. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. 2020 IEEE Symposium Series on Computational Intelligence (SSCI), 737–744.
URL: https://api.semanticscholar.org/CorpusID:221971078

Zhu, K., Zhang, T., 2021. Deep reinforcement learning based mobile robot navigation: A review. Tsinghua Science and Technology 26 (5), 674–691.
DOI: 10.26599/TST.2021.9010012