COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

MASTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

MEMORIA FINAL

DESARROLLO Y EJECUCIÓN DE TRAYECTORIAS ARTIFICIALES DE ROBOTS INDUSTRIALES INTEGRANDO ALGORITMOS DE RL

Autor: Pablo Santiago Giménez Suárez

Directores:

Álvaro Jesús López López

Ignacio de Rodrigo Tobías

Madrid

Agosto de 2022

Copyright © 2022 Pablo Santiago Giménez Suárez

Este trabajo fue escrito con LaTeX y compilado en Overleaf. Las familias de fuentes usadas son Bitstream Charter, Utopia, Bookman y Computer Modern. A menos que se indique lo contrario, todas las figuras fueron creadas por el autor usando Lucidchart®, draw.io® y Python®.

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Desarrollo y ejecución de trayectorias artificiales de robots industriales integrando algoritmos de RL

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2021/2022 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: _____ Fecha: 16/08/2022

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Ignacio de Rodrigo Tobías Fecha: 22/08/2022

Fdo.: ÁLVARO LÓPEZ 22/08/2022

COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

MASTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

MEMORIA FINAL

DESARROLLO Y EJECUCIÓN DE TRAYECTORIAS ARTIFICIALES DE ROBOTS INDUSTRIALES INTEGRANDO ALGORITMOS DE RL

Autor: Pablo Santiago Giménez Suárez

Directores:

Álvaro Jesús López López

Ignacio de Rodrigo Tobías

Madrid

Agosto de 2022

Copyright © 2022 Pablo Santiago Giménez Suárez

Este trabajo fue escrito con LaTeX y compilado en Overleaf. Las familias de fuentes usadas son Bitstream Charter, Utopia, Bookman y Computer Modern. A menos que se indique lo contrario, todas las figuras fueron creadas por el autor usando Lucidchart®, draw.io® y Python®.

DESARROLLO Y EJECUCIÓN DE TRAYECTORIAS ARTIFICIALES DE ROBOTS INDUSTRIALES INTEGRANDO ALGORITMOS DE RL

Autor: Giménez Suárez, Pablo Santiago
Directores: López López, Álvaro Jesús y Tobías, Ignacio de Rodrigo
Entidad colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

Debido al aumento de la flexibilidad demandada sobre los sistemas de producción, se ha vuelto necesario el desarrollo de nuevas tecnologías que permitan a las máquinas adaptar su comportamiento a cada situación. Es aquí donde la inteligencia artificial y el aprendizaje por refuerzo entran en acción. El objetivo del proyecto es desarrollar una herramienta flexible y compatible con las librerías más extendidas en el sector para permitir la adición y entrenamiento de robots con vistas a una aplicación futura en fábrica.

Palabras clave: Artificial Intelligence, Reinforcement Learning, Pick & Place, Gym, Baselines, Computer generated trajectories

1. Introducción

Hoy en día, la forma más sencilla de explicar la inteligencia artificial es que es la capacidad de máquinas, ordenadores y sistemas de tener comportamientos o habilidades que requieren de cierto nivel de entendimiento. Esto expresado en términos más propios de lenguaje técnico viene a decir que se considera inteligencia artificial a la capacidad de usar algoritmos, procesar datos y aprender de ellos. [Wik22]

Hay tres tipos de aprendizaje en el mundo de la inteligencia artificial: [Gon22]

- Aprendizaje supervisado.
- Aprendizaje no-supervisado.
- Aprendizaje por refuerzo.

El aprendizaje supervisado se basa en el uso de unos datos de entrada y salida conocidos con el objetivo de entrenar un algoritmo capaz. En este caso todos los datos son conocidos. Algunos de los algoritmos más usados serían los de *Regresión Lineal, Árboles de decisión, Redes Neuronales ó Modelos K-NN* y una aplicación práctica sería, por ejemplo, la clasificación de los pacientes de un hospital en función de si reingresaron o no, sabiendo de antemano si lo fueron o no. [Rue22]

En el aprendizaje no-supervisado el funcionamiento es muy parecido, salvo por el hecho de que el modelo entrenado solo tiene en cuenta los datos de entrada, siendo los de salida totalmente desconocidos. Tomando de base el ejemplo anterior de pacientes, el objetivo sería agruparlos pero sin conocer de antemano a qué grupo pertenecen o si hay grupos diferenciables para empezar.

Por último, el aprendizaje por refuerzo se diferencia de los otros modelos en tanto que lo que busca es maximizar o minimizar una medida de recompensa en función de las acciones

tomadas, lo cual vuelve el aprendizaje un problema de optimización. Este tipo de aprendizaje se beneficia enormemente del uso de una memoria que funciona a base de experiencia. Un ejemplo básico sería el caso de tener un péndulo invertido e inestable en donde el objetivo del agente es impedir que caiga, pudiendo solo desplazar la base del péndulo hacia derecha o izquierda.

2. Definición del proyecto

Los objetivos y alcance del proyecto se desarrollan a lo largo de tres puntos principales:

- Desarrollo de una batería de entornos modulares de entrenamiento para diferentes modelos de robot para tareas tipo *Reach* o *Pick&Place*.
- Entrenamiento y comparación de diferentes algoritmos de aprendizaje y los agentes resultantes.
- Extracción de las trayectorias obtenidas para los casos solicitados en ficheros excel con un formato conocido y fácil de interpretar.

La necesidad de que se pretenda generar una batería de entornos y no modificar uno de base durante la ejecución para las distintas situaciones es que el simulador de físicas que se va a utilizar (*Mujoco*) no permite la modificación *on the fly* de los parámetros del archivo xml.

3. Descripción del sistema

El programa a desarrollar se compone de 3 elementos principales. Una librería de entornos de entrenamiento desarrollados en Python y basados en Open AI Gym y un *framework* de entrenamiento basado en RL-Baselines-Zoo[Raf20].

La librería de entornos consiste en una serie de marcos de simulación creados en Mujoco a través de archivos *xml*. La idea es implementar los diferentes códigos necesarios para definir el movimiento general de los robots y personalizar el comportamiento de los robots en cada entorno. Por otro lado, el *framework* de entrenamiento nos permitirá seleccionar el algoritmo y la política empleados en el agente. Además de los distintos parámetros relacionados con el mismo, también nos permitirá almacenar toda la información relevante del proceso de aprendizaje y optimizar los hiperparámetros del sistema.

Por último, se pretende generar una función de ejecución que permita utilizar un agente entrenado en su correspondiente entorno para unas condiciones concretas, de tal manera que se pueda obtener una trayectoria particular.

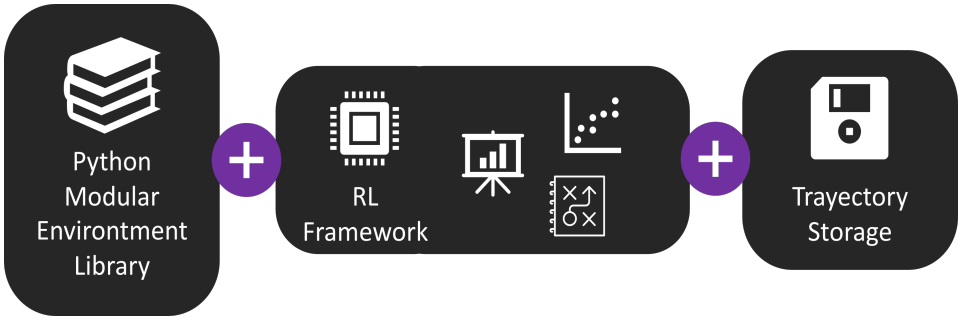


Figura 1. Esquema de la arquitectura del sistema.

4. Resultados

4.1. IRB14000 - Alcance de objetivo

En este primer ensayo se ha entrenado el entorno que pretende únicamente la posición objetivo sin restringir la orientación.

Se ha empleado una recompensa de tipo *sparse* la cual implica que aumenta únicamente cuando se alcanza el objetivo (el episodio es bueno o malo dependiendo de si ha alcanzado el objetivo independientemente de las acciones realizadas o si está más o menos cerca del mismo). Además, se ha usado el algoritmo *DDPG* con apoyo de *HER*, siglas de *Deep Deterministic Policy Gradient* y *Hindsight Experience Replay*.

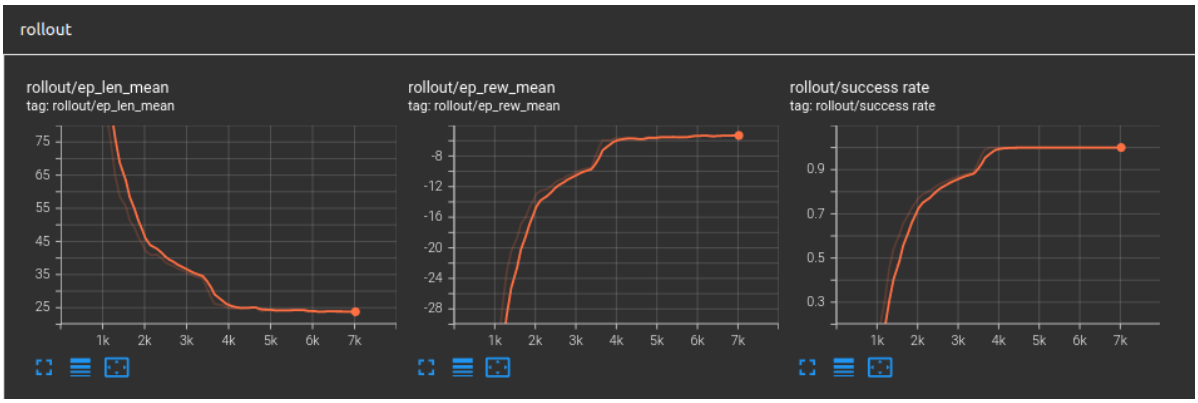


Figura 2. IRB14000 - Reach - DDPG - SPARSE - Entrenamiento

El motivo por el que se ha elegido el tipo de recompensa *sparse* es que generalmente presenta mejores resultados en menos tiempo de entrenamiento. Como podemos observar en la figura Figura 2, en escasos diez mil episodios, hemos alcanzado una tasa de acierto del 100 %

4.2. IRB14000 - Alcance de objetivo y orientación

En los últimos dos ensayos se ha utilizado el entorno avanzado del irb14000 que pretende no solo llegar al objetivo, sino además con una orientación concreta.

Se ha realizado un primer entrenamiento con *DDPG* y *sparse* repitiendo la metodología empleada en el entorno sencillo. Sin embargo, se puede observar en la Figura 3 que en ningún momento se logra alcanzar una tasa de acierto del 100 % y que el valor límite oscila alrededor del 65 %

Por último, se ha intentado realizar el entrenamiento con el algoritmo experimental *TQC* y la recompensa tipo *dense* para comprobar las diferencias.

5. Conclusiones

Como se puede observar a simple vista, las curvas de sendas aproximaciones de aprendizaje presentan comportamiento distintos, pero ninguno logra alcanzar una tasa de acierto máxima. En ambos casos los primeros miles de episodios se realizan sin permitir aprender al algoritmo. Lo que es especialmente destacable es que el algoritmo *TQC* es más estable (tiene menos ruido) al alcanzar su máxima capacidad, mientras que en el caso del algoritmo *DDPG* podemos ver una oscilación fuerte en cuanto llega a su punto máximo de aprendizaje, observándose incluso picos en los que pierde considerablemente su capacidad de acierto.

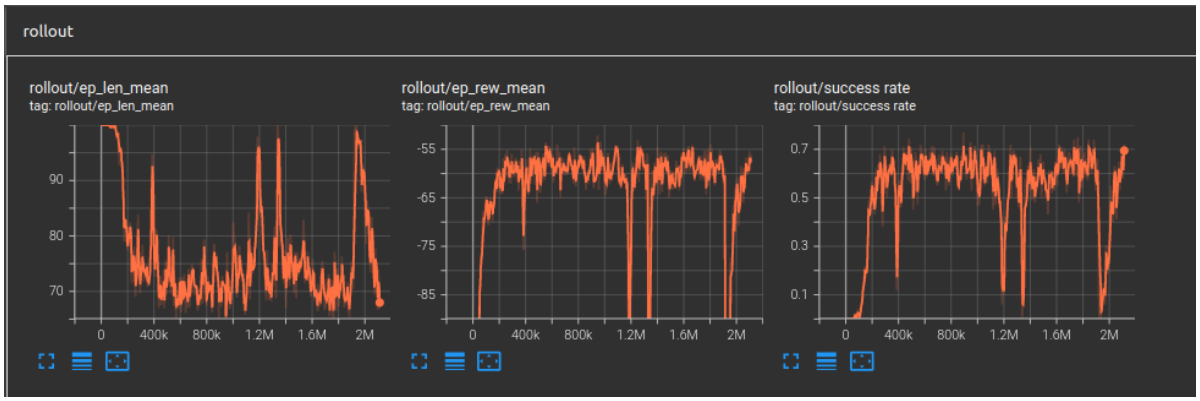


Figura 3. IRB14000 - ReachVec - DDPG - SPARSE - Entrenamiento

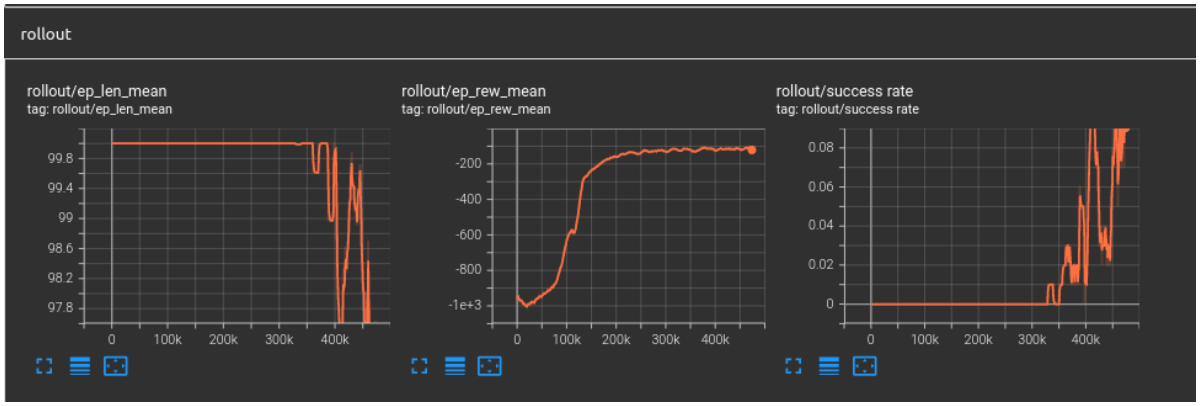


Figura 4. IRB14000 - ReachVec - TQC - DENSE - Entrenamiento

Es importante destacar que los resultados con el algoritmo *TQC* pueden estar sesgados por el reducido número de episodios en comparación con los 2 millones realizados con *DDPG*, pero al tratarse de un algoritmo más pesado, el tiempo empleado para realizar semejante tarea habría sido desproporcionado con los recursos disponibles.

Por último, es crítico remarcar que los resultados obtenidos están claramente sesgados por la configuración del entorno, lo cual engloba entre otros el volumen de generación de objetivos y el hardware empleado. Es posible que la potencia de computación requerida para solucionar el entorno planteado sea superior a la disponible durante el desarrollo de este proyecto.

6. Trabajos futuros

Puesto que el proyecto se presenta como el *esqueleto* de una futura herramienta más avanzada, la idea es desarrollar los siguientes puntos en el futuro:

- Herramiento de adición de CADs/robots a la librería.
- Finalizar la implementación del robot U3Re.
- Almacenar en la propia librería distintos agentes entrenados para los distintos entornos programados
- Implementar una conexión entre el robot simulado y robots reales de tal manera que permitan la ejecución de las trayectorias calculadas de forma fluida.

Bibliografía

[Gon22] J. L. Gonzalez. «Tipos de aprendizaje automático.» (2022), dirección: https://medium.com/soldai/tipos-de-aprendizaje-autom%C3%A1tico-6413e3c615e2.

[Raf20] A. Raffin, *RL Baselines3 Zoo*, https://github.com/DLR-RM/rl-baselines3-zoo, 2020.

[Rue22] J. F. V. Rueda. «Aprendizaje supervisado y no supervisado.» (2022), dirección: https://healthdataminer.com/data-mining/aprendizaje-supervisado-y-no-supervisado/.

[Wik22] L. e. l. Wikipedia. «Aprendizaje por refuerzo.» (2022), dirección: https://es.wikipedia.org/wiki/Aprendizaje_por_refuerzo.

DEVELOPMENT AND EXECUTION OF ARTIFICIAL TRAJECTORIES OF INDUSTRIAL ROBOTS INTEGRATING RL ALGORITHMS

Author: Giménez Suárez, Pablo Santiago
Directors: López López, Álvaro Jesús y Tobías, Ignacio de Rodrigo
Collaborating Entity: ICAI - Comillas Pontifical University

ABSTRACT

Due to the increased flexibility demanded on production systems, new technologies have become necessary to allow machines to adapt their behavior to each situation. This is where artificial intelligence and reinforcement learning come into play. The objective of the project is to develop a flexible tool compatible with the most widespread libraries in the sector to allow the addition and training of robots with a view to a future application in the factory.

Keywords: Artificial Intelligence, Reinforcement Learning, Pick & Place, Gym, Baselines, Computer generated trajectories

1. Introduction

Nowdays the simplest way to explain artificial intelligence is that it is the ability of machines, computers and systems to have behaviors or skills that require a certain level of understanding. Expressed in more technical terms, this means that artificial intelligence is considered to be the ability to use algorithms, process data and learn from them. [Wik22]

There are three types of learning in the world of artificial intelligence: [Gon22]

- Supervised learning.
- Non-supervised learning.
- Reinforcement learning.

Supervised learning is based on the use of known input and output data in order to train a capable algorithm. In this case all the data are known. Some of the most commonly used algorithms would be those of *Linear Regression, Decision Trees, Neural Networks or K-NN Models* and a practical application would be, for example, the classification of patients in a hospital according to whether they were readmitted or not, knowing in advance whether they were readmitted or not. [Rue22]

In unsupervised learning the operation is very similar, except for the fact that the trained model only takes into account the input data, the output data being totally unknown. Based on the previous example of patients, the objective would be to group them but without knowing beforehand to which group they belong or whether there are differentiable groups to begin with.

Finally, reinforcement learning differs from the other models in that it seeks to maximize or minimize a measure of reward as a function of the actions taken, which makes learning an optimization problem. This type of learning benefits greatly from the use of a memory that functions on the basis of experience. A basic example would be the case of having an inverted

and unstable pendulum where the agent's objective is to prevent it from falling, being able only to move the base of the pendulum to the right or left.

2. Project definition

The objectives and scope of the project are developed along three main points:

- Development of a battery of modular training environments for different robot models for *Reach* or *Pick&Place* type tasks.
- Training and comparison of different learning algorithms and the resulting agents.
- Extraction of the trajectories obtained for the requested cases in excel files with a known and easy to interpret format.

The need to generate a battery of environments and not to modify a basic one during the execution for the different situations is that the physics simulator to be used (*Mujoco*) does not allow the modification *on the fly* of the parameters of the xml file.

3. System description

The program to be developed consists of 3 main elements. A library of training environments developed in Python and based on Open AI Gym and a training *framework* based on RL-Baselines-Zoo. [Raf20].

The environment library consists of a series of simulation frames created in Mujoco through *xml* files. The idea is to implement the different codes needed to define the general movement of the robots and to customize the behavior of the robots in each environment. On the other hand, the training *framework* will allow us to select the algorithm and policy used in the agent. In addition to the different parameters related to it, it will also allow us to store all the relevant information of the learning process and to optimize the hyperparameters of the system.

Finally, it is intended to generate an execution function that allows the use of an agent trained in its corresponding environment for specific conditions, so that a particular trajectory can be obtained.

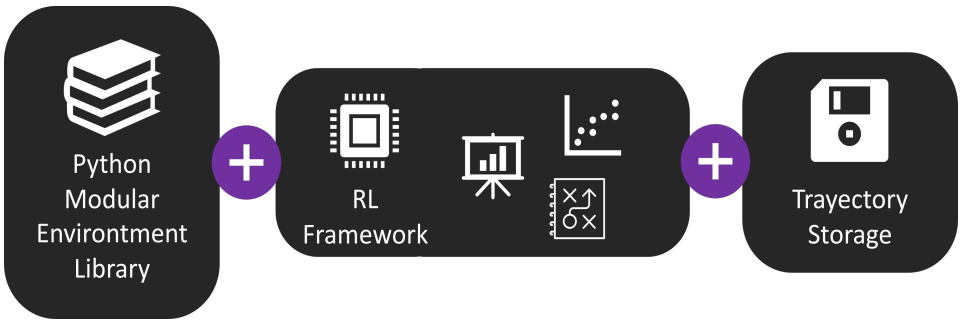


Figure 1. Schematic of the system architecture.

4. Results

4.1. IRB14000 - Target reach

In this first trial we trained the environment aiming only for the target position without restricting the orientation.

A *sparse* type reward has been used which implies that it increases only when the goal is reached (the episode is good or bad depending on whether it has reached the goal regardless of the actions performed or if it is more or less close to the goal). In addition, the *DDPG* algorithm has been used with support from *HER*, short for *Deep Deterministic Policy Gradient* and *Hindsight Experience Replay*.

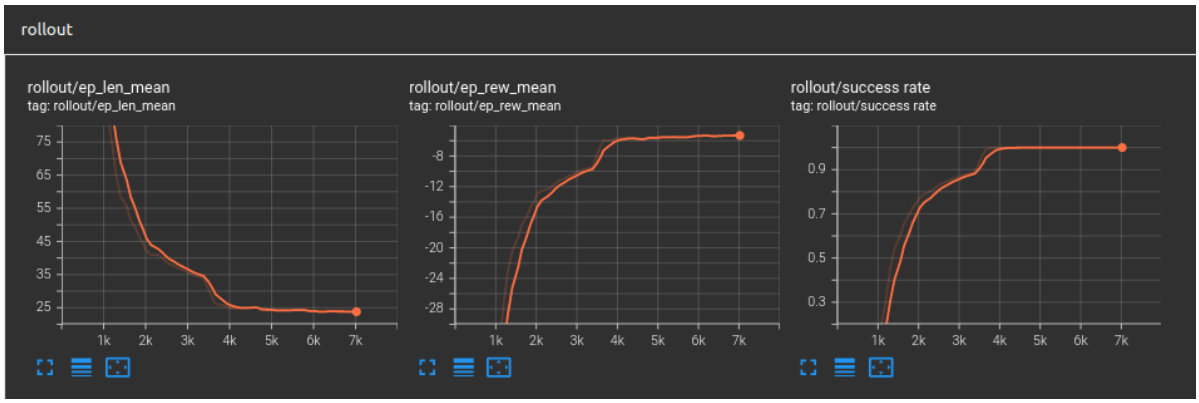


Figure 2. IRB14000 - Reach - DDPG - SPARSE - Training

The reason for choosing the *sparse* reward type is that it generally presents better results in less training time. As we can see in the Figure 2, in just ten thousand episodes we have achieved a hit rate of 100%.

4.2. IRB14000 - Target reach with orientation

In the last two tests, the advanced environment of irb14000 has been used, which aims not only to reach the target, but also with a specific orientation.

A first training has been performed with *DDPG* and *sparse* repeating the methodology used in the simple environment. However in the Figure 3r, it can be observed that a hit rate of 100% is not achieved at any time and that the limit value oscillates around 65%.

Finally, training with the experimental algorithm *TQC* and the reward type *dense* have been tried to test the differences.

5. Conclusions

As it can be seen at a glance, the curves of the two learning approaches show different behavior, but neither achieves a maximum hit rate. In both cases the first few thousand episodes are performed without allowing the algorithm to learn. What is especially remarkable is that the *TQC* algorithm is more stable (has less noise) when it reaches its maximum capacity, while in the case of the *DDPG* algorithm, we can see a strong oscillation when it reaches its maximum learning point, even peaks where it loses considerably its hit capacity.

It is important to note that the results with the *TQC* algorithm may be biased by the small number of episodes compared to the 2 million performed with *DDPG*, but being a heavier

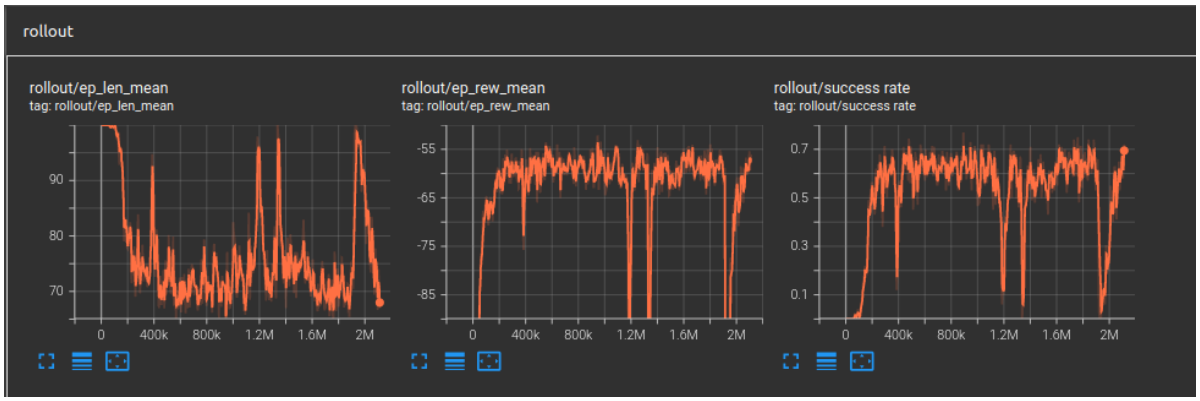


Figure 3. IRB14000 - ReachVec - DDPG - SPARSE - Training

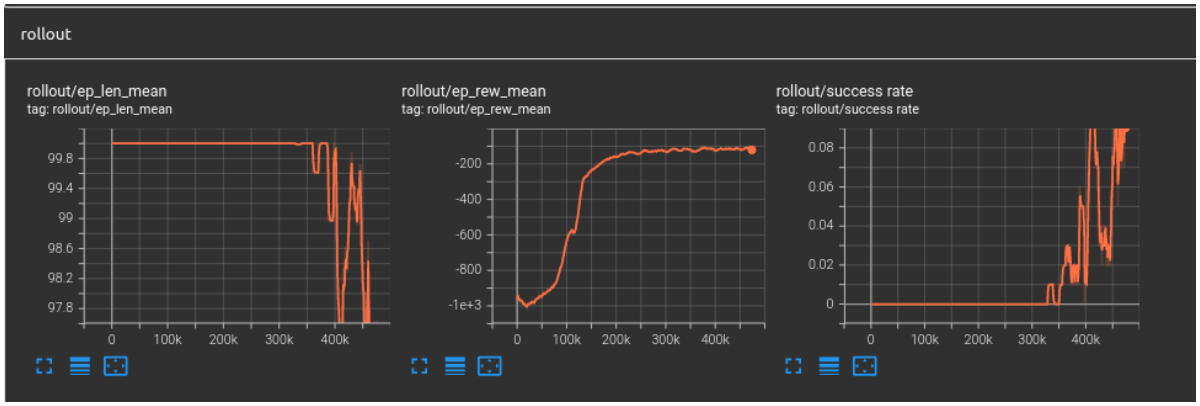


Figure 4. IRB14000 - ReachVec - TQC - DENSE - Training

algorithm, the time taken to perform such a task would have been disproportionate to the available resources.

Finally, it is critical to note that the results obtained are clearly biased by the configuration of the environment, which includes, among others, the volume of target generation and the hardware used. It is possible that the computing power required to solve the proposed environment is higher than that available during the development of this project.

6. Future projects

Since the project is presented as the *skeleton* of a future more advanced tool, the idea is to develop the following points in the future:

- Hardware addition of CADs/robots to the library.
- Finalize the implementation of the U3Re robot.
- Store in the own library different trained agents for the different programmed environments.
- Implement a connection between the simulated robot and real robots in such a way that they allow the execution of the calculated trajectories in a fluent way.

Bibliography

[Gon22] J. L. Gonzalez. “Tipos de aprendizaje automático.” (2022), [Online]. Available: https://medium.com/soldai/tipos-de-aprendizaje-autom%C3%A1tico-6413e3c615e2.

[Raf20] A. Raffin, *Rl baselines3 zoo*, https://github.com/DLR-RM/rl-baselines3-zoo, 2020.

[Rue22] J. F. V. Rueda. “Aprendizaje supervisado y no supervisado.” (2022), [Online]. Available: https://healthdataminer.com/data-mining/aprendizaje-supervisado-y-no-supervisado/.

[Wik22] L. e. l. Wikipedia. “Aprendizaje por refuerzo.” (2022), [Online]. Available: https://es.wikipedia.org/wiki/Aprendizaje_por_refuerzo.

Índice general

Índice de figuras

1

Introducción

Ser autodidacta es, estoy convencido, el único tipo de educación que existe
Isaac Asimov (1920–1992)

En este primer capítulo se realiza una introducción al mundo de la inteligencia artificial y la robótica. Con este proyecto se pretende modernizar las instalaciones del Grupo Antolín® y aumentar las capacidades de la línea de montaje y ensamblaje actual aportando nuevas tecnologías y herramientas de control.

La Inteligencia Artificial como disciplina empieza a desarrollarse a partir de 1950 de la mano de Alan Turing [Cop13] con su artículo *Computing Machinery and Intelligence* en el que desarrollaba la forma de evaluar si una máquina era capaz de pensar y que más tarde se denominó *Test de Turing*. Alan Turing tuvo un papel trascendental en la segunda guerra mundial, aportando una ventaja técnica sin precedentes al permitir descifrar las comunicaciones alemanas a base de fuerza bruta de cálculo. Es considerado, y con razón, uno de los padres de la computación automática. Sin embargo, no fue hasta seis años más tarde con John McCarty, Marvin Misky y Claude Shannon [Ber17], padres de la inteligencia artificial moderna, que se acuñó el termino durante una conferencia en Darmouth, Estados Unidos. [Tri98]

Una de las aplicaciones más importantes de la inteligencia artificial es la robótica. Es en esta especialidad donde la IA puede brillar más, dotando a estos algoritmos constituidos por unos y ceros de un cuerpo físico que les permite interactuar con el medio, abriendo un abanico de posibilidades en la industria. Uno de los referentes más importantes en este campo es sin duda Isaac Asimov, padre de la robótica y quien propuso sus tres leyes fundamentales:

- "Un robot no hará daño a un ser humano ni, por inacción, permitirá que un ser humano sufra daño"
- "Un robot debe cumplir las órdenes dadas por los seres humanos, a excepción de aquéllas que entren en conflicto con la primera ley."
- "Un robot debe proteger su propia existencia en la medida en que esta protección no entre en conflicto con la primera o con la segunda ley."

Es desde 1954 que encontramos la primera patente de lo que podría considerarse un robot y desde entonces se han ido extendiendo y especializando para diferentes tareas y procesos, destacando sobre todo en paletizado y soldadura, así como en tareas de picking o exploración aeroespacial.

Figura 1.1. IRB120 Render.

Uno de los robots más extendidos es el PUMA, término con el que se conoce de forma genérica a los robot de seis ejes. ABB, por ejemplo, tiene su propio PUMA, el IRB120. Y también está en la vanguardia de la robótica con modelos de robot colaborativos, como el YuMi, o su nuevo modelo, el Swifty.

Figura 1.2. Yumi Render.

Otras empresas como Boston Dynamics centran sus esfuerzos en el diseño de robots humanoides o de arquitectura semejante a los animales con extremidades, que equivaldría a conectar varios brazos robóticos a un cuerpo principal y dotarlo de la inteligencia para controlarlas y poder desplazarse por el medio.

Parece evidente, pero el paso de los modelos de robots más sencillos a las complejas estructuras electromecánicas que existen hoy en día ha hecho necesario el uso de algoritmos y software más complejos y pesados a la hora de ser procesado. Es aquí donde entran en funcionamiento las técnicas de aprendizaje por refuerzo y todas sus ramificaciones.

1.1. Motivación

El rápido crecimiento de la industria y el avance de la tecnología han hecho más y más presión sobre las necesidades de flexibilidad en la producción. Es por esto que el desarrollo de algoritmos y sistemas de control capaces de adaptarse y tomar la mejor decisión posible para completar tareas concretas de forma eficiente se ha hecho crítico.

1.2. Objetivos

Los objetivos y alcance del proyecto se desarrollan a lo largo de tres puntos principales:

- Desarrollo de una batería de entornos modulares de entrenamiento para diferentes modelos de robot para tareas tipo *Reach* o *Pick&Place*.
- Entrenamiento y comparación de diferentes algoritmos de aprendizaje y los agentes resultantes.
- Extracción de las trayectorias obtenidas para los casos solicitados en ficheros excel con un formato conocido y fácil de interpretar.

La necesidad de que se pretenda generar una batería de entornos y no modificar uno de base durante la ejecución para las distintas situaciones es que el simulador de físicas que se va a utilizar (*Mujoco*) no permite la modificación *on the fly* de los parámetros del archivo xml.

2

Estado de la cuestión

En este capítulo se desarrollaran las diferentes aproximaciones al cálculo de trayectorias, los diferentes tipos de aprendizaje en los que se desarrolla la inteligencia artificial y el estado de la robótica industrial.

2.1. Inteligencia artificial

Hoy en día, la forma más sencilla de explicar la inteligencia artificial es que es la capacidad de máquinas, ordenadores y sistemas de tener comportamientos o habilidades que requieren de cierto nivel de entendimiento. Esto expresado en términos más propios de lenguaje técnico viene a decir que se considera inteligencia artificial a la capacidad de usar algoritmos, procesar datos y aprender de ellos. [Wik22a]

A lo largo de los años se han ido sucediendo diferentes hitos propiciados por algoritmos avanzados, como en 1997 con la victoria de la supercomputadora *Deep Blue* de IBM sobre el campeón mundial de ajedrez Gari Kasparov o *Alpha Go* de Google que también logró ganar en 2016 al campeón mundial. IBM también desarrolló una IA llamada *Watson* cuyo objetivo era reproducir la cognición humana y ser capaz de comunicarse con el mismo lenguaje. Por si fuera poco, en las últimas décadas gran variedad de empresas han desarrollado sus propias asistentes basados en IA como Google con *Google Now*, Apple con *Siri*, Microsoft con *Cortana* o Samsung con *Sam*. Para poder lograrlo se ha vuelto necesario generar diferentes formas de entrenamiento y aprendizaje optimizadas para distintas aproximaciones. [Por]

2.1.1. Tipos de aprendizaje

Hay tres tipos de aprendizaje en el mundo de la inteligencia artificial: [Gon22]

- Aprendizaje supervisado.
- Aprendizaje no-supervisado.
- Aprendizaje por refuerzo.

El aprendizaje supervisado se basa en el uso de unos datos de entrada y salida conocidos con el objetivo de entrenar un algoritmo capaz. En este caso todos los datos son conocidos. Algunos de los algoritmos más usados serían los de *Regresión Lineal, Árboles de decisión, Redes Neuronales ó Modelos K-NN* y una aplicación práctica sería, por ejemplo, la clasificación de los pacientes de un hospital en función de si reingresaron o no, sabiendo de antemano si lo fueron o no. [Rue22]

En el aprendizaje no-supervisado el funcionamiento es muy parecido, salvo por el hecho de que el modelo entrenado solo tiene en cuenta los datos de entrada, siendo los de salida totalmente desconocidos. Tomando de base el ejemplo anterior de pacientes, el objetivo sería agruparlos pero sin conocer de antemano a qué grupo pertenecen o si hay grupos diferenciables para empezar.

Por último, el aprendizaje por refuerzo se diferencia de los otros modelos en tanto que lo que busca es maximizar o minimizar una medida de recompensa en función de las acciones tomadas, lo cual vuelve el aprendizaje un problema de optimización. Este tipo de aprendizaje se beneficia enormemente del uso de una memoria que funciona a base de experiencia. Un ejemplo básico sería el caso de tener un péndulo invertido e inestable en donde el objetivo del agente es impedir que caiga, pudiendo solo desplazar la base del péndulo hacia derecha o izquierda.

2.2. Robótica industrial

Hoy en día los robots industriales están siendo extensamente usados en casi todas las cadenas de producción. Las dos principales virtudes que los separa de los manipuladores industriales típicos es que son multifuncionales y reprogramables, esto les permite amoldarse al proceso. Debido a esto, se han determinado una serie de características que permiten diferenciar a los distintos tipos de robot:

- Grados de libertad. Lo cual determina el nivel de complejidad del movimiento del robot.
- Zona de trabajo. Algunos robots disponen de un gran alcance y otros lo tienen limitado y viene generalmente determinado por el tamaño de los eslabones y el número de GDL
- Carga a sostener. Tanto el volumen como el peso de los objetos a levantar repercuten directamente en el tamaño y robustez del robot empleado.
- Nivel de programabilidad. No todos los robots cuentan con alta capacidad de procesamiento.

2.2.1. Tipos de robot industial

Los principales tipos de robots en la industria serían por tanto:

- Robot cartesiano.
- Robot Scara.
- Robot Cíclico.
- Robot de seis ejes/PUMA.
- Robot de doble brazo.

Además, en los últimos años han ido cobrando fuerza los llamados robots colaborativos que están expresamente diseñados para trabajar en el mismo entorno que las personas, con velocidades reducidas y menor par en los motores. Algunos ejemplos de esta tecnología los hemos mencionado en el Capítulo 1, tales como el *Swifty, GoFa ó el YuMi*

2.3. Cálculo de trayectorias

La problemática del cálculo de trayectorias siempre ha sido desafiante y compleja. Hoy en día se le exige más que nunca, teniendo que encontrar soluciones en entornos llenos de obstáculos y con cientos de variables a tener en cuenta.

Este tipo de problemas se suele afrontar partiendo del conocimiento de la cinemática directa [Wik22b] e inversa [Wik22c] del robot, lo cual nos proporciona unas ecuaciones generales de movimiento. La cinemática inversa, realmente compleja y con capacidad de dar resultados que bloquean el movimiento del robot, fue solucionada en primera instancia gracias a la caracterización de Denavit-Hartenberg [Wik22d].

Las tecnologías más avanzadas se basan en sistemas de inteligencia artificial entrenadas con aprendizaje por refuerzo, teniendo que afrontar cuatro problemas principales: [Riv19]

- Eficiencia de muestreo.
- Paso de entorno simulado a real.
- Desarrollo de las funciones de recompensa.
- Seguridad entendida en el entorno físico de las personas.

Todos estos puntos son clave y determinan el desarrollo de los diferentes algoritmos. Algunos de los más empleados son *Deep Deterministic Policy Gradient (DDPG)*, *Soft Actor Critic (SAC)* o *Twin Delayed DDPG (TD3)*, siendo este último un sucesor de *DDPG*. *DDPG y TD3* funcionan realmente bien en espacios continuos, mientras que *SAC* obtiene muy buenos resultados en entornos de optimización de energía debido a regularización del factor de entropia que emplea. [Tew20]

Todos los algoritmos previamente mencionados comparten una característica clave y es que son considerados *off-policy algorithms*. La diferencia entre los algoritmos *on-policy y off-policy* es que los primeros evalúan y mejoran la misma política empleada para la selección de acciones en el entorno, esto implica que la política de comportamiento es igual a la política objetivo; mientras que las segundas evalúan y mejoran una política distinta a la empleada para la selección de las acciones en el entorno, es decir, son opuestas. Algunas de las ventajas de las *off-policy* son que permiten la exploración continua, el aprendizaje a través de demostraciones y el aprendizaje paralelo. [Sur20]

2.3.1. Evasión de colisiones

La evasión de colisión no entra en el ámbito de estudio del proyecto a desarrollar, pero es necesario mencionarlo debido a su importancia.

El cambio de paradigma que implica el aprendizaje en entornos abigarrados y cambiantes obliga al desarrollo de nuevos algoritmos o aproximaciones, porque no es posible enseñar a través de métodos convencionales a los diferentes agentes a desplazarse por un entorno dinámico. Algunas soluciones consisten en dividir el espacio de trabajo en celdas dando a conocer al robot aquéllos espacios por lo que tiene permitido el movimiento y aquellos por los que no. Otros se basan en generar nubes de puntos que representan los diferentes obstáculos del entorno a partir de los cuales el robot puede generar una norma de distancia a los objetos más cercanos. [NR21]

3

Metodología de trabajo

En este capítulo se mencionarán las diferentes herramientas que se van a emplear durante el desarrollo del proyecto, la aproximación inicial de la arquitectura del proyecto y el cronograma del mismo.

3.1. Herramientas

Para el desarrollo del proyecto se requiere de un ordenador con los siguientes SO/paquetes/-programas:

- Ubuntu 20.04 LTS.
- PyTorch.
- Python 3.8 o superior.
- Conda ó miniconda.
- Mucojo, ROS u otro entorno de simulación de físicas.
- Open AI Gym.
- Stable Baselines 3. [Raf+21]
- Open AI Spinning Up. [Ach18]
- Tensorboard.
- Cuda para acelerar el proceso de aprendizaje a través de GPU.
- *Opcional* Robot físico para realizar el paso *sim2real*

3.2. Descripción de las herramientas

3.2.1. Ubuntu 20.04

Ubuntu 20.04 es una distribución de Linux y el sistema operativo que se empleará para la realización del proyecto. El principal motivo para seleccionarlo es la compatibilidad con las librerías y herramientas que se van a utilizar para la creación del entorno de aprendizaje y el uso de algorítmos de RL.

Además, cuenta con otras ventajas como son el acceso completo al sistema o ser un sistema operativo 'ligero' comparado con Windows o iOS. Este es uno de los motivos de que sea uno de los sistemas operativos más usados por desrrolladores o productos finales embebidos.

3.2.2. Miniconda

Miniconda es un instalador mínimo gratuito para Conda. Es una versión pequeña de arranque de Anaconda que incluye solo conda, Python, los paquetes de los que dependen y un pequeño número de otros paquetes útiles, incluyendo pip, zlib y algunos otros. La principal ventaja de Conda es que nos permite generar entornos independiente de desarrollo con las diferentes versiones de Python u otros paquetes que sean necesarios para cada proyecto. [20]

3.2.3. Mujoco y mujoco-py

MuJoCo son las siglas de Multi-Joint dynamics with Contact. Se trata de un motor de físicas de propósito general que pretende facilitar la investigación y el desarrollo en robótica, biomecánica, gráficos y animación, aprendizaje automático y otras áreas que exigen una simulación rápida y precisa de estructuras articuladas que interactúan con su entorno. Desarrollado inicialmente por Roboti LLC, fue adquirido y puesto a disposición del público por DeepMind en octubre de 2021 y de código abierto en mayo de 2022. El código base de MuJoCo está disponible en el repositorio deepmind/mujoco en GitHub. [TET12]

Mujoco-py es la librería de python (cython) que permite interactuar con el simulador de físicas y el principal motivo para usar Linux como SO, puesto que la versión de Windows está desfasada y sin mantenimiento.

3.2.4. Open AI Gym y Stable Baselines

Open AI Gym es una API estandarizada que permite implementar el ciclo *agente-entorno*, esto nos aporta una base sobre la que construir nuestros entornos de aprendizaje sabiendo que podrán ser usadas por otras librerías que son compatibles con Gym, como Stable Baselines.

Stable Baselines por su lado se trata de una herramienta de desarrollo de algoritmos de aprendizaje por refuerzo con ayuda de PyTorch. Además de usar entornos basados en Open AI Gym, cuenta con otras dos librerías de apoyo, RL Baselines zoo[Raf20] y sb3-contrib.

El primero es un *framework* de aprenziaje que permite aplicar los distintos algoritmos de Stable Baselines 3 y sb3-contrib, optimizar hiperparámetros de los mismos para cada entorno y supervisar el aprendizaje a través de varios módulos, como Tensorboard. El segundo contiene todos los algoritmos y contribuciones a la librería principal que todavía estan en desarrollo y verificación.

3.3. Arquitectura del sistema

El programa a desarrollar se compone de 3 elementos principales. Una librería de entornos de entrenamiento desarrollados en Python y basados en Open AI Gym y un *framework* de entrenamiento basado en RL-Baselines-Zoo[Raf20].

La librería de entornos consiste en una serie de marcos de simulación creados en Mujoco a través de archivos *xml*. La idea es implementar los diferentes códigos necesarios para definir el movimiento general de los robots y personalizar el comportamiento de los robots en cada entorno. Por otro lado, el *framework* de entrenamiento nos permitirá seleccionar el algoritmo y la política empleados en el agente. Además de los distintos parámetros relacionados con el mismo, también nos permitirá almacenar toda la información relevante del proceso de aprendizaje y optimizar los hiperparámetros del sistema.

Por último, se pretende generar una función de ejecución que permita utilizar un agente entrenado en su correspondiente entorno para unas condiciones concretas, de tal manera que se pueda obtener una trayectoria particular.

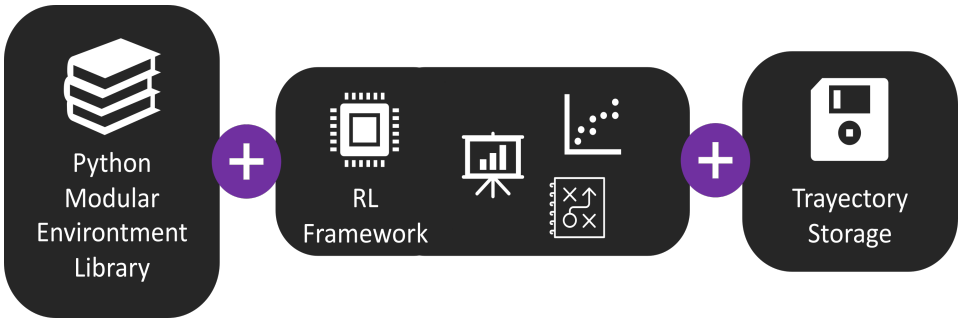


Figura 3.1. Esquema de la arquitectura del sistema.

3.4. Cronograma del proyecto

Desgranando los objetivos prinicpales definidos en la Sección 1.2 obtenemos la lista de tareas a continuación:

- Desarrollo de una batería de entornos modulares de entrenamiento para diferentes modelos de robot para tareas tipo *Reach* o *Pick&Place*.
  - Investigación de los entornos de entrenamiento más usados.
  - Desarrollo de una arquitectura modular de ficheros para el entorno de entrenamiento.
  - Aleatorización controlada.
    - Dotar de variabilidad al entorno para la posición inicial del robot y el objetivo.
    - Desarrollo de la función de recompensa.
  - Generador de modelos xml
    - Modulo de adición de nuevos modelos de robot.
    - Añadir IRB14000 y IRB120.
    - Añadir GoFa y Swifty.
- Entrenamiento y comparación de diferentes algoritmos de aprendizaje y los agentes resultantes.
  - Investigación de las diferentes implementaciones de aprendizaje por refuerzo.
  - Caracterización de los algoritmos de aprendizaje.
  - Análisis de datos.
    - Optimización de hiperparámetros.
    - Extracción de curvas de aprendizaje.
    - Comparación de curvas de aprendizaje de modelos.
  - Validación de agentes entrenados.
- Extracción de las trayectorias obtenidas para los casos solicitados en ficheros excel con un formato conocido y fácil de interpretar.
  - Almacenamiento de trayectorias durante la ejecución.
  - Validación de trayectorias a posteriori.

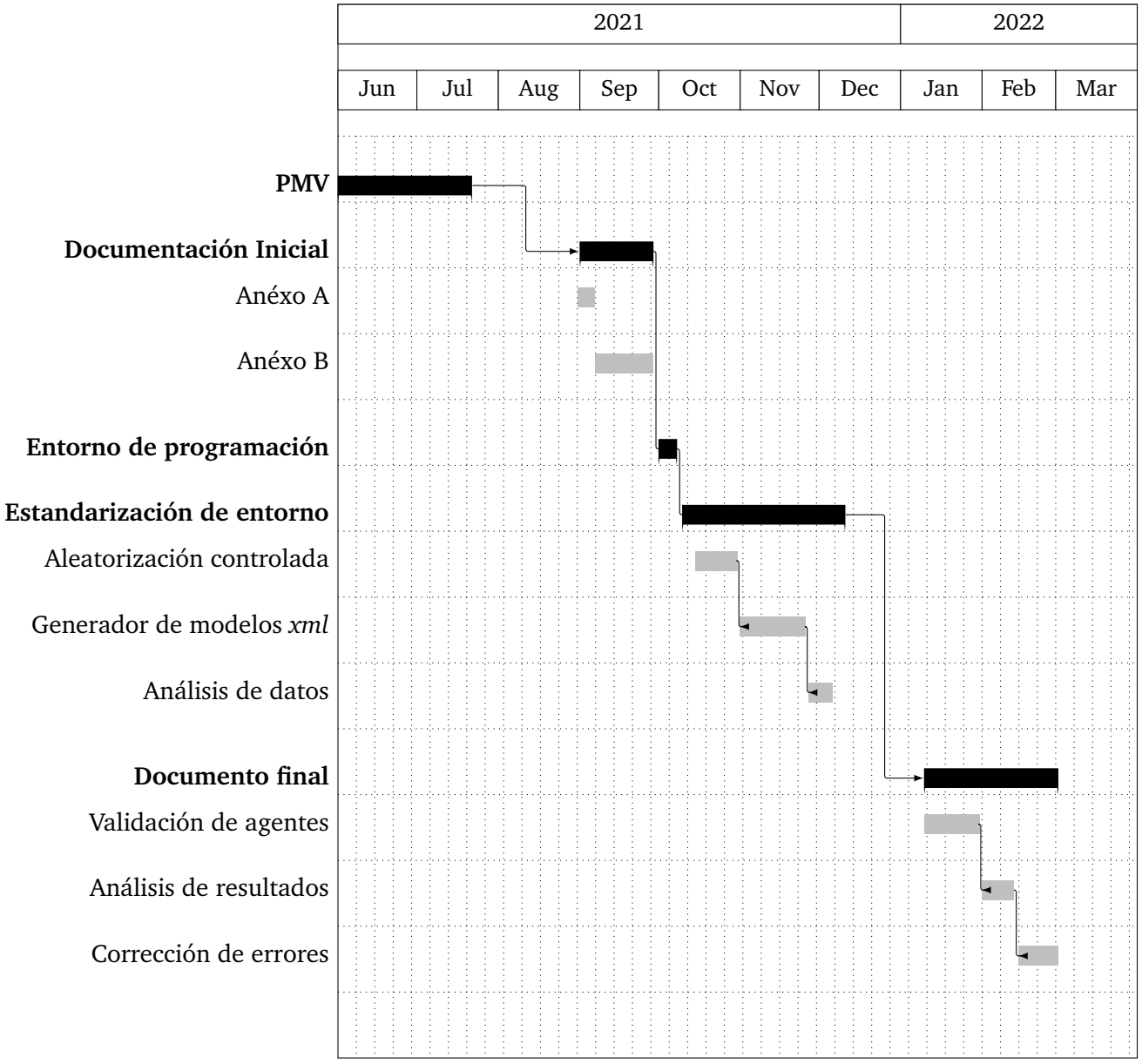


Figura 3.2. Gantt Chart.

4

Robot Gym

En este capítulo se explicará la librería desarrollada para unificar los distintos entornos de aprendizaje para robots.

4.1. Características principales

La librería desarrollada cuenta con varios de los puntos con los que se había planteado el proyecto en un inicio pero no todos. Entre los puntos conseguidos se encuentran:

- Batería de entornos modulares de entrenamiento, actualmente con 2 robots completamente implementados.
- Aleatorización controlada tanto de la posición incial del robot como del objetivo.
- Entornos tipo *reach* que tienen en cuenta posición y rotación de forma independiente y configurable.
- La función de recompensa empleada, al tener en cuenta posición y rotación de forma simulánea, debe ponderar sus valores para no sobre-entrenar el modelo para que, por ejemplo, obtenga la orientación buscada pero la posición alcanza diste mucho del objetivo.
- Almacenamiento de las trayectorias generadas en formato *csv* de forma opcional.
- Se tiene disponible dos tipos de recompensa *sparse* y *dense*

Actualmente, el irb14000 y el irb120 estan implementados con ligeros cambios. Es importante tener en cuenta que al tratarse el irb14000 de un robot con dos brazos independientes requiere de configuración adicional, pero éstas no son las diferencias a las que se hacen mención.

En el caso del irb1400, que fue el primero en implementarse, no cuenta con la opción de aleateorizar la posición inicial del robot y además tiene una gestión distinta de la generación de los *targets* que alcanzar en cada episodio.

Para el irb14000 se buscó limitar el área de generación de objetivos a la zona predilecta de cada brazo, mientras que el irb120 tiene un volumen de aparición de objetivos mayor. En

cualquier caso, se ha evitado la aparción de objetivos en zonas conflictivas como puede ser alrededor del eje principal del robot, formando un volumen cilíndrico restringido en la cercanía de los robots. Este volumen es configurable para cada uno y es así para dar versatilidad y modularidad a la hora de añadir nuevos robots o entornos de aprendizaje.

La mayor parte del código esta presente en el Apéndice A

En general, cualquier entorno definido en esta librería se puede dividir en 3 partes:

- Env-id.
- Clase de python.
- Argumentos de configuración.

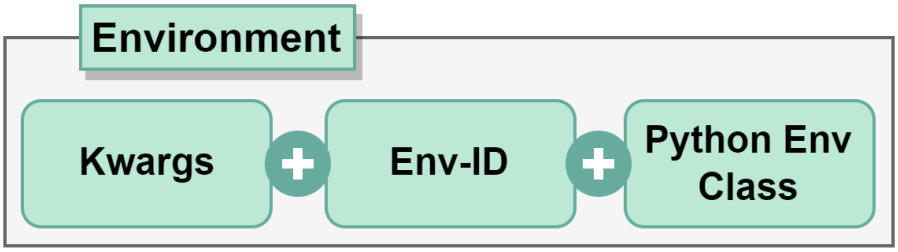


Figura 4.1. Elementos que componen un entorno

Todos los entornos tienen un id único asociado que vincula unos argumentos y una clase de python. Estos argumentos se utilizan para inicializar la clase de python y configurar su funcionamiento al llamar al entorno con ayuda de 'gym.make()'.

Como ejemplo, en el caso del IRB120 y otros entornos de tipo reach, los argumentos configurables serían los siguientes:

- model_name (string): path al modelo XML.
- n_substeps (int): número de *substeps* que se ejecutan en mujoco con cada llamada a *step* del agente.
- default_qpos (dict): Diccionario que contiene el nombre las articulaciones y el valor inicial deseado.
- robot_qpos_conf (string): El tipo de posición inicial del robot:
  - fixed: Posición fija en cada episodio.
  - random: Posición aleatoria en cada episodio.
- target_pos_conf (string): Tipo de posición del objetivo:
  - ignore: La posición del objetivo es ignorada en cada episodio.
  - fixed: La posición del objetivo es fija en cada episodio.
  - random: La posición del objetivo es aleatoria en cada episodio.
- target_rot_conf (string): Tipo de rotación del objetivo:
  - ignore: La orientación del objetivo es ignorada en cada episodio.

  - fixed: La orientación del objetivo es fija en cada episodio.
  - random: La orientación del objetivo es aleatoria en cada episodio.
- target_offset (float or array with 3 elements): Desviación del área de aparición del objetivo
- distance_threshold (float, meters): Distancia a partir de la cual se considera que se ha alcanzado el objetivo.
- rotation_threshold (float, radians): Distancia a partir de la cual se considera que se ha alcanzado la orientación objetivo.
- training (bool): Define si el entorno va a ser entrenado o utilizado.
- reward_type ('sparse' or 'dense'): Define el tipo de recompensa. i.e. sparse o dense
- alpha_reward (float): Peso que equipara el valor de la distancia de la posición con la de orientación.

4.2. Dinámica de funcionamiento

En esta sección se explicará la dinámica de los entornos que componen la librería a la hora de ser utilizados.

Todos los entornos implementados utilizan las bases del aprendizaje por refuerzo con el extendido ciclo entorno-agente.

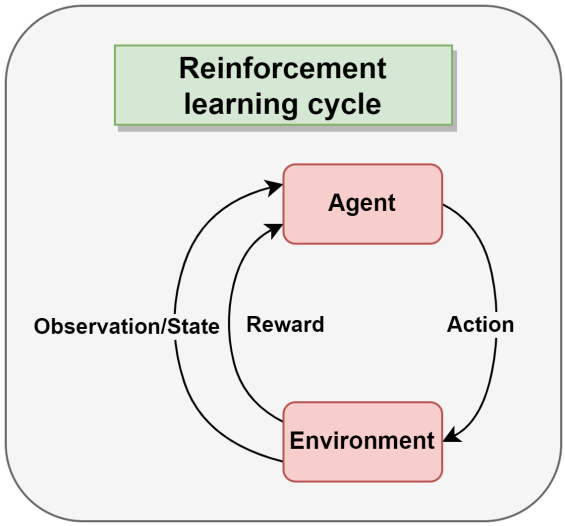


Figura 4.2. Cíclo de aprendizaje por refuerzo

Éste cíclo se repite durante todo el proceso de aprendizaje en lo que se denominan *steps*. La Figura 4.3 ejemplifica este cíclo y los procesos internos que realiza el entorno en cada uno de ellos.

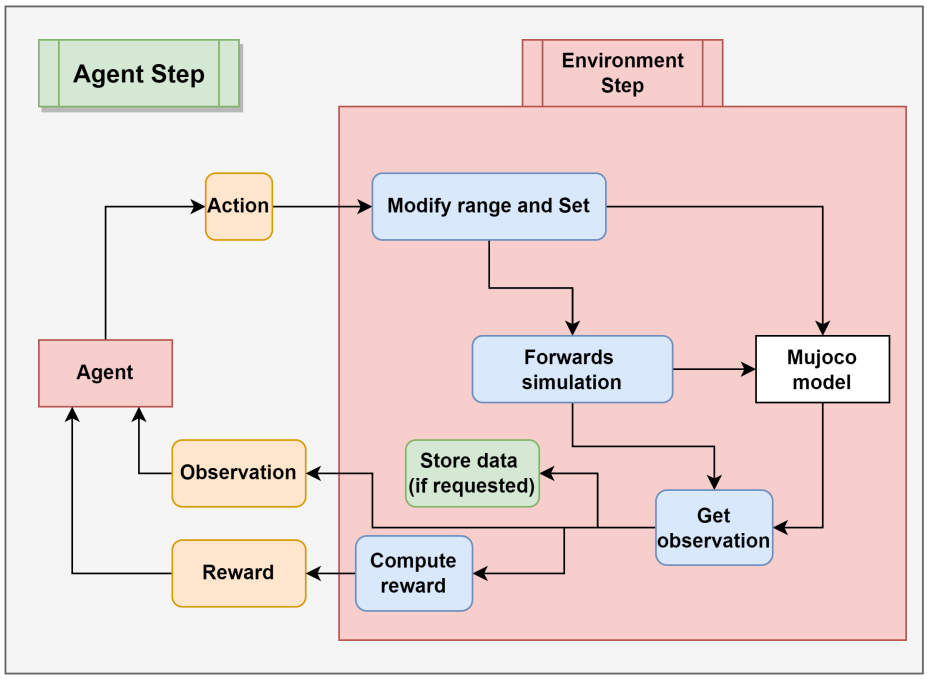


Figura 4.3. Steps del entorno y el agente

Estos *steps* son limitados dentro de un episodio. Un episodio comprende el número de *steps* entre los que se inicializa el entorno y se alcanza el objetivo o se alcanza un limite. El límite de steps esta configurado en el registro de los entornos junto con el Env-ID. La Figura 4.4 resume un episodio de entrenamiento.

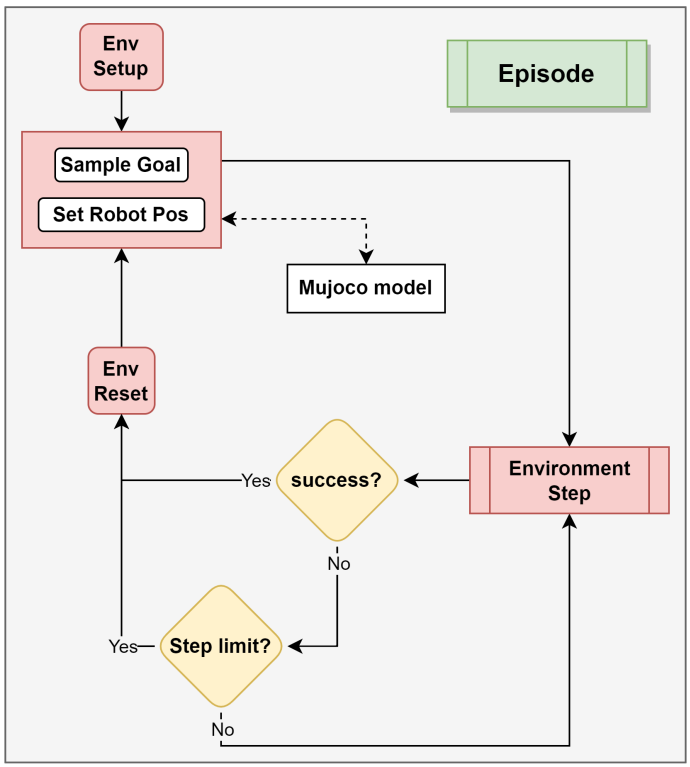


Figura 4.4. Diagrama resumen de un episodio de entrenamiento

4.3. Entorno de trabajo

En esta sección se explicará la distribución de archivos y la funcionalidad estructural de algunos de ellos además del hardware empleado en el desarrollo del proyecto.

Ha habido dos grandes configuraciones de hardware distintas en el desarrollo del proyecto. En primera instancia se utilizó un ordenador con Ubuntu instalado de forma nativa, de tal manera que permitía la utilización de la *GPU* para el proceso de entrenamiento. Sin embargo, en la última fase del proyecto se usó una maquina virtual, lo cual redujo la potencia de aprendizaje. Es por tanto que los resultados presentados en la Capítulo 5 son los obtenidos en la primera fase.

4.3.1. Árbol general de archivos

A continuación se muestra un esquema que representa la distribución general de archivos del proyecto.

```
$HOME/TFM
├── assets
├── .gitignore
├── LICENSE
├── README.md
├── rl-baselines3-zoo
├── robot-gym
├── robot-gym-1.0.0.yml
├── Tensorboard
├── tests
└── tmp
```

Figura 4.5. Árbol general de archivos

El archivo *robot-gym-1.0.0.yml* sirve para generar el entorno de *Conda* correspondiente al proyecto y contiene todas las librerías que utiliza el mismo.

4.3.2. Recursos del proyecto

En esta carpeta se almacenan todos los archivos procesados de los robots en distintas fases de implementación. Por ejemplo, mientras que el irb120, irb14000 y UR3e ya tienen todos los archivos producidos, el CRB15000 (robot colaborativo de ABB) aún esta en la primera fase.

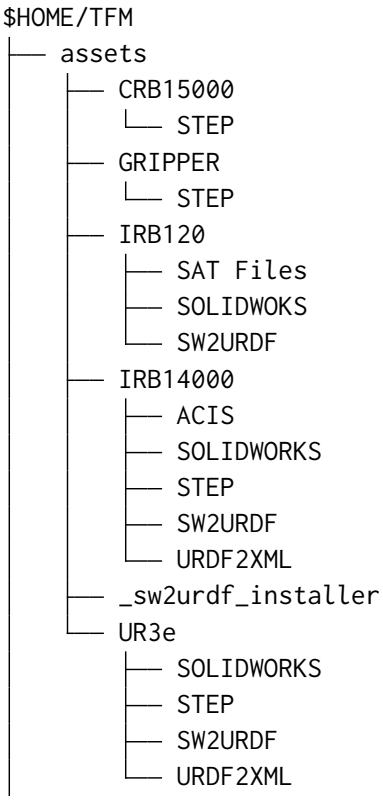


Figura 4.6. Recursos del proyecto

4.3.3. RL-baselines3-zoo

En el esquema de abajo se distinguen las carpetas principales de la librería *RL-baselines3-zoo* que nos proporciona toda la utilidad necesaria para aplicar los distintos algorítmos de aprendizaje, entrenar o probar los agentes y supervisarlos con *Tensorboard*

```
$HOME/TFM
├── rl-baselines3-zoo
│   ├── .gitignore
│   ├── .gitlab-ci.yml
│   ├── .gitmodules
│   ├── hyperparams
│   ├── LICENSE
│   ├── logs
│   │   ├── ddpg
│   │   │   └── Irb120ReachPoint-v0_1
│   │   ├── sac
│   │   │   └── Irb120ReachPoint-v0_1
│   │   ├── td3
│   │   │   ├── Irb120ReachVector-v0_1
│   │   │   ├── Irb120ReachVector-v0_2
│   │   │   ├── Irb120ReachVector-v0_3
│   │   │   ├── Irb120ReachVector-v0_4
│   │   │   └── Irb14000Reach-v0_9
│   │   └── tqc
│   │       ├── FetchReach-v1_1
│   │       ├── Irb120ReachPoint-v0_1
│   │       ├── Irb120ReachPoint-v0_5
│   │       ├── Irb120ReachPoint-v0_6
│   │       ├── Irb120ReachPoint-v0_7
│   │       └── Irb14000Reach-v0_1
│   ├── Makefile
│   ├── README.md
│   ├── requirements.txt
│   ├── scripts
│   ├── setup.cfg
│   ├── tests
│   ├── train.py
│   ├── enjoy.py
│   ├── utils
│   └── version.txt
```

Figura 4.7. Carpeta de RL-baselines3-zoo

En la subcarpeta de *logs* podemos encontrar los diferentes agentes entrenados.

4.3.4. robot-gym

En el esquema a continuación se muestra la jerarquía interna de la librería y los distintos archivos que la componen. La librería proporciona principalmente archivos que definen el funcionamiento de los distintos entornos implementados, entre los que destacan los archivos *.xml*.

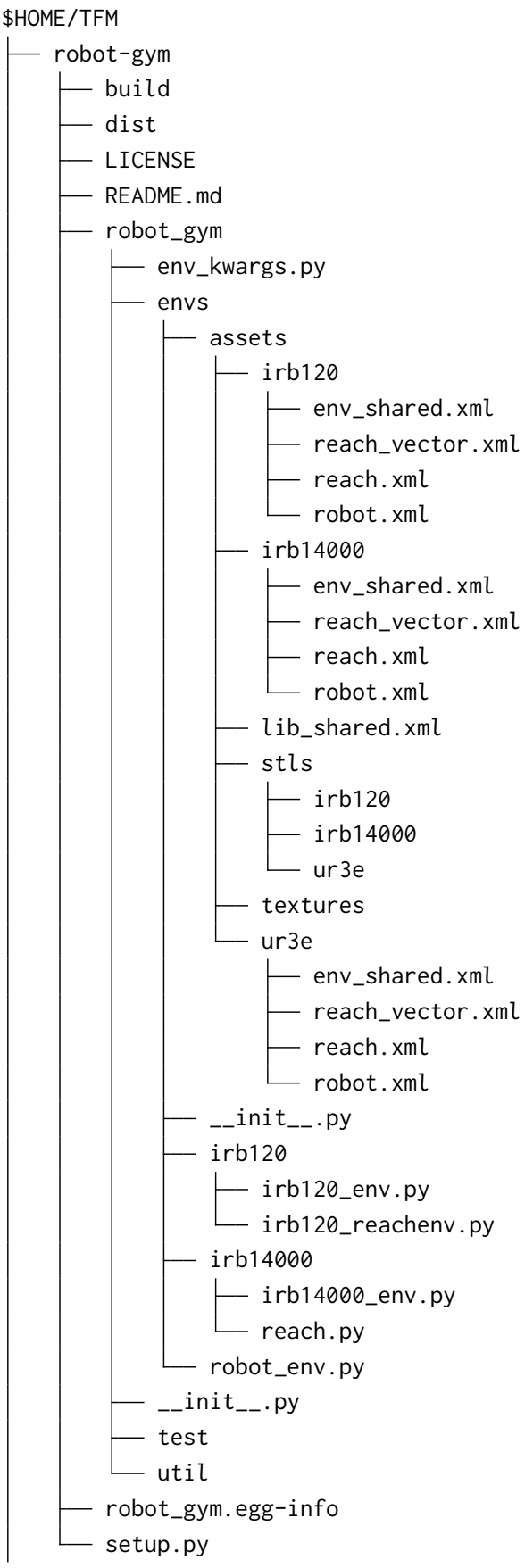


Figura 4.8. robot-gym

4.3.5. Otros

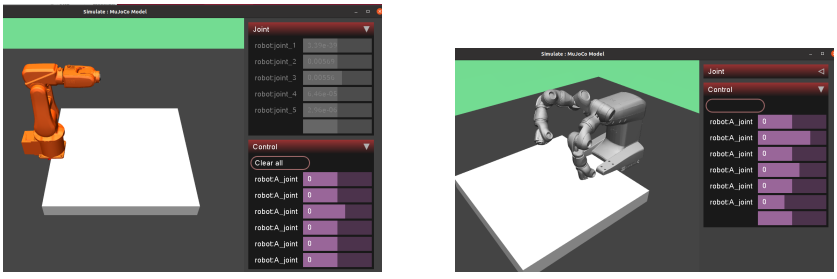
En este último esquema se muestra el resto de archivos del proyecto. Las carpetas *tmp* y *Tensorboard* contienen los resultados de las pruebas de aprendizaje que se analizan en el Capítulo 5

```
$HOME/TFM
├── .gitignore
├── LICENSE
├── README.md
├── robot-gym-1.0.0.yml
├── Tensorboard
│   └── Irb120Reach-v1
├── tests
│   ├── old
│   └── test.py
├── tmp
│   └── stable-baselines
│       ├── FetchReach-v1
│       ├── Irb120ReachPoint-v0
│       ├── Irb120ReachVector-v0
│       └── Irb14000Reach-v0
```

Figura 4.9. Resto de archivos del proyecto

4.4. Recursos CAD y archivos XML

Para el desarrollo del proyecto ha sido necesario adquirir y virtualizar correctamente los modelos CAD de los distintos robots. En el proyecto estan completamente implementados el irb120 y el irb14000 con dos aproximaciones diferentes, tal y como se explica en la Sección 4.1. Además, se han incluido los archivos procesados del UR3e.



(a) IRB120 **(b)** IRB14000

Figura 4.10. Irb120 e Irb14000 in Mujoco

El proceso que se ha seguido para verificar los cad y extraer la información es bastante simple.

- Se descargan los archivos CAD desde la respectiva web o repositorio oficiales
- (OPCIONAL) Los archivos descargados, ya sean el *assembly* o los diferentes elementos del robot por separado, se han de abrir en *Autodeks Fusion 360* para:
    - Verificar la integridad de las mallas.

  - Verificar la disposición de los diferentes elementos.
  - Verificar las propiedades físicas designadas a cada componente.
- De *Autodesk Fusion 360* pasamos a *SOLIDWORKS* que es la herramienta principal para el procesado de los CAD.
  - Instalamos el complemento *SW2URDF*
  - Añadimos ejes de coordenadas en aquellos puntos del robot que vayamos a usar como articulaciones. Lo lógico es seguir las normas propuestas por Denavit Hartenberg [Wik22d], pero no es imprescindible en este caso.
  - Inicializamos el complemente y lo configuramos proporcionando toda la información que nos pida respecto a los ejes, los solidos a usar y sus relaciones.
- Tras seguir los pasos del complemento de *SOLIDWORS* obtendremos varios archivos entre los que se encuentran los .stls y el archivo .urdf del robot.
- Con estos archivos y Mujoco instalado en nuestro sistema podemos exportar el formato a xml con ayuda de la función *compile* que contiene Mujoco.

El proceso completo es explicado en detalle en el Sección D.1.1

Finalmente, se obtiene un modelo en formato .xml que corresponde a un entorno de entrenamiento concreto. La dependencia de archivos sería la mostrada en la Figura 4.11

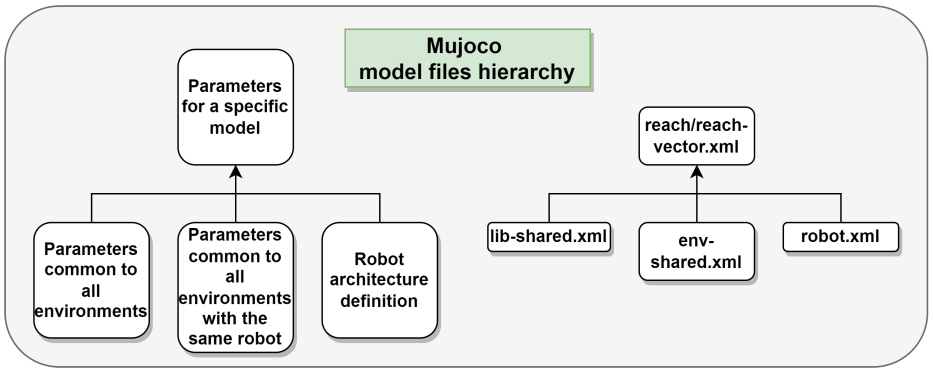


Figura 4.11. Dependencia archivos XML

5

Análisis de resultados

En este capítulo se van a analizar los resultados obtenidos durante el desarrollo del proyecto

Para verificar la funcionalidad de los entornos de desarrollo se ha hecho un código breve presente en Sección A.8. Este código simplemente carga el entorno y ejecuta acciones aleatorias sobre el mismo para verficar que la reinicialización del mismo y la aplicación de las acciones es consistente con lo esperado.

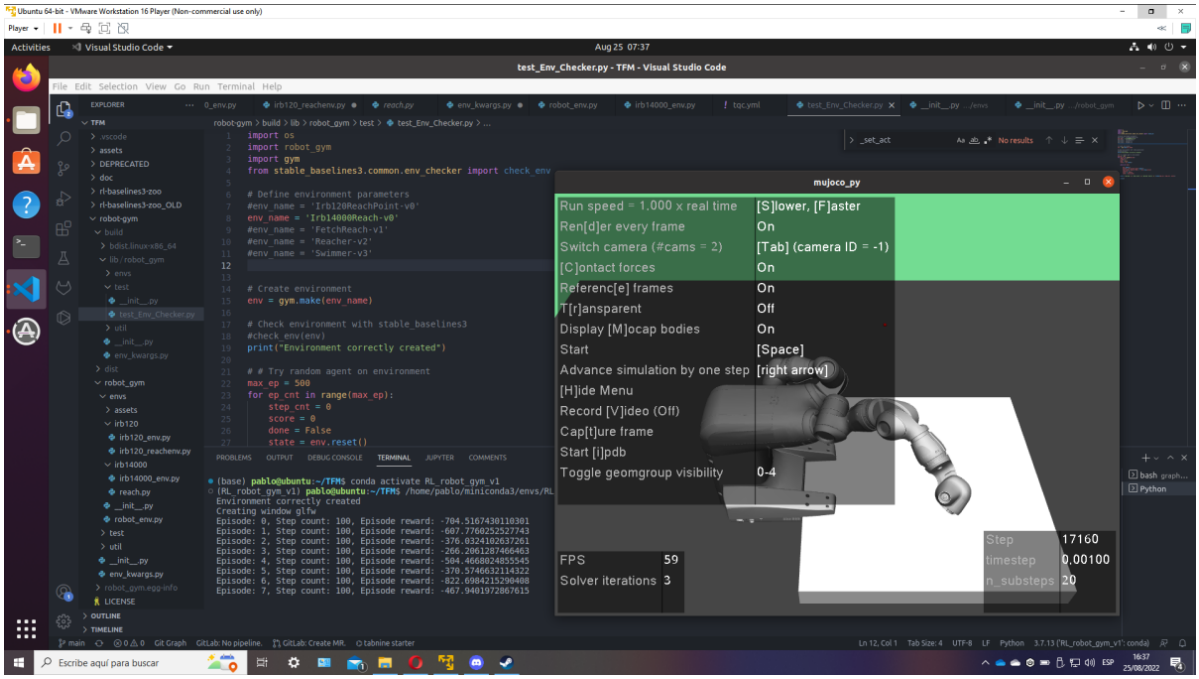


Figura 5.1. Ejemplo de validación de entorno

Una vez hecho esto se ejecutaron distintas pruebas de aprendizaje sobre los entornos. En los próximos apartados entraremos en detalle en tres de ellas.

Durante el aprendizaje el usuario puede supervisar el avance con la ayuda de tensorboard y con la información que se presentan en la linea de comandos:

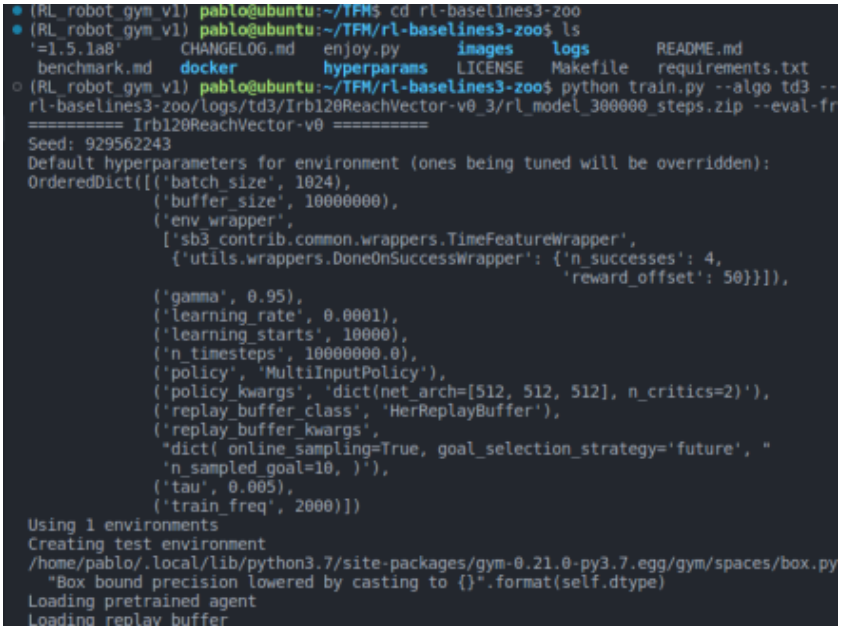


Figura 5.2. Información presentada al usuario al inicializar el aprendizaje

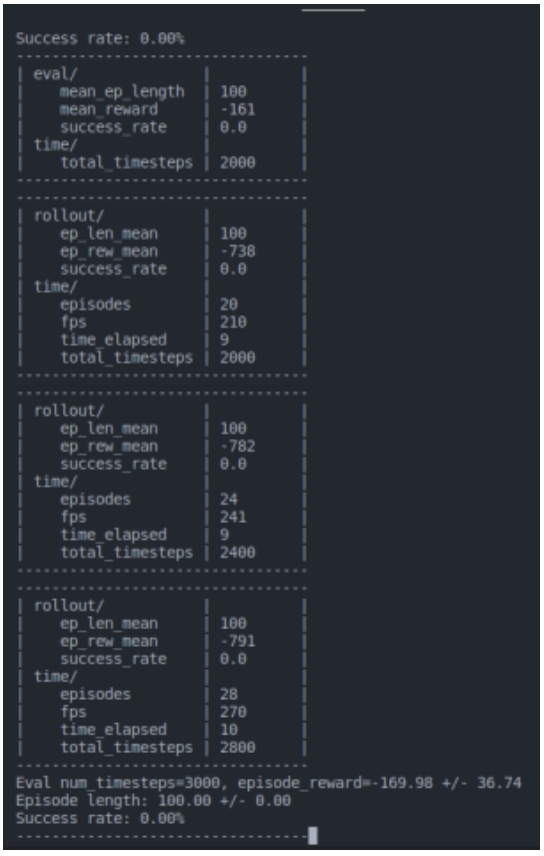


Figura 5.3. Información presentada al usuario durante el aprendizaje

Para más información se recomienda la lectura de la Apéndice D

En cuanto a los hiperparámetros utilizados durante el entrenamiento, estos varían en función del algoritmo empleado. En general se han utilizado los propuestos por *rl-baselines3-zoo* [Raf20] en otros entornos y los obtenidos en la optimización de los mismos con la herramienta de optuna que tiene implementada esta librería. Es importante destacar los wrappers utilizados, que incluyen un wrapper por limite de tiempo de ejecución de sb3-contrib y un wrapper denominado *'DoneOnSuccess'* que permite finalizar el pisodio una vez se ha alcanzado el objetivo.

Un ejemplo de hiperparámetros configurados es el siguiente:

```
Irb14000Reach-v0:
  env_wrapper:
    - sb3_contrib.common.wrappers.TimeFeatureWrapper
    - utils.wrappers.DoneOnSuccessWrapper:
        reward_offset: 0
        n_successes: 4
    - stable_baselines3.common.monitor.Monitor
  n_timesteps: !!float 20000
  policy: 'MultiInputPolicy'
  buffer_size: 1000000
  batch_size: 2048
  gamma: 0.95
  learning_rate: 0.001
  learning_starts: 1000
  noise_type: 'normal'
  noise_std: 0.2
  train_freq: 128
  normalize: True
  tau: 0.005
  replay_buffer_class: HerReplayBuffer
  replay_buffer_kwargs: "dict(
    online_sampling=True,
    goal_selection_strategy='episode',
    n_sampled_goal=4
  )"
  policy_kwargs: "dict(net_arch=[512, 512, 512], n_critics=2)"
```

Figura 5.4. Ejemplo de hiperparámetros para TD3

Para validar los agentes entrenados se ejecutaron los mismos con ayuda de otra de las funciones de *rl-baselines3-zoo* [Raf20] que permite ejecutarlos con semillas de generación distintas al entrenamiento. Una vez se comprueba que en estas pruebas el robot es capaz de alcanzar los objetivos, generados con las mismas reglas que en el entrenamiento, se considera por validado el agente.

Una vez se validaron los agentes entrenados se procedió a implementarlos en robots reales. Para las pruebas se empleó el Irb14000 presente en el laboratorio y el agente explicado en la Sección 5.1. Los resultados fueron casi óptmos. Por desgracia, debido a la utilización de una librería intermedia no se logró realizar la trayectoria fluida sino a intervalos. Esto debería poderse resolver a futuro.

5.1. IRB14000 - Alcance de objetivo

En este primer ensayo se ha entrenado el entorno que pretende únicamente la posición objetivo sin restringir la orientación.

Se ha empleado una recompensa de tipo *sparse* la cual implica que aumenta únicamente cuando se alcanza el objetivo (el episodio es bueno o malo dependiendo de si ha alcanzado el objetivo independientemente de las acciones realizadas o si está más o menos cerca del mismo)

Además, se ha usado el algoritmo *DDPG* con apoyo de *HER*, siglas de *Deep Deterministic Policy Gradient* y *Hindsight Experience Replay*. Estos conceptos se explican brevemente en la Sección 2.3

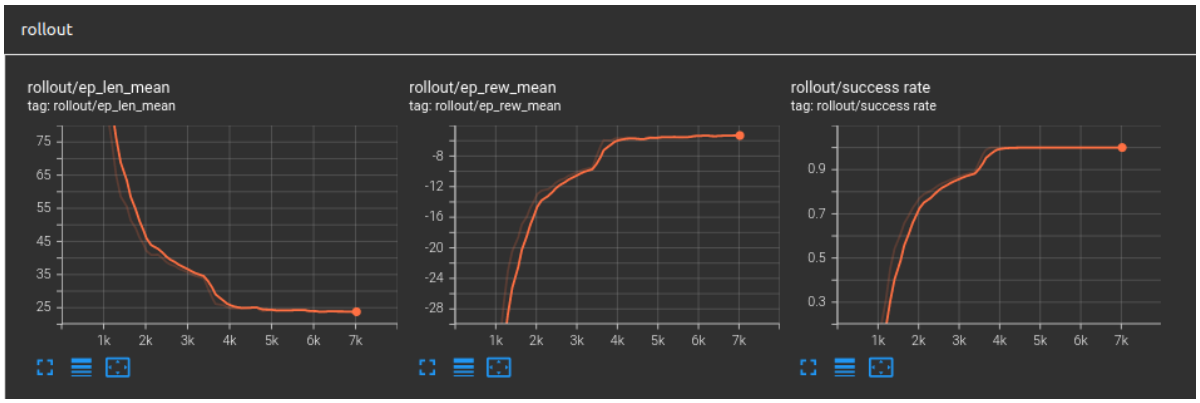


Figura 5.5. IRB14000 - Reach - DDPG - SPARSE - Entrenamiento

El motivo por el que se ha elegido el tipo de recompensa *sparse* es que generalmente presenta mejores resultados en menos tiempo de entrenamiento. Como podemos observar en la figura Figura 5.5, en escasos diez mil episodios, hemos alcanzado una tasa de acierto del 100 %

5.2. IRB14000 - Alcance de objetivo y orientación

En los últimos dos ensayos se ha utilizado el entorno avanzado del irb14000 que pretende no solo llegar al objetivo, sino además con una orientación concreta.

Se ha realizado un primer entrenamiento con *DDPG* y *sparse* repitiendo la metodología empleada en el entorno sencillo. Sin embargo, se puede observar en la Figura 5.6 que en ningún momento se logra alcanzar una tasa de acierto del 100 % y que el valor límite oscila alrededor del 65 %

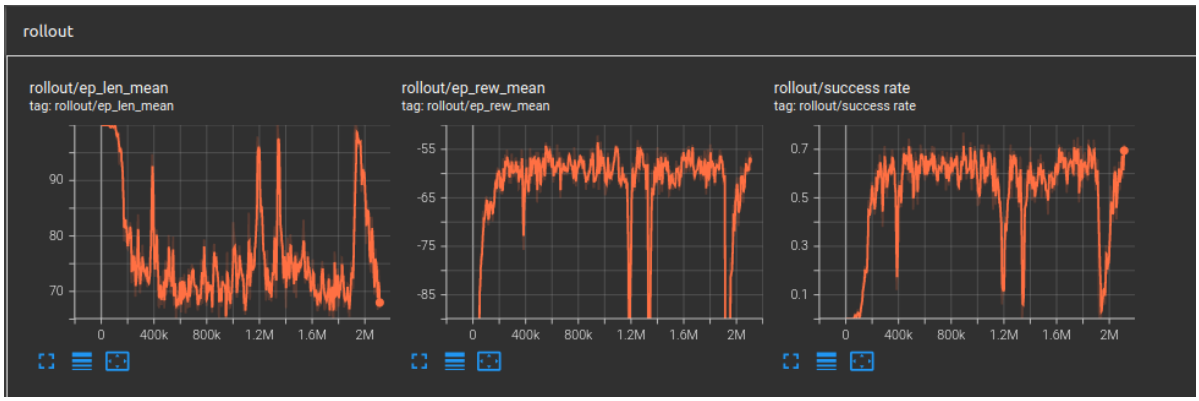


Figura 5.6. IRB14000 - ReachVec - DDPG - SPARSE - Entrenamiento

Por último, se ha intentado realizar el entrenamiento con el algoritmo experimental *TQC* y la recompensa tipo *dense* para comprobar las diferencias.

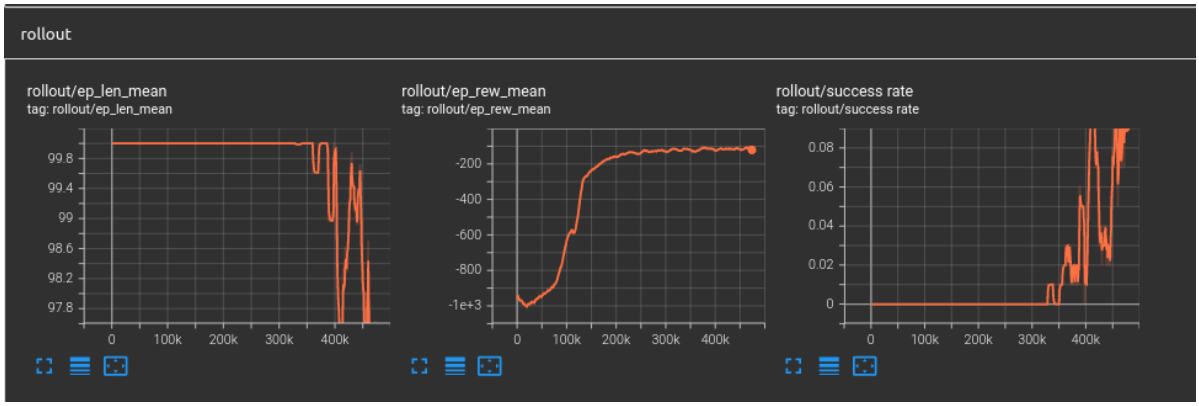


Figura 5.7. IRB14000 - ReachVec - TQC - DENSE - Entrenamiento

Como se puede observar a simple vista, las curvas de sendas aproximaciones de aprendizaje presentan comportamiento distintos, pero ninguno logra alcanzar una tasa de acierto máxima. En ambos casos los primeros miles de episodios se realizan sin permitir aprender al algoritmo. Lo que es especialmente destacable es que el algoritmo *TQC* es más estable (tiene menos ruido) al alcanzar su máxima capacidad, mientras que en el caso del algoritmo *DDPG* podemos ver una oscilación fuerte en cuanto llega a su punto máximo de aprendizaje, observándose incluso picos en los que pierde considerablemente su capacidad de acierto.

Es importante destacar que los resultados con el algoritmo *TQC* pueden estar sesgados por el reducido número de episodios en comparación con los 2 millones realizados con *DDPG*, pero al tratarse de un algoritmo más pesado, el tiempo empleado para realizar semejante tarea habría sido desproporcionado con los recursos disponibles.

Por último, es crítico remarcar que los resultados obtenidos están claramente sesgados por la configuración del entorno, lo cual engloba entre otros el volumen de generación de objetivos y el hardware empleado. Es posible que la potencia de computación requerida para solucionar el entorno planteado sea superior a la disponible durante el desarrollo de este proyecto tal y como se explica en la Sección 4.3.

5.3. Trabajos futuros

Puesto que el proyecto se presenta como el *esqueleto* de una futura herramienta más avanzada, la idea es desarrollar los siguientes puntos en el futuro:

- Herramiento de adición de CADs/robots a la librería. Con la intención de reducir los pasos presentados en el Apéndice C
- Finalizar la implementación del robot U3Re.
- Almacenar en la propia librería distintos agentes entrenados para los distintos entornos programados
- Implementar una conexión entre el robot simulado y robots reales de tal manera que permitan la ejecución de las trayectorias calculadas de forma fluida.

6

Objetivos de desarrollo sostenible

En esta última sección se hace una comparativa entre los objetivos del proyecto y con los objetivos de desarollo sostenible de la Unión Europea

El 25 de septiembre de 2015 se adoptaron una serie de medidas en conjunto con los diferentes líderes mundiales. Cada uno de ellos tiene una serie de metas para 2030 y son una inversión para las generaciones futuras.

Figura 6.1. Objetivos de desarrollo sostenible aprobados en 2015

En total se han planteado 17 objetivos para llevar a cabo. Este proyecto se enmarca dentro de tres de ellos y a continuación se van a desarrollar de forma individual.

- Objetivo 8 - Trabajo decente y crecimiento económico: La implementación de las nuevas herramientras desarrolladas en este proyecto permitirá eliminar puestos de trabajo precario y repetitivo favoreciendo la creación de puestos de personal especializado en robótica o mantenimiento industrial, en donde los riesgos laborales serán menores y los trabajadores podán sentirse más seguros.
- Objetivo 9 - Industria, innovación e infraestructura: Con este proyecto se busca realizar un impacto significativo en las infraestructuras de las fábricas aportando tecnologías innovadoras que permitan un cambio inteligente en la industria.
- Objetivo 12 - Producción y consumo responsable. Con ayuda de los sistemas de inteligencia artificial y la robótica se podrá optimizar el uso del suelo en las fábricas e industrias reduciendo su superficie y aumentando la producción con un consumo más estable y predecible, cualidad en línea con las nuevas tendencias de gestión de la demanda en el sector eléctrico.

Es por todo esto que se considera que este proyecto puede conllevar mejoras para la sociedad a nivel económico y social. Siempre y cuando la implantación de estos sistemas se realicen con respeto y decencia. El objetivo de todo proyecto de innovación no debe ser destruir y rehacer, sino hacer evolucionar los distintos sectores, adapando tanto las técnicas productivas como a las personas involucradas.

A

Anexo - Código de robot-gym

A.1. robot-env.py

```
import os
import copy
import numpy as np

import gym
from gym import error, spaces, utils

try:
    import mujoco_py
except ImportError as e:
    raise error.DependencyNotInstalled("{}. (HINT: you need to install mujoco_py, and also perform the setup instructions here: https://github.com/openai/mujoco-py/.)".format(e))

DEFAULT_SIZE = 500

class RobotEnv(gym.GoalEnv):
    def __init__(self, model_path, default_qpos, frame_skip, seed):
        # Checks if the path to the model exists and load it
        fullpath = os.path.join(os.path.dirname(__file__), 'assets', model_path)
        if not os.path.exists(fullpath):
            raise IOError('File {} does not exist'.format(fullpath))
        model = mujoco_py.load_model_from_path(fullpath)

        # Mujoco definitions
        self.sim = mujoco_py.MjSim(model, nsubsteps=frame_skip)
        self.viewer = None
        self._viewers = {}
        self.metadata = {
            'render.modes': ['human', 'rgb_array'],
            'video.frames_per_second': int(np.round(1.0 / self.dt))
        }

        # Seed of the model
        self.seed(seed)

        # First goal
```

```
36          self.goal = self._sample_goal()
37          # Environment setup
38          self._env_setup(default_qpos=default_qpos)
39          # Store the initial state
40          self.initial_state = copy.deepcopy(self.sim.get_state())
41
42          # Define action space with the number of actuators of the model.
43          self.action_space = spaces.Box(-1., 1., shape=(len(self.sim.model.
    actuator_ctrlrange),), dtype='float32')
44
45          # Define observation space (We get the obs from a method child classes has
     to define)
46          obs = self._get_obs()
47          self.observation_space = spaces.Dict({
48              'goal': spaces.Box(-np.inf, np.inf, shape=obs['goal'].shape, dtype='
    float32'),
49              'goal_diff': spaces.Box(-np.inf, np.inf, shape=obs['goal_diff'].shape,
     dtype='float32'),
50              'joints_angle': spaces.Box(-np.inf, np.inf, shape=obs['joints_angle'].
    shape, dtype='float32'),
51              'joints_angular_velocity': spaces.Box(-np.inf, np.inf, shape=obs['
    joints_angular_velocity'].shape, dtype='float32'),
52          })
53
54      @property
55      def dt(self):
56          return self.sim.model.opt.timestep * self.sim.nsubsteps
57
58      # Env methods
59      # ----------------------------
60      def seed(self, seed=None):
61          self.np_random, seed = utils.seeding.np_random(seed)
62          return [seed]
63
64      def step(self, action):
65          # Force the action to be limited between the action_space limits and then
    applies it to the sim
66          action = np.clip(action, self.action_space.low, self.action_space.high)
67          self._set_action(action)
68
69          # Execute the step
70          self.sim.step()
71
72          # Get the observation post step and check success
73          obs = self._get_obs()
74
75          done = False
76          info = {
77              'is_success': self._is_success(obs['goal_diff']),
78          }
79          reward = self.compute_reward(obs['goal_diff'], info)
80          return obs, reward, done, info
81
82      def reset(self):
83          # Attempt to reset the simulator. Since we randomize initial conditions,
    it
84          # is possible to get into a state with numerical issues (e.g. due to
    penetration or
```

```
85          # Gimbel lock) or we may not achieve an initial condition (e.g. an object
    is within the hand).
86          # In this case, we just keep randomizing until we eventually achieve a
    valid initial
87          # configuration.
88          super(RobotEnv, self).reset()
89          self.goal = self._sample_goal().copy()
90          did_reset_sim = False
91          while not did_reset_sim:
92              did_reset_sim = self._reset_sim()
93          obs = self._get_obs()
94          return obs
95
96      def close(self):
97          if self.viewer is not None:
98              # self.viewer.finish()
99              self.viewer = None
100             self._viewers = {}
101
102     def render(self, mode='human', width=DEFAULT_SIZE, height=DEFAULT_SIZE):
103         self._render_callback()
104         if mode == 'rgb_array':
105             self._get_viewer(mode).render(width, height)
106             # window size used for old mujoco-py:
107             data = self._get_viewer(mode).read_pixels(width, height, depth=False)
108             # original image is upside-down, so flip it
109             return data[::-1, :, :]
110         elif mode == 'human':
111             self._get_viewer(mode).render()
112
113     def _get_viewer(self, mode):
114         self.viewer = self._viewers.get(mode)
115         if self.viewer is None:
116             if mode == 'human':
117                 self.viewer = mujoco_py.MjViewer(self.sim)
118             elif mode == 'rgb_array':
119                 self.viewer = mujoco_py.MjRenderContextOffscreen(self.sim,
    device_id=-1)
120             self._viewer_setup()
121             self._viewers[mode] = self.viewer
122         return self.viewer
123
124     # Extension methods
125     # ----------------------------
126
127     def _reset_sim(self):
128         """Resets a simulation and indicates whether or not it was successful.
129         If a reset was unsuccessful (e.g. if a randomized state caused an error in
     the
130         simulation), this method should indicate such a failure by returning False
    .
131         In such a case, this method will be called again to attempt a the reset
    again.
132         """
133         self.sim.set_state(self.initial_state)
134         self.sim.forward()
135         return True
136
137     def _get_obs(self):
```

```
        """Returns the observation.
        """
        raise NotImplementedError()

    def _set_action(self, action):
        """Applies the given action to the simulation.
        """
        raise NotImplementedError()

    def _is_success(self, achieved_goal):
        """Indicates whether or not the achieved goal successfully achieved the
    desired goal.
        """
        raise NotImplementedError()

    def _sample_goal(self):
        """Samples a new goal and returns it.
        """
        raise NotImplementedError()

    def _env_setup(self, default_qpos):
        """Initial configuration of the environment. Can be used to configure
    initial state
        and extract information from the simulation.
        """
        pass

    def _viewer_setup(self):
        """Initial configuration of the viewer. Can be used to set the camera
    position,
        for example.
        """
        pass

    def _render_callback(self):
        """A custom callback that is called before rendering. Can be used
        to implement custom visualizations.
        """
        pass
```

A.2. irb120-env.py

```
import numpy as np
import csv

from robot_gym.util import rotations
from robot_gym.envs import robot_env

class irb120Env(robot_env.RobotEnv):
    """
    Superclass for all irb120 environments.
    """

    def __init__(
        self,
        model_path, frame_skip, seed,
        default_qpos, robot_qpos_conf,
        target_pos_conf, target_rot_conf, target_offset,
```

```
        distance_threshold, rotation_threshold,
        training, reward_type, alpha_reward,
    ):
        """Initializes a new irb120 environment.

        Args:
            > model_path (string): path to the environments XML file
            > frame_skip (int): number of substeps the simulation runs on every call to step
            > seed (int): random seed

            > default_qpos (dict): a dictionary of joint names and values that define the initial configuration
            > robot_qpos_conf (string): the type of robot jpos:
                - fixed: robot initial jpos fixed on every episode
                - random: robot initial jpos randomized

            > target_pos_conf (string): the type of target position:
                - ignore: target position is fully ignored
                - fixed: target position is set
                - random: target position is fully randomized
            > target_rot_conf (string): the type of target rotation:
                - ignore: target rotation is fully ignored
                - fixed: target rotation is set
                - random: fully randomized target rotation around the X, Y and Z axis
            > target_offset (float or array with 3 elements): offset of the target

            > distance_threshold (float, meters): the threshold after which a goal is considered achieved
            > rotation_threshold (float, radians): the threshold after which the rotation of a goal is considered achieved

            > training (bool): whether or not the model is being trained
            > reward_type ('sparse' or 'dense'): the reward type, i.e. sparse or dense
            > alpha_reward (float): Weight variable for distance and rotation balance

            ### Not implemented ###
            > block_gripper (bool): whether or not the gripper is blocked (i.e. not movable) or not
            > has_obstacle (bool): whether or not the environment has an obstacle
        """

        # Robot configuration
        self._robot_qpos_conf = robot_qpos_conf

        # Target configuration
        self._target_pos_conf = target_pos_conf
        self._target_rot_conf = target_rot_conf
        self._target_offset = target_offset

        # Whenever we are training a RL model or not
        self._training = training

        # Reward paraemeters
        self._distance_threshold = distance_threshold
        self._rotation_threshold = rotation_threshold
```

```
68          self._reward_type = reward_type
69          self._alpha_reward = alpha_reward
70
71          # XML parameters
72          self._TCP_name = 'TCP:center'
73          self._joint_names = default_qpos.keys()
74
75          assert self._target_pos_conf in ['ignore', 'fixed', 'random']
76          assert self._target_rot_conf in ['ignore', 'fixed', 'random']
77
78          super(irb120Env, self).__init__(
79              model_path=model_path,
80              default_qpos=default_qpos.items(),
81              frame_skip=frame_skip,
82              seed=seed,
83              )
84
85      def _get_achieved_goal(self):
86          # TCP position and rotation.
87          pos = self.sim.data.get_site_xpos(self._TCP_name)
88          rot = rotations.mat2quat(self.sim.data.get_site_xmat(self._TCP_name))
89          TCP_qpos = np.concatenate([pos, rot])
90          assert TCP_qpos.shape == (7,)
91          return TCP_qpos
92
93      def _goal_distance(self, goal_a, goal_b):
94          assert goal_a.shape == goal_b.shape
95          assert goal_a.shape[-1] == 7
96
97          d_pos = np.zeros_like(goal_a[..., 0])
98          d_rot = np.zeros_like(goal_b[..., 0])
99          if self._target_pos_conf != 'ignore':
100             delta_pos = goal_a[..., :3] - goal_b[..., :3]
101             d_pos = np.linalg.norm(delta_pos, axis=-1)
102
103         if self._target_rot_conf != 'ignore':
104             quat_a, quat_b = goal_a[..., 3:], goal_b[..., 3:]
105
106             # Subtract quaternions and extract angle between them.
107             quat_diff = rotations.quat_mul(quat_a, rotations.quat_conjugate(quat_b
    ))
108             angle_diff = 2 * np.arccos(np.clip(quat_diff[..., 0], -1., 1.))
109             d_rot = angle_diff
110         assert d_pos.shape == d_rot.shape
111         return d_pos, d_rot
112
113     # GoalEnv methods
114     # ----------------------------
115
116     def compute_reward(self, achieved_goal, goal, info):
117         assert achieved_goal.shape == goal.shape
118         assert goal.shape[-1] == 7
119         if self._reward_type == 'sparse':
120             success = self._is_success(achieved_goal, goal).astype(np.float32)
121             return (success - 1.)
122         elif self._reward_type == 'dense':
123             d_pos, d_rot = self._goal_distance(achieved_goal, goal)
124             # We weigh the difference in position to avoid that `d_pos` (in meters
    ) is completely
```

```
125             # dominated by `d_rot` (in radians).
126             return -(self._alpha_reward*d_pos + d_rot)
127         else:
128             raise error('The reward type defined is not supported')
129 
130     # RobotEnv methods
131     # ----------------------------
132     def _is_success(self, achieved_goal, desired_goal):
133         d_pos, d_rot = self._goal_distance(achieved_goal, desired_goal)
134         achieved_pos = (d_pos < self._distance_threshold).astype(np.float32)
135         achieved_rot = (d_rot < self._rotation_threshold).astype(np.float32)
136         achieved_both = achieved_pos * achieved_rot
137         return achieved_both
138 
139     def _env_setup(self, default_qpos):
140         # Sets the robot joint pos
141         if self._robot_qpos_conf == 'fixed':
142             for name, value in default_qpos:
143                 self.sim.data.set_joint_qpos(name, value)
144         elif self._robot_qpos_conf == 'random':
145                 self._random_qpos(limit_joints=True)
146         else:
147             raise error('The robot joint pos configuration defined is not
    supported')
148         self.sim.forward()
149 
150     def _reset_sim(self):
151         if self._robot_qpos_conf == 'random':
152             self._random_qpos(limit_joints=True)
153         elif self._robot_qpos_conf ==  'fixed':
154             self.sim.set_state(self.initial_state)
155         self.sim.forward()
156         return True
157 
158     def _sample_goal(self):
159 
160         # Calculates a random target if needed
161         if self._target_pos_conf == 'random' or self._target_rot_conf == 'random':
162             r_pos = [0,0,-1]
163             while r_pos[2] < 0:
164                 r_pos, r_rot = self._random_qpos()
165 
166         # Select a goal for the tcp position.
167         if self._target_pos_conf == 'random':
168             target_pos = r_pos
169         elif self._target_pos_conf == 'fixed':
170             target_pos = np.array([0.4, -0.15, 0.3])
171         elif self._target_pos_conf == 'ignore':
172             target_pos = np.zeros(3) # It doesn't matter, it will be ignored
173         else:
174             raise error.Error('Unknown target_position option "{}".'.format(self.
    _target_pos_conf))
175         assert target_pos.shape == (3,)
176 
177         # Select a goal for the tcp rotation.
178         if self._target_rot_conf == 'random':
179             target_quat = r_rot
180         elif self._target_rot_conf == 'fixed':
181             angle = -np.pi
```

```
182             axis = np.array([1., 0., 0.])
183             target_quat = rotations.quat_from_angle_and_axis(angle, axis)
184         elif self._target_rot_conf == 'ignore':
185             target_quat = np.ones(4)
186         else:
187             raise error.Error('Unknown target_rotation option "{}".'.format(self.
    _target_rot_conf))
188         assert target_quat.shape == (4,)
189 
190         target_quat /= np.linalg.norm(target_quat)  # normalized quaternion
191         goal = np.concatenate([target_pos, target_quat])
192         return goal.copy()
193 
194     def _render_callback(self):
195         goal = self.goal.copy()
196         assert goal.shape == (7,)
197         self.sim.data.set_joint_qpos('target:joint', goal)
198         self.sim.data.set_joint_qvel('target:joint', np.zeros(6))
199         self.sim.forward()
200         pass
201 
202     def _step_callback(self):
203         pass
204 
205     def _set_action(self, action):
206         assert action.shape == self.action_space.shape
207 
208         ctrlrange = self.sim.model.actuator_ctrlrange
209         actuation_range = (ctrlrange[:, 1] - ctrlrange[:, 0])/ 2.
210         actuation_center = (ctrlrange[:, 1] + ctrlrange[:, 0]) / 2.
211 
212         self.sim.data.ctrl[:] = actuation_center + action * actuation_range
213         self.sim.data.ctrl[:] = np.clip(self.sim.data.ctrl, ctrlrange[:, 0]*0.1,
    ctrlrange[:, 1]*0.1)
214 
215 
216     def _get_obs(self):
217         # TCP data
218         achieved_goal = self._get_achieved_goal().ravel()  # this contains the TCP
     position + rotation
219         TCP_pos = achieved_goal[:3]
220         TCP_rot = achieved_goal[3:]
221 
222         # Robot joint data
223         robot_qpos, robot_qvel = utils.robot_get_obs(self.sim)
224         robot_state = robot_qpos
225         robot_vel = robot_qvel * self.dt  # change to a scalar if the gripper is
    made symmetric
226 
227         # Saves the trajectory to perform real tasks
228         if not self._training:
229             self.save_trajectory(rotations.quat2mat(TCP_rot), TCP_pos, robot_state
    )
230 
231         obs = np.concatenate([TCP_pos, TCP_rot, robot_state, robot_vel])
232 
233         return {
234             'observation': obs.copy(),
235             'achieved_goal': achieved_goal.copy(),
```

```
            'desired_goal': self.goal.copy(),
        }

    def render(self, mode='human', width=500, height=500):
        return super(irb120Env, self).render(mode, width, height)

    def _random_qpos(self, limit_joints=True):
        # Gets robot joints range
        ctrlrange = self.sim.model.actuator_ctrlrange
        random_qpos = np.zeros(len(ctrlrange))

        # Randomize joint position
        for i, actuator in enumerate(ctrlrange):
            random_qpos[i] = self.np_random.uniform(actuator[0], actuator[1], size
    =1)

        # Limits the randomness of the initial robot joints
        if limit_joints:
            random_qpos[:] = np.clip(random_qpos, ctrlrange[:, 0]*0.5, ctrlrange
    [:, 1]*0.5)

        # Sets the randomized value in the simulation
        for i,_ in enumerate(random_qpos):
            self.sim.data.set_joint_qpos(list(self._joint_names)[i], random_qpos[i
    ])

        # TCP position and rotation.
        pos = self.sim.data.get_site_xpos(self._TCP_name)
        rot = rotations.mat2quat(self.sim.data.get_site_xmat(self._TCP_name))
        return pos,rot

    def save_trajectory(self, rot, pos, state):
        csv_rotations = 'tcp_rotations.csv'
        csv_translations = 'tcp_translations.csv'
        csv_angles = 'robot_angles.csv'

        with open (csv_rotations, 'a+', newline='') as file:
            csv_sheet = csv.writer(file)
            csv_sheet.writerow([rot[0][0], rot[0][1], rot[0][2],
                                rot[1][0], rot[1][1], rot[1][2],
                                rot[2][0], rot[2][1], rot[2][2]])

        with open (csv_translations, 'a+', newline='') as file:
            csv_sheet = csv.writer(file)
            csv_sheet.writerow(map(lambda x: x, pos))


        with open (csv_angles, 'a+', newline='') as file:
            csv_sheet = csv.writer(file)
            csv_sheet.writerow(map(lambda x: x, state))
```

A.3. irb120-reachenv.py

```
import os
from gym import utils
from robot_gym.envs.irb120 import irb120_env
from typing import Dict

```

```
class Irb120ReachEnv(irb120_env.irb120Env, utils.EzPickle):
    def __init__(self,
                 model_name, n_substeps, seed,
                 robot_qpos_conf, target_pos_conf, target_rot_conf,
                 target_offset,
                 distance_threshold, rotation_threshold,
                 training, reward_type, alpha_reward,
                 ):

        """Initializes a new irb120 environment.

        Args:
            > model_name (string): path to the environments XML file
            > n_substeps (int): number of substeps the simulation runs on every call to step

            > default_qpos (dict): a dictionary of joint names and values that define the initial configuration
            > robot_qpos_conf (string): the type of robot jpos:
                - fixed: robot initial jpos fixed on every episode
                - random: robot initial jpos randomized

            > target_pos_conf (string): the type of target position:
                - ignore: target position is fully ignored
                - fixed: target position is set
                - random: target position is fully randomized
            > target_rot_conf (string): the type of target rotation:
                - ignore: target rotation is fully ignored
                - fixed: target rotation is set
                - random: fully randomized target rotation around the X, Y and Z axis
            > target_offset (float or array with 3 elements): offset of the target

            > distance_threshold (float, meters): the threshold after which a goal is considered achieved
            > rotation_threshold (float, radians): the threshold after which the rotation of a goal is considered achieved

            > training (bool): whether or not the model is being trained
            > reward_type ('sparse' or 'dense'): the reward type, i.e. sparse or dense
            > alpha_reward (float): Weight variable for distance and rotation balance

            ### Not implemented ###
            > block_gripper (bool): whether or not the gripper is blocked (i.e. not movable) or not
            > has_obstacle (bool): whether or not the environment has an obstacle
        """
        utils.EzPickle.__init__(self)

        # Mujoco model that is going to be loaded
        MODEL_XML_PATH = os.path.join('irb120', model_name)

        # Initial joint position for the robot
        default_qpos: Dict[str, float] = {
            'robot:joint_1': 0,
            'robot:joint_2': 0,
            'robot:joint_3': 0,
```

```
            'robot:joint_4': 0,
            'robot:joint_5': 0.523599,
            'robot:joint_6': 0,
        }

        irb120_env.irb120Env.__init__(
            self,
            model_path=MODEL_XML_PATH,
            frame_skip=n_substeps,
            seed=seed,
            robot_qpos_conf=robot_qpos_conf,
            target_pos_conf=target_pos_conf,
            target_rot_conf=target_rot_conf,
            default_qpos=default_qpos,
            target_offset=target_offset,
            distance_threshold=distance_threshold,
            rotation_threshold=rotation_threshold,
            training=training,
            reward_type=reward_type,
            alpha_reward=alpha_reward,
            )
```

A.4. irb14000-env.py

```
import numpy as np
import csv
import os
import sys
import math

from gym_abb.envs import rotations, robot_env, utils, error


def robot_get_jointnames(sim):
    """Returns all joint positions and velocities associated with
    a robot.
    """
    if sim.data.qpos is not None and sim.model.joint_names:
        names = [n for n in sim.model.joint_names if n.startswith('robot:joint')]
        return names
    return np.zeros(0)


def quat_from_angle_and_axis(angle, axis):
    assert axis.shape == (3,)
    axis /= np.linalg.norm(axis)
    quat = np.concatenate([[np.cos(angle / 2.)], np.sin(angle / 2.) * axis])
    quat /= np.linalg.norm(quat)
    return quat

class irb14000Env(robot_env.RobotEnv):
    """Superclass for all irb14000 environments.
    """

    def __init__(
        self,
        model_path,
        n_substeps,
```

```
        default_qpos,
        target_position,
        target_rotation,
        target_in_the_air,
        target_offset,
        target_range,
        block_gripper,
        has_obstacle,
        obstacle_range,
        distance_threshold,
        rotation_threshold,
        training,
        reward_type,
        alpha_reward,
    ):
        """Initializes a new irb14000 environment.

        Args:
            model_path (string): path to the environments XML file
            n_substeps (int): number of substeps the simulation runs on every call
     to step
            default_qpos (dict): a dictionary of joint names and values that
    define the initial configuration

            target_position (string): the type of target position:
                - ignore: target position is fully ignored
                - fixed: target position is set
                - random: target position is fully randomized according to
    target_range
            target_rotation (string): the type of target rotation:
                - ignore: target rotation is fully ignored
                - fixed: target rotation is set
                - random: fully randomized target rotation around the X, Y and Z
    axis

            target_in_the_air (bool): whether or not the target should be in the
    air above the table or on the table surface
            target_offset (float or array with 3 elements): offset of the target
            target_range (float): range of a uniform distribution for sampling a
    target

            block_gripper (bool): whether or not the gripper is blocked (i.e. not
    movable) or not
            has_obstacle (bool): whether or not the environment has an obstacle
            obstacle_range (float): range of a uniform distribution for sampling
    initial object positions

            distance_threshold (float, meters): the threshold after which a goal
    is considered achieved
            rotation_threshold (float, radians): the threshold after which the
    rotation of a goal is considered achieved

            training (bool): whether or not the model is being trained
            reward_type ('sparse' or 'dense'): the reward type, i.e. sparse or
    dense
            alpha_reward (float): Weight variable for distance and rotation
    balance
        """

```

```
82          self.target_position = target_position
83          self.target_rotation = target_rotation
84          self.target_in_the_air = target_in_the_air
85          self.target_offset = target_offset
86          self.target_range = target_range
87          self.block_gripper = block_gripper
88          self.has_obstacle = has_obstacle
89          self.obstacle_range = obstacle_range
90          self.distance_threshold = distance_threshold
91          self.rotation_threshold = rotation_threshold
92          self.training = training
93          self.reward_type = reward_type
94          self.alpha_reward = alpha_reward
95
96          assert self.target_position in ['ignore', 'fixed', 'random']
97          assert self.target_rotation in ['ignore', 'fixed', 'random']
98
99          super(irb14000Env, self).__init__(
100             model_path=model_path, n_substeps=n_substeps, n_actions=7,
101             default_qpos=default_qpos)
102
103     def _get_achieved_goal(self):
104         # Object position and rotation.
105         site_name = 'TCP:center'
106         TCP_qpos = np.concatenate([self.sim.data.get_site_xpos(site_name),
    rotations.mat2quat(self.sim.data.get_site_xmat(site_name))])
107         assert TCP_qpos.shape == (7,)
108         return TCP_qpos
109
110     def _goal_distance(self, goal_a, goal_b):
111         assert goal_a.shape == goal_b.shape
112         assert goal_a.shape[-1] == 7
113
114         d_pos = np.zeros_like(goal_a[..., 0])
115         d_rot = np.zeros_like(goal_b[..., 0])
116         if self.target_position != 'ignore':
117             delta_pos = goal_a[..., :3] - goal_b[..., :3]
118             d_pos = np.linalg.norm(delta_pos, axis=-1)
119
120         if self.target_rotation != 'ignore':
121             quat_a, quat_b = goal_a[..., 3:], goal_b[..., 3:]
122
123             # Subtract quaternions and extract angle between them.
124             quat_diff = rotations.quat_mul(quat_a, rotations.quat_conjugate(quat_b
    ))
125             angle_diff = 2 * np.arccos(np.clip(quat_diff[..., 0], -1., 1.))
126             d_rot = angle_diff
127         assert d_pos.shape == d_rot.shape
128         return d_pos, d_rot
129
130     # GoalEnv methods
131     # ----------------------------
132
133     def compute_reward(self, achieved_goal, goal, info):
134         if self.reward_type == 'sparse':
135             success = self._is_success(achieved_goal, goal).astype(np.float32)
136             return (success - 1.)
137         else:
138             d_pos, d_rot = self._goal_distance(achieved_goal, goal)
```

```
            # We weigh the difference in position to avoid that `d_pos` (in meters) is completely
            # dominated by `d_rot` (in radians).
            return -(self.alpha_reward*d_pos + d_rot)

    # RobotEnv methods
    # ----------------------------
    def _is_success(self, achieved_goal, desired_goal):
        d_pos, d_rot = self._goal_distance(achieved_goal, desired_goal)
        achieved_pos = (d_pos < self.distance_threshold).astype(np.float32)
        achieved_rot = (d_rot < self.rotation_threshold).astype(np.float32)
        achieved_both = achieved_pos * achieved_rot
        return achieved_both

    def _env_setup(self, default_qpos):
        # Sets the robot initial position
        for name, value in default_qpos.items():
            self.sim.data.set_joint_qpos(name, value)

        self.sim.forward()

        # Extract information for sampling goals.
        self.range_origin_R = self.sim.data.get_site_xpos('robot:link_0').copy() + self.target_offset #Origin for the right arm

    def _reset_sim(self):
        # Gets robot joint names
        jnames = robot_get_jointnames(self.sim)

        # Gets robot joints actuator range
        ctrlrange = self.sim.model.actuator_ctrlrange
        random_init_qpos = []

        # Randomize actuator initial position at each episode
        for lower, upper in ctrlrange:
            random_init_qpos.append(self.np_random.uniform(lower, upper, size=1).ravel())
        random_init_qpos = np.array(random_init_qpos).ravel()

        # Sets the randomized value in the simulation and forwards it
        for i in range(len(random_init_qpos)):
            self.sim.data.set_joint_qpos(jnames[i], random_init_qpos[i])
        self.sim.forward()

        return True

    def _sample_goal(self):
        site_name = 'TCP:center'
        # Select a goal for the tcp position.
        target_pos = None
        if self.target_position == 'random':
            target_pos = self.range_origin_R[:3] + self.irb14000_range(self.target_range)
        elif self.target_position == 'fixed':
            target_pos = np.array([0.4, -0.15, 0.3])
        elif self.target_position == 'ignore':
            target_pos = self.sim.data.get_site_xpos(site_name)
        else:
```

```
193             raise error.Error('Unknown target_position option "{}".'.format(self.
    target_position))
194         if not self.target_in_the_air:
195             target_pos[2] = 0.12
196 
197         assert target_pos is not None
198         assert target_pos.shape == (3,)
199 
200         # Select a goal for the tcp rotation.
201         target_quat = None
202         if self.target_rotation == 'random':
203             angle = self.np_random.uniform(-np.pi, np.pi)
204             axis = self.np_random.uniform(-1., 1., size=3)
205             target_quat = quat_from_angle_and_axis(angle, axis)
206         elif self.target_rotation == 'fixed':
207             angle = -np.pi
208             axis = np.array([1., 0., 0.])
209             target_quat = quat_from_angle_and_axis(angle, axis)
210         elif self.target_rotation == 'ignore':
211             target_quat = rotations.mat2quat(self.sim.data.get_site_xmat(site_name
    ))
212         else:
213             raise error.Error('Unknown target_rotation option "{}".'.format(self.
    target_rotation))
214         assert target_quat is not None
215         assert target_quat.shape == (4,)
216 
217         target_quat /= np.linalg.norm(target_quat)  # normalized quaternion
218         goal = np.concatenate([target_pos, target_quat])
219         return goal.copy()
220 
221     def _render_callback(self):
222         goal = self.goal.copy()
223         assert goal.shape == (7,)
224         self.sim.data.set_joint_qpos('target:joint', goal)
225         self.sim.data.set_joint_qvel('target:joint', np.zeros(6))
226         self.sim.forward()
227 
228     def _step_callback(self):
229        if self.block_gripper:
230             # self.sim.data.set_joint_qpos('robot:l_gripper_finger_joint', 0.)
231             # self.sim.data.set_joint_qpos('robot:r_gripper_finger_joint', 0.)
232             self.sim.forward()
233 
234     def _set_action(self, action):
235         assert action.shape == (7,)
236 
237         ctrlrange = self.sim.model.actuator_ctrlrange
238         actuation_range = (ctrlrange[:, 1] - ctrlrange[:, 0])/ 2.
239         actuation_center = (ctrlrange[:, 1] + ctrlrange[:, 0]) / 2.
240 
241         self.sim.data.ctrl[:] = actuation_center + action * actuation_range
242         self.sim.data.ctrl[:] = np.clip(self.sim.data.ctrl, ctrlrange[:, 0],
    ctrlrange[:, 1])
243 
244 
245     def _get_obs(self):
246         # TCP data
```

```
247         achieved_goal = self._get_achieved_goal().ravel()  # this contains the TCP
     position + rotation
248         TCP_pos = achieved_goal[:3]
249         TCP_rot = achieved_goal[3:]
250 
251         # Robot joint data
252         dt = self.sim.nsubsteps * self.sim.model.opt.timestep
253         robot_qpos, robot_qvel = utils.robot_get_obs(self.sim)
254         robot_state = robot_qpos[3:10]
255         robot_vel = robot_qvel[3:10] * dt  # change to a scalar if the gripper is
    made symmetric
256 
257         # Saves the trajectory to perform real tasks
258         if not self.training:
259             self.save_trajectory(rotations.quat2mat(TCP_rot), TCP_pos, robot_state
    )
260 
261         obs = np.concatenate([TCP_pos, TCP_rot, robot_state, robot_vel])
262 
263         return {
264             'observation': obs.copy(),
265             'achieved_goal': achieved_goal.copy(),
266             'desired_goal': self.goal.copy(),
267         }
268 
269     def render(self, mode='human', width=500, height=500):
270         return super(irb14000Env, self).render(mode, width, height)
271 
272     def irb14000_range(self, k):
273         lower_limit = 0.123
274         upper_limit = 0.55
275 
276         # Limits the target position to the robot work range
277         if (k > upper_limit):
278             k = upper_limit
279             print("Error: Target range is bigger than the robots. Target range has
     been corrected")
280         elif (k < lower_limit):
281             k = lower_limit
282             print("Error: Target range is smaller than the robots. Target range
    has been corrected")
283 
284         # Generate a random target position for the right arm
285         target_wr = self.np_random.uniform(-k, k, size=3)
286 
287         # Adjust target working area
288         # X
289         if (target_wr[0] < 0):
290             target_wr[0] = -target_wr[1]
291 
292         if (target_wr[0] < lower_limit):
293             target_wr[0] = lower_limit
294 
295         # Y
296         if (target_wr[1] > 0):
297             target_wr[1] = -target_wr[1]
298 
299         if (target_wr[1] > -lower_limit):
300             target_wr[1] = lower_limit
```

```
301
302         # Z
303         if ((target_wr[2] > -lower_limit)&(target_wr[2] < lower_limit)):
304             if (target_wr[2] < 0):
305                 target_wr[2] = -lower_limit
306             else:
307                 target_wr[2] = lower_limit
308
309         return target_wr
310
311     def save_trajectory(self, rot, pos, state):
312         csv_rotations = 'tcp_rotations.csv'
313         csv_translations = 'tcp_translations.csv'
314         csv_angles = 'robot_angles.csv'
315
316         with open (csv_rotations, 'a+', newline='') as file:
317             csv_sheet = csv.writer(file)
318             csv_sheet.writerow([rot[0][0], rot[0][1], rot[0][2],
319                                 rot[1][0], rot[1][1], rot[1][2],
320                                 rot[2][0], rot[2][1], rot[2][2]])
321
322         with open (csv_translations, 'a+', newline='') as file:
323             csv_sheet = csv.writer(file)
324             csv_sheet.writerow(map(lambda x: x, pos))
325
326
327         with open (csv_angles, 'a+', newline='') as file:
328             csv_sheet = csv.writer(file)
329             csv_sheet.writerow(map(lambda x: x, state))
```

A.5. reach.py

```
1  import os
2  from gym import utils
3  from gym_abb.envs.irb14000 import irb14000_env
4
5  class irb14000ReachEnv(irb14000_env.irb14000Env, utils.EzPickle):
6      def __init__(self, model_name, target_position, target_rotation,
       target_in_the_air, target_offset,
7                   block_gripper, has_obstacle, obstacle_range, target_range,
       distance_threshold, rotation_threshold,
8                   training, reward_type, alpha_reward,
9                   ):
10
11         """
12         Args:
13             model_name (string): path to the environments XML file
14             n_substeps (int): number of substeps the simulation runs on every call
        to step
15             default_qpos (dict): a dictionary of joint names and values that
       define the initial configuration
16
17             target_position (string): the type of target position:
18                 - ignore: target position is fully ignored
19                 - fixed: target position is set
20                 - random: target position is fully randomized according to
       target_range
21             target_rotation (string): the type of target rotation:
```

```
22                - ignore: target rotation is fully ignored
23                - fixed: target rotation is set
24                - random: fully randomized target rotation around the X, Y and Z
    axis
25
26            target_in_the_air (bool): whether or not the target should be in the
    air above the table or on the table surface
27            target_offset (float or array with 3 elements): offset of the target
28            target_range (float): range of a uniform distribution for sampling a
    target
29
30            block_gripper (bool): whether or not the gripper is blocked (i.e. not
    movable) or not
31            has_obstacle (bool): whether or not the environment has an obstacle
32            obstacle_range (float): range of a uniform distribution for sampling
    initial object positions
33
34            distance_threshold (float, meters): the threshold after which a goal
    is considered achieved
35            rotation_threshold (float, radians): the threshold after which the
    rotation of a goal is considered achieved
36
37            training (bool): whether or not the model is being trained
38            reward_type ('sparse' or 'dense'): the reward type, i.e. sparse or
    dense
39            alpha_reward (float): Weight variable for distance and rotation
    balance
40        """
41
42        # Mujoco model that is going to be loaded
43        MODEL_XML_PATH = os.path.join('irb14000', model_name)
44
45        # Initial joint position for the robot
46        default_qpos = {
47            'robot:slide0': 0,
48            'robot:slide1': 0,
49            'robot:slide2': 0,
50
51            'robot:joint_1_R': 0,
52            'robot:joint_2_R': -2.268928,
53            'robot:joint_3_R': 0.5235988,
54            'robot:joint_4_R': 0,
55            'robot:joint_5_R': 0.698132,
56            'robot:joint_6_R': 0,
57            'robot:joint_7_R': -2.3561945,
58
59            'robot:joint_1_L': 0,
60            'robot:joint_2_L': -2.268928,
61            'robot:joint_3_L': 0.5235988,
62            'robot:joint_4_L': 0,
63            'robot:joint_5_L': 0.698132,
64            'robot:joint_6_L': 0,
65            'robot:joint_7_L': 2.3561945,
66        }
67
68        irb14000_env.irb14000Env.__init__(
69            self,
70            MODEL_XML_PATH,
71            n_substeps=20,
```

```
            default_qpos=default_qpos,
            target_position=target_position,
            target_rotation=target_rotation,
            target_in_the_air=target_in_the_air,
            target_offset=target_offset,
            target_range=target_range,
            block_gripper=block_gripper,
            has_obstacle=has_obstacle,
            obstacle_range=obstacle_range,
            distance_threshold=distance_threshold,
            rotation_threshold=rotation_threshold,
            training=training,
            reward_type=reward_type,
            alpha_reward=alpha_reward,
            )
        utils.EzPickle.__init__(self)
```

A.6. env-kwargs.py

```
"""
Args:
    > model_name (string): path to the environments XML file
    > n_substeps (int): number of substeps the simulation runs on every call to
    step

    > default_qpos (dict): a dictionary of joint names and values that define the
    initial configuration
    > robot_qpos_conf (string): the type of robot jpos:
        - fixed: robot initial jpos fixed on every episode
        - random: robot initial jpos randomized

    > target_pos_conf (string): the type of target position:
        - ignore: target position is fully ignored
        - fixed: target position is set
        - random: target position is fully randomized
    > target_rot_conf (string): the type of target rotation:
        - ignore: target rotation is fully ignored
        - fixed: target rotation is set
        - random: fully randomized target rotation around the X, Y and Z axis
    > target_offset (float or array with 3 elements): offset of the target

    > distance_threshold (float, meters): the threshold after which a goal is
    considered achieved
    > rotation_threshold (float, radians): the threshold after which the rotation
    of a goal is considered achieved

    > training (bool): whether or not the model is being trained
    > reward_type ('sparse' or 'dense'): the reward type, i.e. sparse or dense
    > alpha_reward (float): Weight variable for distance and rotation balance

    ### Not implemented ###
    > block_gripper (bool): whether or not the gripper is blocked (i.e. not
    movable) or not
    > has_obstacle (bool): whether or not the environment has an obstacle
"""

kwargs_dicts = {
    'irb14000Reach-v0': {
```

```
        'model_name' : 'reach.xml',
        'target_position' : 'random',
        'target_rotation': 'ignore',
        'target_in_the_air': True,
        'target_offset' : [0.1379, -0.1065, 0.462],
        'target_range': 0.4,
        'block_gripper': True,
        'has_obstacle': None,
        'obstacle_range': 0.15,
        'distance_threshold': 0.03,
        'rotation_threshold' : 0.1,
        'training' : True,
        'reward_type' : 'dense',
        'alpha_reward': 10,
        },
    'irb14000Reach-v1': {
        'model_name' : 'reach_vector.xml',
        'target_position' : 'random',
        'target_rotation': 'fixed',
        'target_in_the_air': False,
        'target_offset' : [0.1379, -0.1065, 0.462],
        'target_range': 0.3,
        'block_gripper': True,
        'has_obstacle': None,
        'obstacle_range': 0.15,
        'distance_threshold': 0.03,
        'rotation_threshold' : 0.05,
        'training' : True,
        'reward_type' : 'sparse',
        'alpha_reward': 10,
    },
    'Irb120_ReachPoint': {
        'model_name' : 'reach.xml',
        'n_substeps' : 10,
        'robot_qpos_conf': 'random',
        'target_pos_conf' : 'random',
        'target_rot_conf': 'ignore',
        'target_offset' : [0.1379, -0.1065, 0.462],
        'distance_threshold': 0.03,
        'rotation_threshold' : 0.1,
        'training' : True,
        'reward_type' : 'dense',
        'alpha_reward': 10,
        },
    'Irb120_ReachVector': {
        'model_name' : 'reach_vector.xml',
        'n_substeps' : 10,
        'robot_qpos_conf': 'random',
        'target_pos_conf' : 'random',
        'target_rot_conf': 'random',
        'target_offset' : [0.1379, -0.1065, 0.462],
        'distance_threshold': 0.03,
        'rotation_threshold' : 0.1,
        'training' : True,
        'reward_type' : 'dense',
        'alpha_reward': 10,
        },
}
```

A.7. main __init__.py

```
""" Register environments in Gym

Irb120_ReachPoint: Irb120 environment where the robot hast to reach a random point
     from a random pos

Irb120_ReachVector:  Irb120 environment where the robot hast to reach a random
    point and angle from a random pos

"""

from gym.envs.registration import  register
from .env_kwargs import kwargs_dicts

# register(
#     id='Irb14000Reach-v0',
#     entry_point='gym_abb.envs:irb14000ReachEnv',
#     kwargs=kwargs_dicts['irb14000Reach-v0'],
#     max_episode_steps=100,
# )

# register(
#     id='Irb14000Reach-v1',
#     entry_point='gym_abb.envs:irb14000ReachEnv',
#     kwargs=kwargs_dicts['irb14000Reach-v1'],
#     max_episode_steps=100,
# )

register(
    id='Irb120_ReachPoint',
    entry_point='gym_abb.envs:Irb120ReachEnv',
    kwargs=kwargs_dicts['Irb120_ReachPoint'],
    max_episode_steps=100,
)

register(
    id='Irb120_ReachVector',
    entry_point='gym_abb.envs:Irb120ReachEnv',
    kwargs=kwargs_dicts['Irb120_ReachVector'],
    max_episode_steps=100,
)
```

A.8. Env-Checker

```
import os
import robot_gym
from stable_baselines3.common.env_checker import check_env

# Define environment parameters
env_name = 'YumiReach-v0'

# Create environment
env = gym.make(env_name)

# Check environment with stable_baselines3
check_env(env)

```

```
14
15 # Try random agent on environment
16 obs = env.reset()
17 n_steps = 10
18 for _ in range(n_steps):
19     # Random action
20     action = env.action_space.sample()
21     obs, reward, done, info = env.step(action)
22     if done:
23         obs = env.reset()
```

B

Anexo - Recursos en formato XML de robot-gym

B.1. lib-shared.xml

```
<mujoco>
    <asset>
        <texture type="skybox" builtin="gradient" rgb1="0.44 0.85 0.56" rgb2="0.46 0.87 0.58" width="32" height="32"></texture>
        <texture name="texture_block" file="block.png" gridsize="3 4" gridlayout=".U..LFRB.D.."></texture>
        <texture name="texture:hidden" file="block_hidden.png" gridsize="3 4" gridlayout=".U..LFRB.D.."></texture>

        <material name="material:hidden" texture="texture:hidden" specular="1" shininess="0.3" reflectance="0"></material>
        <material name="floor_mat" specular="0" shininess="0.5" reflectance="0" rgba="0.2 0.2 0.2 1"></material>
        <material name="table_mat" specular="0" shininess="0.5" reflectance="0" rgba="0.93 0.93 0.93 1"></material>
        <material name="block_mat" specular="0" shininess="0.5" reflectance="0" rgba="0.2 0.2 0.2 1"></material>
        <material name="puck_mat" specular="0" shininess="0.5" reflectance="0" rgba="0.2 0.2 0.2 1"></material>
        <material name="robot:geomMat" shininess="0.03" specular="0.4"></material>
        <material name="robot:arm_mat" shininess="0.03" specular="0.4" reflectance="0"></material>
        <material name="robot:head_mat" shininess="0.03" specular="0.4" reflectance="0"></material>
        <material name="robot:base_mat" shininess="0.03" specular="0.4" reflectance="0"></material>

    </asset>

</mujoco>
```

B.2. irb120

B.2.1. irb120 env-shared.xml

```xml
<mujoco>

    <!-- Mesh loading for each robot link -->
    <asset>
        <mesh file="link_0.stl" name="robot:link_0"></mesh>
        <mesh file="link_1.stl" name="robot:link_1"></mesh>
        <mesh file="link_2.stl" name="robot:link_2"></mesh>
        <mesh file="link_3.stl" name="robot:link_3"></mesh>
        <mesh file="link_4.stl" name="robot:link_4"></mesh>
        <mesh file="link_5.stl" name="robot:link_5"></mesh>
        <mesh file="link_6.stl" name="robot:link_6"></mesh>

        <!-- Arrow target for complex reach environments -->
        <mesh file="arrow.stl" name="arrow"></mesh>

        <!-- Gripper should be added for pick and place environments -->
    </asset>

    <!-- Default definition of the robot. This will help with all the internal
    element definition -->
    <default>
        <default class="robot">
            <geom margin="0.001" material="robot:geomMat" rgba="1 0.3 0 1" solimp=
    "0.99 0.99 0.01" solref="0.01 1" type="mesh"></geom>
            <joint armature="0.01" damping="100" frictionloss="0"></joint>
            <general ctrllimited="true"></general>
            <position kp="1000" gear="1"></position>
         </default>
         <!--  <default class="Gripper">
                <geom condim="4" margin="0.001" type="box" user="0" rgba="0.356
    0.361 0.376 1.0"></geom>
                <joint armature="100" damping="1000" limited="true" solimplimit
    ="0.99 0.999 0.01" solreflimit="0.01 1" type="slide"></joint>
            </default>
        -->

    </default>

    <!-- We should exclude the contact between consecutive links to avoid unwanted
     collisions -->
    <contact>
        <exclude body1="robot:link_0" body2="robot:link_1"></exclude>
        <exclude body1="robot:link_1" body2="robot:link_2"></exclude>
        <exclude body1="robot:link_2" body2="robot:link_3"></exclude>
        <exclude body1="robot:link_3" body2="robot:link_4"></exclude>
        <exclude body1="robot:link_4" body2="robot:link_5"></exclude>
        <exclude body1="robot:link_5" body2="robot:link_6"></exclude>
    </contact>

    <!-- Position actuator definition -->
    <actuator>
        <position name="robot:A_joint_1" class="robot" joint="robot:joint_1"
    ctrlrange="-2.87979 2.87979"></position>
        <position name="robot:A_joint_2" class="robot" joint="robot:joint_2"
    ctrlrange="-1.91986 1.91986"></position>
        <position name="robot:A_joint_3" class="robot" joint="robot:joint_3"
    ctrlrange="-1.91986 1.221730"></position>
        <position name="robot:A_joint_4" class="robot" joint="robot:joint_4"
    ctrlrange="-2.7925268 2.7925268"></position>
```

```
        <position name="robot:A_joint_5" class="robot" joint="robot:joint_5"
    ctrlrange="-2.094395 2.094395"></position>
        <position name="robot:A_joint_6" class="robot" joint="robot:joint_6"
    ctrlrange="-6.981317 6.981317"></position>
    </actuator>

</mujoco>
```

B.2.2. irb120 robot.xml

```
<mujoco model="irb120">
  <!-- IRB120 body -->
  <body childclass="robot" name="robot:link_0" pos="0 0 0">
    <inertial pos="-0.04204 8.025e-05 0.079638" quat="0.579606 0.411479 0.40617
    0.574255" mass="6.2151" diaginertia="0.0270343 0.0252804 0.0245393" />
    <geom mesh="robot:link_0" name="robot:link_0" material="robot:base_mat" />
    <!-- Robot base location site -->
    <site name="robot:link_0" pos="0 0 0" size="0.01 0.01 0.01"></site>

    <!-- IRB120 ARM -->
    <!-- Link_1 -->
    <body name="robot:link_1" pos="0 0 0.166">
          <inertial pos="9.7659e-05 -0.00011924 0.072412" quat="0.706982
    -0.00180192 0.00178823 0.707227" mass="3.067" diaginertia="0.0124361 0.012269
    0.0104209" />
          <joint name="robot:joint_1" pos="0 0 0" axis="0 0 1" limited="true"
    range="-2.87979326579064 2.87979326579064"/>
          <geom mesh="robot:link_1" material="robot:arm_mat" name="robot:link_1"
     />

      <!-- Link_2 -->
      <body name="robot:link_2" pos="0 0 0.12402" quat="0.707105 -0.707108 0 0">
        <inertial pos="0.00075533 -0.10124 -0.002117" quat="0.500236 0.499898
    -0.49976 0.500105" mass="3.9086" diaginertia="0.040576 0.038125 0.00472638" />
        <joint name="robot:joint_2" pos="0 0 0" axis="0 0 1" limited="true" range=
    "-1.91986217719376 1.91986217719376"/>
        <geom mesh="robot:link_2" material="robot:arm_mat" name="robot:link_2" />

        <!-- Link_3 -->
        <body name="robot:link_3" pos="0 -0.27 0" quat="0.707105 0 0 -0.707108">
          <inertial pos="0.05791 0.022808 0.0010644" quat="0.709877 0.661444
    -0.229593 -0.0765141" mass="2.9437" diaginertia="0.012423 0.0121935 0.00412511"
     />
          <joint name="robot:joint_3" pos="0 0 0" axis="0 0 1" limited="true"
    range="-1.5707963267949 1.22173047639603"/>
          <geom mesh="robot:link_3" material="robot:arm_mat" name="robot:link_3"
    />

          <!-- Link_4 -->
          <body name="robot:link_4" pos="0.069509 0.1496 0" quat="0.707105
    -0.707108 0 0">
            <inertial pos="0.00037206 0.0045682 0.076575" quat="0.706635
    0.00447823 -0.00607205 0.707538" mass="1.4476" diaginertia="0.0042236
    0.00364171 0.00109819" />
            <joint name="robot:joint_4" pos="0 0 0" axis="0 0 1" limited="true"
    range="-2.79252680319093 2.79252680319093"/>
            <geom mesh="robot:link_4" material="robot:arm_mat" name="robot:link_4"
     />

```

```
            <!-- Link_5 -->
            <body name="robot:link_5" pos="0 0 0.1524" quat="0.707105 0.707108 0 0
    ">
              <inertial pos="0.00053453 -0.0010786 3.6881e-05" quat="0.51132
    0.486515 0.513145 0.488404" mass="0.54663" diaginertia="0.000404673 0.000367389
     0.000290008" />
              <joint name="robot:joint_5" pos="0 0 0" axis="0 0 1" limited="true"
    range="-2.0943951023932 2.0943951023932"/>
              <geom mesh="robot:link_5" material="robot:arm_mat" name="robot:
    link_5" />

              <!-- Link_6 -->
              <body name="robot:link_6" pos="0 0.066016 0" quat="0.707105
    -0.707108 0 0">
                <inertial pos="0.00016958 1.3237e-06 -0.0010621" quat="0.49847
    0.498538 -0.501491 0.501492" mass="0.013678" diaginertia="2.97685e-06 1.69425e
    -06 1.6577e-06" />
                <joint name="robot:joint_6" pos="0 0 0" axis="0 0 1" limited="true
    " range="-6.98131700797732 6.98131700797732" type="hinge"/>
                <geom mesh="robot:link_6" material="robot:arm_mat" name="robot:
    link_6" rgba="0.2 0.2 0.2 1"/>

                <!--TCP Location-->
                <site name="TCP:center" pos="0 0 0" rgba="1 0 0 1" size="0.005
    0.005 0.005"></site>
              </body>
            </body>
          </body>
        </body>
      </body>
        </body>
  </body>
</mujoco>
```

B.2.3. irb120 reach.xml

```
<?xml version="1.0" encoding="utf-8"?>
<mujoco>
  <compiler angle="radian" coordinate="local" meshdir="../stls/irb120" texturedir=
    "../textures"></compiler>
  <option timestep="0.001" gravity="0 0 -9.8">
    <flag warmstart="enable"></flag>
  </option>

  <!-- Loading of common properties -->
  <include file="../lib_shared.xml"></include>
  <include file="env_shared.xml"></include>

  <!-- Here we define all the environment elements-->
  <worldbody>
    <!-- The robot of this  environment -->
    <include file="robot.xml"></include>

    <!-- Plane that works as floor of the simulation -->
    <geom name="floor_0" pos="0.5 0 -0.1" size="2.125 1.75 1" type="plane"
    material="floor_mat"></geom>
    <body name="floor_0" pos="0.5 0 -0.1"></body>

    <!-- Working table -->
```

```
    <body pos="0.5 0 -0.05" name="table_0">
      <geom size="0.5 0.5 0.05" type="box" mass="2000" material="table_mat"></geom>
    </body>

    <!-- Target definition -->
    <body name="target" pos="0 0 0">
      <joint name="target:joint" type="free" damping="0"></joint>
            <geom name="target:geom" type="sphere" size="0.005 0.005 0.005" rgba="1 0 0 1"></geom>
            <site name="target:center" pos="0 0 0" size="0.005 0.005 0.005" rgba="1 0 0 1"></site>
        </body>

    <!-- Basic lightning -->
    <light directional="true" ambient="0.2 0.2 0.2" diffuse="0.8 0.8 0.8" specular="0.3 0.3 0.3" castshadow="false" pos="0 0 4" dir="0 0 -1" name="light0"></light>
  </worldbody>

  <actuator></actuator>
</mujoco>
```

B.2.4. irb120 reach-vector.xml

```
<?xml version="1.0" encoding="utf-8"?>
<mujoco>
  <compiler angle="radian" coordinate="local" meshdir="../stls/irb120" texturedir="../textures"></compiler>
  <option timestep="0.001" gravity="0 0 -9.8">
    <flag warmstart="enable"></flag>
  </option>

  <!-- Loading of common properties -->
  <include file="../lib_shared.xml"></include>
  <include file="env_shared.xml"></include>

  <!-- Here we define all the environment elements-->
  <worldbody>
    <!-- The robot of this  environment -->
    <include file="robot.xml"></include>

    <!-- Plane that works as floor of the simulation -->
    <geom name="floor_0" pos="0.5 0 -0.1" size="2.125 1.75 1" type="plane" material="floor_mat"></geom>
    <body name="floor_0" pos="0.5 0 -0.1"></body>

    <!-- Working table -->
    <body pos="0.5 0 -0.05" name="table_0">
      <geom size="0.5 0.5 0.05" type="box" mass="2000" material="table_mat"></geom>
    </body>

    <!-- Target definition -->
    <body name="target" pos="0 0 0">
      <joint name="target:joint" type="free" damping="0.01"></joint>
      <geom name="target:geom" type="mesh" mesh="arrow" rgba="1 0 0 1"></geom>
      <site name="target:center" pos="0 0 0" size="0.005 0.005 0.005" rgba="1 0 0 1"></site>
```

```
        </body>

    <!-- Basic lightning -->
    <light directional="true" ambient="0.2 0.2 0.2" diffuse="0.8 0.8 0.8" specular
    ="0.3 0.3 0.3" castshadow="false" pos="0 0 4" dir="0 0 -1" name="light0"></
    light>
  </worldbody>

  <actuator></actuator>
</mujoco>
```

B.3. irb14000

B.3.1. irb14000 env-shared.xml

```
<mujoco>

    <!-- Mesh loading for each robot link -->
    <asset>
        <mesh file="link_0.stl" name="robot:link_0"></mesh>
        <mesh file="link_1.stl" name="robot:link_1"></mesh>
        <mesh file="link_2.stl" name="robot:link_2"></mesh>
        <mesh file="link_3.stl" name="robot:link_3"></mesh>
        <mesh file="link_4.stl" name="robot:link_4"></mesh>
        <mesh file="link_5.stl" name="robot:link_5"></mesh>
        <mesh file="link_6.stl" name="robot:link_6"></mesh>
        <mesh file="link_7.stl" name="robot:link_7"></mesh>

        <!-- Arrow target for complex reach environments -->
        <mesh file="arrow.stl" name="arrow"></mesh>

        <!-- Gripper should be added for yumi pick and place environments -->
    </asset>

    <default>
        <default class="robot">
            <geom margin="0.001" material="robot:geomMat" rgba="0.45 0.45 0.45 1"
    solimp="0.99 0.99 0.01" solref="0.01 1" type="mesh"></geom>
            <joint armature="0.01" damping="100" frictionloss="0"></joint>
            <general ctrllimited="true"></general>
            <position kp="60"></position>
        </default>
          <!--  <default class="robot:yumiGripper">
                <geom condim="4" margin="0.001" type="box" user="0" rgba="0.356
    0.361 0.376 1.0"></geom>
                <joint armature="100" damping="1000" limited="true" solimplimit
    ="0.99 0.999 0.01" solreflimit="0.01 1" type="slide"></joint>
            </default>
          -->
    </default>

    <!-- We should exclude the contact between consecutive links to avoid unwanted
     collisions -->
    <contact>
        <exclude body1="robot:link_0" body2="robot:link_1_R"></exclude>
        <exclude body1="robot:link_1_R" body2="robot:link_2_R"></exclude>
        <exclude body1="robot:link_2_R" body2="robot:link_3_R"></exclude>
        <exclude body1="robot:link_3_R" body2="robot:link_4_R"></exclude>
```

```
        <exclude body1="robot:link_4_R" body2="robot:link_5_R"></exclude>
        <exclude body1="robot:link_5_R" body2="robot:link_6_R"></exclude>
        <exclude body1="robot:link_6_R" body2="robot:link_7_R"></exclude>

        <exclude body1="robot:link_0" body2="robot:link_1_L"></exclude>
        <exclude body1="robot:link_1_L" body2="robot:link_2_L"></exclude>
        <exclude body1="robot:link_2_L" body2="robot:link_3_L"></exclude>
        <exclude body1="robot:link_3_L" body2="robot:link_4_L"></exclude>
        <exclude body1="robot:link_4_L" body2="robot:link_5_L"></exclude>
        <exclude body1="robot:link_5_L" body2="robot:link_6_L"></exclude>
        <exclude body1="robot:link_6_L" body2="robot:link_7_L"></exclude>
    </contact>

    <!-- Position actuator definition -->
    <actuator>
        <position name="robot:A_joint_1_R" class="robot" joint="robot:joint_1_R"
    ctrlrange="-2.9409 2.9409"></position>
        <position name="robot:A_joint_2_R" class="robot" joint="robot:joint_2_R"
    ctrlrange="-2.5045 0.75922"></position>
        <position name="robot:A_joint_7_R" class="robot" joint="robot:joint_7_R"
    ctrlrange="-2.9409 2.9409"></position>
        <position name="robot:A_joint_3_R" class="robot" joint="robot:joint_3_R"
    ctrlrange="-2.1555 1.3963"></position>
        <position name="robot:A_joint_4_R" class="robot" joint="robot:joint_4_R"
    ctrlrange="-5.0615 5.0615"></position>
        <position name="robot:A_joint_5_R" class="robot" joint="robot:joint_5_R"
    ctrlrange="-1.5359 2.4086"></position>
        <position name="robot:A_joint_6_R" class="robot" joint="robot:joint_6_R"
    ctrlrange="-3.9968 3.9968"></position>
    </actuator>

    <!--
    <actuator>
        <position name="robot:A_joint_1_L" class="robot:yumi" joint="robot:
    joint_1_L" ctrlrange="-2.9409 2.9409"></position>
        <position name="robot:A_joint_2_L" class="robot:yumi" joint="robot:
    joint_2_L" ctrlrange="-2.5045 0.75922"></position>
        <position name="robot:A_joint_7_L" class="robot:yumi" joint="robot:
    joint_7_L" ctrlrange="-2.9409 2.9409"></position>
        <position name="robot:A_joint_3_L" class="robot:yumi" joint="robot:
    joint_3_L" ctrlrange="-2.1555 1.3963"></position>
        <position name="robot:A_joint_4_L" class="robot:yumi" joint="robot:
    joint_4_L" ctrlrange="-5.0615 5.0615"></position>
        <position name="robot:A_joint_5_L" class="robot:yumi" joint="robot:
    joint_5_L" ctrlrange="-1.5359 2.4086"></position>
        <position name="robot:A_joint_6_L" class="robot:yumi" joint="robot:
    joint_6_L" ctrlrange="-3.9968 3.9968"></position>
    </actuator>
    -->
</mujoco>
```

B.3.2. irb14000 robot.xml

```
<mujoco model="irb14000">
  <!-- Irb14000 Body -->
  <body childclass="robot" name="robot:link_0" pos="0 0 0">
    <inertial pos="-0.180659 -8.14964e-05 0.165304" quat="0.70652 0.00515169
    0.00565215 0.707652" mass="45.2531" diaginertia="1.89629 1.69089 0.835147" />
    <geom mesh="robot:link_0" name="robot:link_0" material="robot:base_mat" />
```

```
        <!-- Robot base location site -->
        <site name="robot:link_0" pos="0 0 0" rgba="0 0 0 0" size="0.01 0.01 0.01" ></site>

    <!-- IRB14000 RIGHT ARM -->
            <!-- Link_1 Right Arm -->
            <body name="robot:link_1_R" pos="0.0511065 -0.0714789 0.413506" quat="0.828827 0.314122 0.407926 0.219017">
                <inertial pos="-0.0102891 0.022587 0.0542902" quat="0.65678 -0.253182 0.058172 0.707923" mass="0.962897" diaginertia="0.00344744 0.00324539 0.0010754" />
                <joint name="robot:joint_1_R" pos="0 0 0" axis="0 0 1" limited="true" range="-2.9409 2.9409"/>
                <geom mesh="robot:link_1" material="robot:arm_mat" name="robot:link_1_R" />

                <!-- Link_2 Right Arm -->
                <body name="robot:link_2_R" pos="-0.03 0.012 0.1032" quat="2.58206e-12 2.58206e-12 0.707107 0.707107">
                    <inertial pos="-0.0178662 0.0642913 -0.0266056" quat="0.620049 0.737651 0.256696 -0.0742745" mass="1.41054" diaginertia="0.00731934 0.00724274 0.00177357" />
                    <joint name="robot:joint_2_R" pos="0 0 0" axis="0 0 1" limited="true" range="-2.5045 0.75922"/>
                    <geom mesh="robot:link_2" material="robot:arm_mat" name="robot:link_2_R" />

                <!-- Link_3 Right Arm -->
                    <body name="robot:link_3_R" pos="-0.03 0.17283 -0.012" quat="0 0 -0.707107 -0.707107">
                        <inertial pos="0.0201722 0.0266762 0.0495184" quat="0.682438 0.0205618 0.31629 0.658647" mass="0.71874" diaginertia="0.00203503 0.00187486 0.000672467" />
                        <joint name="robot:joint_7_R" pos="0 0 0" axis="0 0 1" limited="true" range="-2.9409 2.9409"/>
                        <geom mesh="robot:link_3" material="robot:arm_mat" name="robot:link_3_R" />

                        <!-- Link_4 Right Arm -->
                        <body name="robot:link_4_R" pos="0.0405 0.011 0.0786701" quat="-0.5 0.5 0.5 0.5">
                            <inertial pos="0.022675 0.05591 -0.026683" quat="0.617031 0.759478 -0.204139 -0.0281782" mass="1.1808" diaginertia="0.00549316 0.00547297 0.00124577" />
                            <joint name="robot:joint_3_R" pos="0 0 0" axis="0 0 1" limited="true" range="-2.1555 1.3963"/>
                            <geom mesh="robot:link_4" material="robot:arm_mat" name="robot:link_4_R" />

                            <!-- Link_5 Right Arm -->
                            <body name="robot:link_5_R" pos="0.0405 0.164608 -0.011" quat="1.49666e-12 1.49666e-12 -0.707107 -0.707107">
                                <inertial pos="0.0074084 0.027822 0.045809" quat="0.818505 -0.130653 0.223406 0.512903" mass="0.34346" diaginertia="0.000860272 0.000815447 0.000199811" />
                                <joint name="robot:joint_4_R" pos="0 0 0" axis="0 0 1" limited="true" range="-5.0615 5.0615"/>
                                <geom mesh="robot:link_5" material="robot:arm_mat" name="robot:link_5_R" />
```

```
39
40                              <!-- Link_6 Right Arm -->
41                              <body name="robot:link_6_R" pos="0.027 0.027
    0.100392" quat="1.49459e-11 1.49459e-11 0.707107 0.707107">
42                                  <inertial pos="0.010694 -0.011809 -0.039384"
    quat="0.84965 0.0495877 0.344434 -0.396234" mass="0.49968" diaginertia="
    0.000601179 0.000581617 0.000493544" />
43                                  <joint name="robot:joint_5_R" pos="0 0 0" axis
    ="0 0 1" limited="true" range="-1.5359 2.4086"/>
44                                  <geom mesh="robot:link_6" material="robot:
    arm_mat" name="robot:link_6_R" />
45
46                                  <!-- Link_7 Right Arm -->
47                                  <body name="robot:link_7_R" pos="0.027 0.036
    -0.027" quat="5.60496e-12 5.60496e-12 -0.707107 -0.707107">
48                                      <inertial pos="-1.04461e-05 0.000177102
    -0.0105294" quat="0.484298 0.515482 -0.508498 0.491083" mass="0.0309422"
    diaginertia="8.71943e-06 7.44537e-06 7.32948e-06" />
49                                      <joint name="robot:joint_6_R" pos="0 0 0"
    axis="0 0 1" limited="true" range="-3.9968 3.9968"/>
50                                      <geom mesh="robot:link_7" material="robot:
    arm_mat" name="robot:link_7_R" />
51
52                                      <site name="TCP:center" pos="0 0 0" rgba="
    1 0 0 0" size="0.01 0.01 0.01"></site>
53                                   </body>
54                              </body>
55                          </body>
56                      </body>
57                  </body>
58              </body>
59          </body>
60
61      <!-- IRB14000 LEFT ARM -->
62          <!-- Link_1 Left Arm -->
63          <body name="robot:link_1_L" pos="0.0511065 0.0714789 0.413506" quat="
    0.828827 -0.314122 0.407926 -0.219017">
64              <inertial pos="-0.0102891 0.022587 0.0542902" quat="0.656776
    -0.25318 0.0581739 0.707927" mass="0.962896" diaginertia="0.00344742 0.00324538
     0.00107539" />
65              <joint name="robot:joint_1_L" pos="0 0 0" axis="0 0 1" limited="
    true" range="-2.9409 2.9409" />
66              <geom mesh="robot:link_1" material="robot:arm_mat" name="robot:
    link_1_L" />
67
68              <!-- Link_2 Left Arm -->
69              <body name="robot:link_2_L" pos="-0.03 0.012 0.1032" quat="2.58206
    e-12 2.58206e-12 0.707107 0.707107">
70                  <inertial pos="-0.0178607 0.0642921 -0.0266029" quat="0.620055
     0.737698 0.256566 -0.074206" mass="1.41058" diaginertia="0.00731957 0.00724288
     0.0017737" />
71                  <joint name="robot:joint_2_L" pos="0 0 0" axis="0 0 1" limited
    ="true" range="-2.5045 0.75922" />
72                  <geom mesh="robot:link_2" material="robot:arm_mat" name="robot
    :link_2_L" />
73
74                  <!-- Link_3 Left Arm -->
75                  <body name="robot:link_3_L" pos="-0.03 0.17283 -0.012" quat="0
     0 0.707107 0.707107">
```

```
                        <inertial pos="0.020397 0.0269733 0.0505184" quat="
    0.687324 0.0205637 0.32457 0.649475" mass="0.710818" diaginertia="0.00196021
    0.00180398 0.000662852" />
                        <joint name="robot:joint_7_L" pos="0 0 0" axis="0 0 1"
    limited="true" range="-2.9409 2.9409" />
                        <geom mesh="robot:link_3" material="robot:arm_mat" name="
    robot:link_3_L" />

                        <!-- Link_4 Left Arm -->
                        <body name="robot:link_4_L" pos="0.0405 0.011 0.0786701"
    quat="-0.5 0.5 0.5 0.5">
                            <inertial pos="0.022675 0.05591 -0.026684" quat="
    0.617032 0.759463 -0.204198 -0.0281185" mass="1.1808" diaginertia="0.0054931
    0.00547297 0.00124582" />
                            <joint name="robot:joint_3_L" pos="0 0 0" axis="0 0 1"
     limited="true" range="-2.1555 1.3963" />
                            <geom mesh="robot:link_4" material="robot:arm_mat"
    name="robot:link_4_L" />

                            <!-- Link_5 Left Arm -->
                            <body name="robot:link_5_L" pos="0.0405 0.164608
    -0.011" quat="1.11842e-12 1.11842e-12 -0.707107 -0.707107">
                                <inertial pos="0.00740864 0.027823 0.0458101" quat
    ="0.8185 -0.130652 0.223413 0.512907" mass="0.343454" diaginertia="0.000860239
    0.000815408 0.00019979" />
                                <joint name="robot:joint_4_L" pos="0 0 0" axis="0
    0 1" limited="true" range="-5.0615 5.0615" />
                                <geom mesh="robot:link_5" material="robot:arm_mat"
     name="robot:link_5_L" />

                                <!-- Link_6 Left Arm -->
                                <body name="robot:link_6_L" pos="0.027 0.027
    0.100392" quat="1.49459e-11 1.49459e-11 0.707107 0.707107">
                                    <inertial pos="0.010694 -0.011809 -0.039384"
    quat="0.84965 0.0495877 0.344434 -0.396234" mass="0.49968" diaginertia="
    0.000601179 0.000581617 0.000493544" />
                                    <joint name="robot:joint_5_L" pos="0 0 0" axis
    ="0 0 1" limited="true" range="-1.5359 2.4086" />
                                    <geom mesh="robot:link_6" material="robot:
    arm_mat" name="robot:link_6_L" />

                                    <!-- Link_7 Left Arm -->
                                    <body name="robot:link_7_L" pos="0.027 0.036
    -0.027" quat="5.60496e-12 5.60496e-12 -0.707107 -0.707107">
                                        <inertial pos="-1.04461e-05 0.000177102
    -0.0105294" quat="0.484298 0.515482 -0.508498 0.491083" mass="0.0309422"
    diaginertia="8.71943e-06 7.44537e-06 7.32948e-06" />
                                        <joint name="robot:joint_6_L" pos="0 0 0"
    axis="0 0 1" limited="true" range="-3.9968 3.9968" />
                                        <geom mesh="robot:link_7" material="robot:
    arm_mat" name="robot:link_7_L" />
                                        <site name="robot:grip_L" pos="0 0 0" rgba
    ="0 0 0 0" size="0.01 0.01 0.01"></site>
                                    </body>
                                </body>
                            </body>
                        </body>
                    </body>
                </body>
```

```
            </body>

    <body name="robot:external_camera_body_0" pos="0 0 0">
      <camera euler="0 0.75 1.57" fovy="43.3" name="external_camera_0" pos="1.3 0 1.2"></camera>
    </body>
  </body>
</mujoco>
```

B.3.3. irb14000 reach.xml

```
<?xml version="1.0" encoding="utf-8"?>
<mujoco>
  <compiler angle="radian" coordinate="local" meshdir="../stls/irb14000" texturedir="../textures"></compiler>
  <option timestep="0.001" gravity="0 0 -9.8">
    <flag warmstart="enable"></flag>
  </option>

  <!-- Loading of common properties -->
  <include file="../lib_shared.xml"></include>
  <include file="env_shared.xml"></include>

  <!-- Here we define all the environment elements-->
  <worldbody>
    <!-- The robot of this  environment -->
    <include file="robot.xml"></include>

    <!-- Plane that works as floor of the simulation -->
    <geom name="floor_0" pos="0.5 0 -0.1" size="2.125 1.75 1" type="plane" material="floor_mat"></geom>
    <body name="floor_0" pos="0.5 0 -0.1"></body>

    <!-- Working table -->
    <body pos="0.5 0 -0.05" name="table_0">
      <geom size="0.5 0.5 0.05" type="box" mass="2000" material="table_mat"></geom>
    </body>

    <!-- Target definition -->
    <body name="target" pos="0 0 0">
      <joint name="target:joint" type="free" damping="0"></joint>
      <geom name="target:geom" type="sphere" size="0.005 0.005 0.005" rgba="1 0 0 1"></geom>
      <site name="target:center" pos="0 0 0" size="0.005 0.005 0.005" rgba="1 0 0 1"></site>
        </body>

    <!-- Basic lightning -->
    <light directional="true" ambient="0.2 0.2 0.2" diffuse="0.8 0.8 0.8" specular="0.3 0.3 0.3" castshadow="false" pos="0 0 4" dir="0 0 -1" name="light0"></light>
  </worldbody>

  <actuator></actuator>
</mujoco>
```

B.3.4. irb14000 reach-vector.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<mujoco>
  <compiler angle="radian" coordinate="local" meshdir="../stls/irb14000"
    texturedir="../textures"></compiler>
  <option timestep="0.001" gravity="0 0 -9.8">
    <flag warmstart="enable"></flag>
  </option>

  <!-- Loading of common properties -->
  <include file="../lib_shared.xml"></include>
  <include file="env_shared.xml"></include>

  <!-- Here we define all the environment elements-->
  <worldbody>
    <!-- The robot of this  environment -->
    <include file="robot.xml"></include>

    <!-- Plane that works as floor of the simulation -->
    <geom name="floor_0" pos="0.5 0 -0.1" size="2.125 1.75 1" type="plane"
    material="floor_mat"></geom>
    <body name="floor_0" pos="0.5 0 -0.1"></body>

    <!-- Working table -->
    <body pos="0.5 0 -0.05" name="table_0">
      <geom size="0.5 0.5 0.05" type="box" mass="2000" material="table_mat"></geom
    >
    </body>

    <!-- Target definition -->
    <body name="target" pos="0 0 0">
      <joint name="target:joint" type="free" damping="0.01"></joint>
      <geom name="target:geom" type="mesh" mesh="arrow" rgba="1 0 0 1"></geom>
      <site name="target:center" pos="0 0 0" size="0.005 0.005 0.005" rgba="1 0 0
    1"></site>
        </body>

    <!-- Basic lightning -->
    <light directional="true" ambient="0.2 0.2 0.2" diffuse="0.8 0.8 0.8" specular
    ="0.3 0.3 0.3" castshadow="false" pos="0 0 4" dir="0 0 -1" name="light0"></
    light>
  </worldbody>

  <actuator></actuator>
</mujoco>
```

C

Anexo - Guía de instalación

En este anexo se explicarán los pasos a seguir para instalar la librería robot-gym y el simulador de físicas Mujoco

C.1. Instalar Mujoco 2.1

En esta guía se explicará como instalar Mujoco 2.1 en un sistema operativo Linux, concretamente Ubuntu 20.04 x64.

- Descargar los archivos pertinentes del repositorio de github Mujoco 2.1 - Linux - x64
- Crear la siguiente jerarquía de carpetas en el sistema operativo *$HOME/.mujoco/mujoco210/*
- Descomprimir el archivo .zip descargado en la nueva carpeta creada. Para ser exactos, la jerarquía final debe ser como la presentada en el esquema de la Figura C.1

```
$HOME/.mujoco/
└── mujoco210
    ├── bin
    ├── include
    ├── model
    ├── sample
    └── THIRD_PARTY_NOTICES
```

Figura C.1. Jerarquía de archivos de mujoco

- Por último, es necesario añadir el *PATH* para que el sistema pueda encontrar los archivos. Para ello volvemos al directorio *$HOME* y editamos el archivo *.bashrc*. Solamente debemos añadir el siguiente comando:

  export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/.mujoco/mujoco210/bin

C.2. Instalar robot-gym

El proceso de instalación de robot-gym se ha simplificado lo máximo posible.

- Descargar la carpeta de archivos de robot-gym
- La jerarquía de archivos debería ser la presente en la Figura C.2

```
$HOME/TFM/robot-gym
├── build
├── dist
├── LICENSE
├── README.md
├── robot_gym
├── robot_gym.egg-info
└── setup.py
```

Figura C.2. Jerarquía instalación robot-gym

- Accedemos al directorio y ejecutamos el siguiente comando:

  python setup.py install –user

Se recomienda encarecidamente el uso de Miniconda[20]

C.3. Instalar rl-baselines3-zoo

Esta librería no es necesaria pero sí altamente recomendable para poder usar los distintos algoritmos de aprendizaje y el *framework* de RL sobre los entornos de aprendizaje.

Para ello se recomienda seguir la guía presente en RL-Baselines3-zoo

D

Anexo - Guía de uso

A lo largo de esta guía se explicará que archivos debería editar el usuario en caso de querer modificar los entornos de aprendizaje y como utilizar la librería RL-Baselines3-zoo para entrenar agentes

D.1. Uso de robot-gym

En esta sección desgranaremos la información necesaria para utilizar la librería de forma adecuada, dónde almacenar los archivos *.stl* y *.xml* (mallas y archivos de configuración de mujoco), cómo configurar los entornos a través de los ficheros en la Sección A.7 y Sección A.6

D.1.1. Añadir robot a la librería robot-gym

A la hora de añadir un robot a la librería hay que seguir una seria de pasos claramente definidos, sin embargo se recomienda tener experiencia con la gestión de modelos 3D y dinámica robótica para no atascarse, sobre todo en los primeros pasos que pueden resultar poco intuitivos.

- Se descargan los archivos CAD desde la respectiva web o repositorio oficiales
- (OPCIONAL) Los archivos descargados, ya sean el *assembly* o los diferentes elementos del robot por separado, se han de abrir en *Autodeks Fusion 360* para:
    - Verificar la integridad de las mallas.
    - Verificar la disposición de los diferentes elementos.
    - Verificar las propiedades físicas designadas a cada componente.
- De *Autodesk Fusion 360* pasamos a *SOLIDWORKS* que es la herramienta principal para el procesado de los CAD.
    - Instalamos el complemento *SW2URDF*

- Añadimos ejes de coordenadas en aquellos puntos del robot que vayamos a usar como articulaciones. Lo lógico es seguir las normas propuestas por Denavit Hartenberg [Wik22d], pero no es imprescindible en este caso.

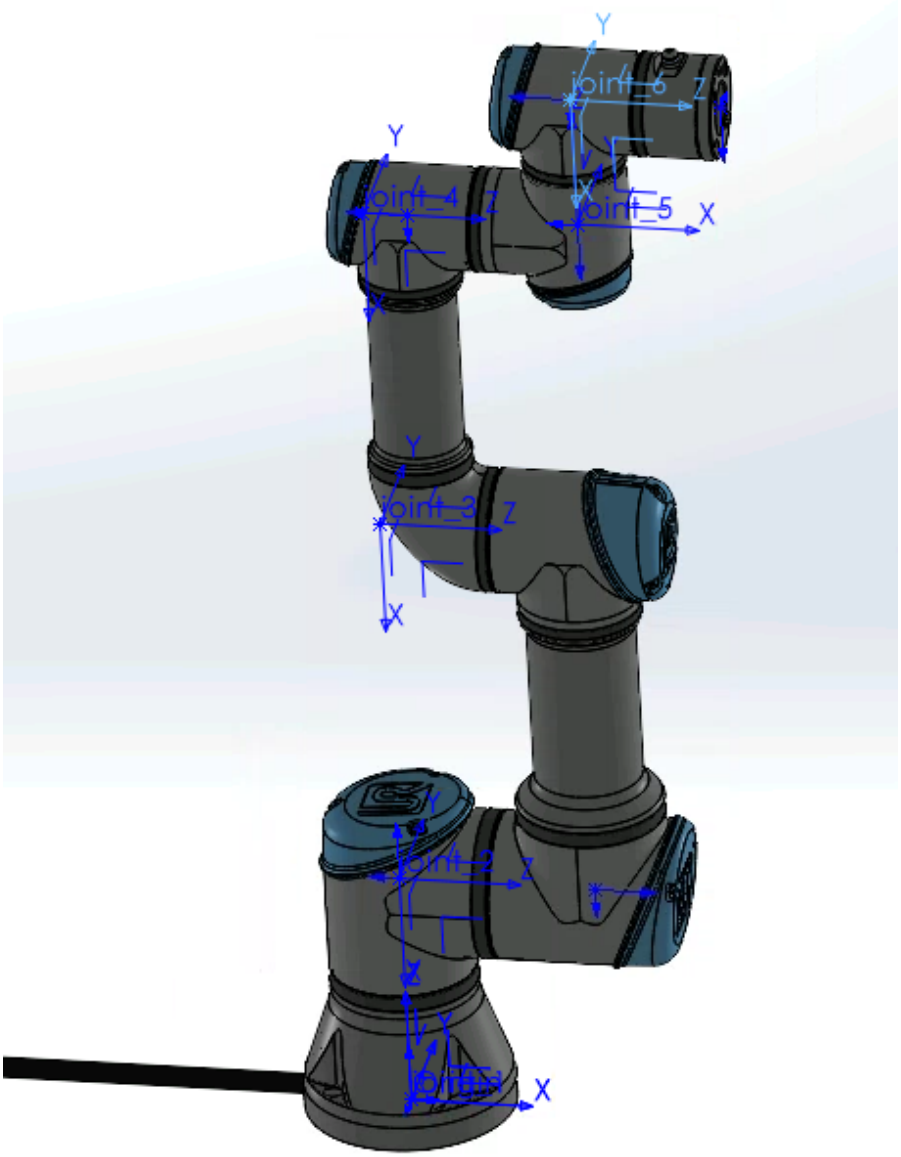


Figura D.1. Ejemplo de robot con los ejes y puntos relevantes añadidos

- Inicializamos el complemente y lo configuramos proporcionando toda la información que nos pida respecto a los ejes, los solidos a usar y sus relaciones.

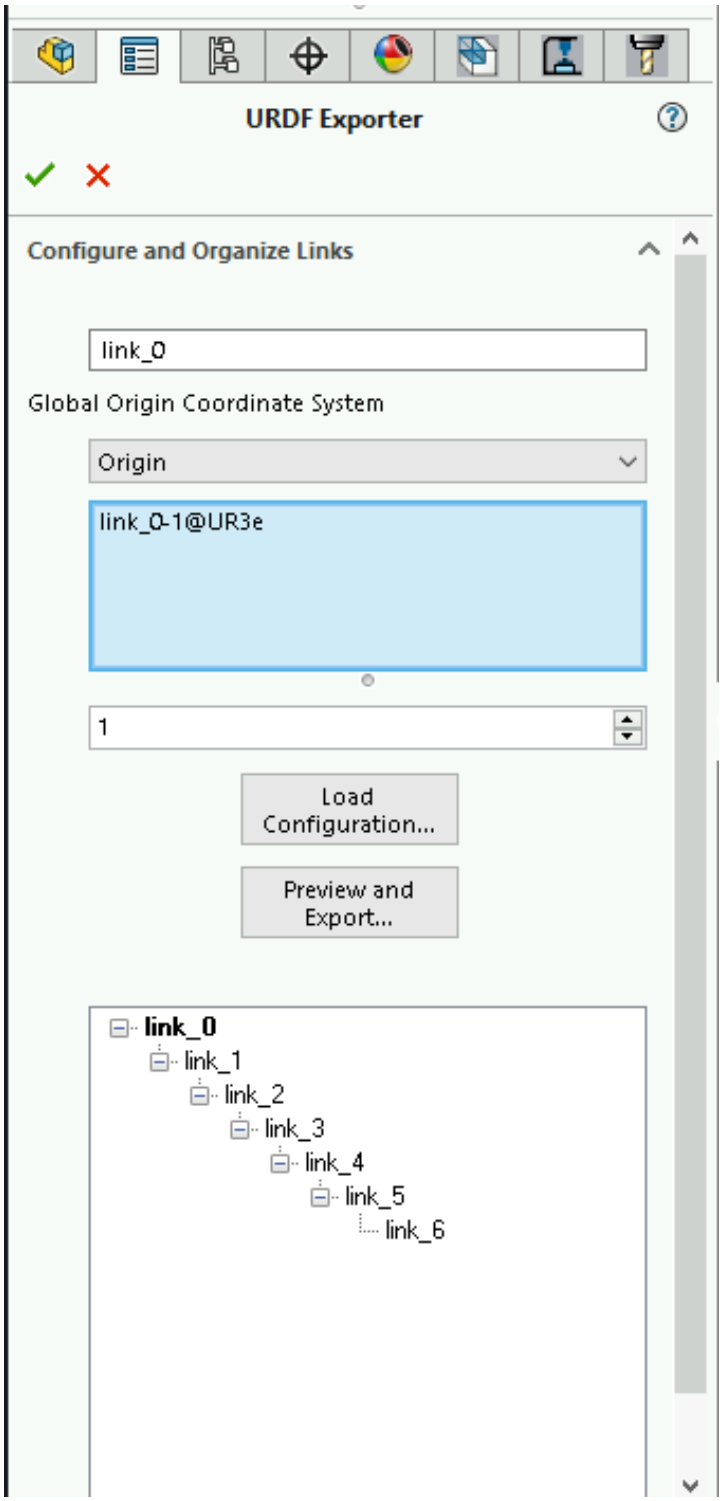


Figura D.2. Ejemplo de uso del complemento SW2URDF - 1

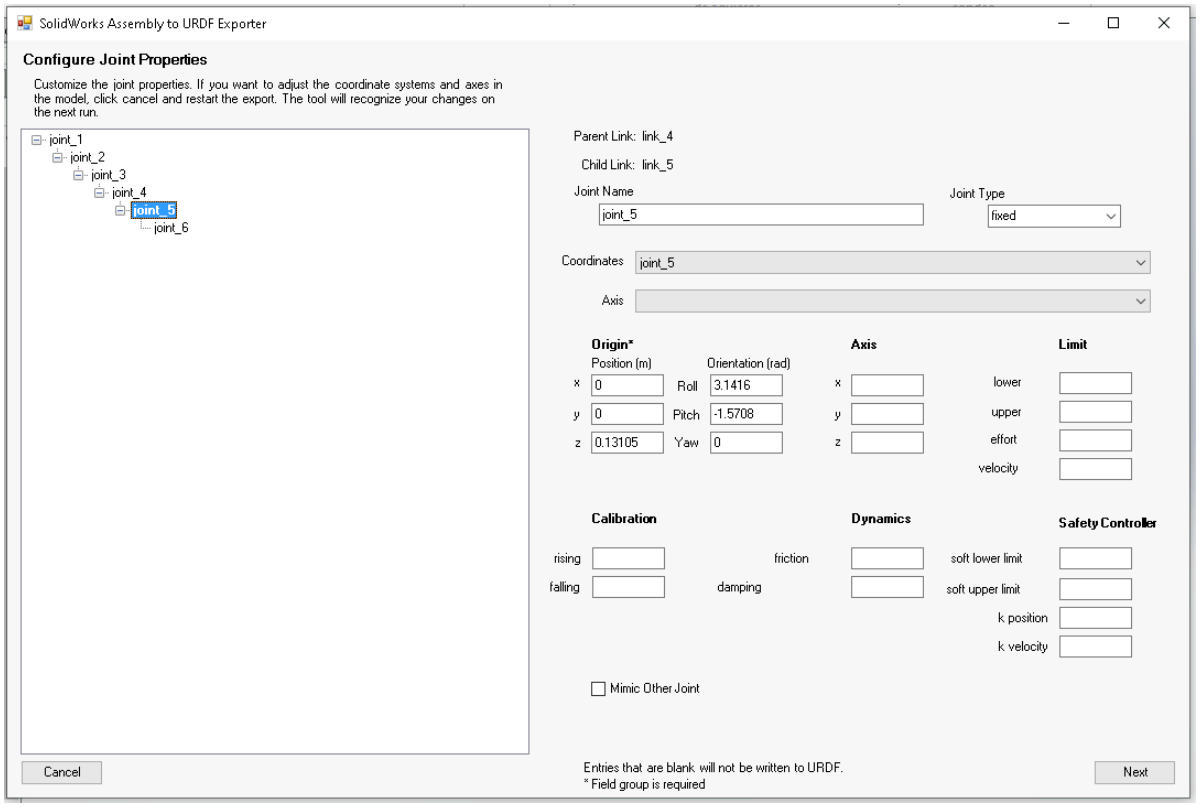


Figura D.3. Ejemplo de uso del complemento SW2URDF - 2

- Tras seguir los pasos del complemento de *SOLIDWORS* obtendremos varios archivos entre los que se encuentran los .stls y el archivo .urdf del robot. De todas estas carpetas, solo

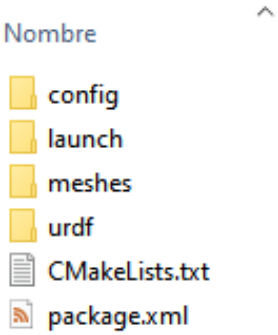


Figura D.4. Salida del complemento SW2URDF

  nos interesan la carpeta *meshes* y *urdf*

- A continuación en el terminal, nos dirijimos a la ubicación de la carpeta de archivos binarios según la estructura de archivos propuesta de mujoco con el siguiente comando:

  cd $HOME/.mujoco/mujoco210/bin

- Con estos archivos y Mujoco instalado en nuestro sistema, podemos exportar el formato a xml con ayuda de la función *compile* que contiene Mujoco.

  Basta con copiar todos los archivos *.stl* y el archivo *.urdf* correspondiente en el mismo directorio y ejecutar el comando:

***./compile** 'Ubicación archivo urdf de origen' 'Ubicación y nombre del archivo xml de salida'*

Un ejemplo sería: **./compile $HOME/UR3e.urdf $HOME/UR3e.xml**

- A partir de aquí todo lo necesario es introducir los archivos en la ubicación adecuada y mantener el formato presentado en la librería.
  - Los archivos *.stl* se ubican en el subdirectorio *.../robot-gym/robot-gym/envs/assets/stls* y se guardan en una carpeta con el nombre del robot. Es recompendable copiar el archivo *arrow.stl* presente en otros robots para poder visualizar la orientación del objetivo en los entornos de aprendizaje correspondientes.
  - En cuanto al archivo *.xml*, se debe guardar en la subcarpeta *.../robot-gym/robot-gym/envs/assets/* en una carpeta con el nombre del robot, igual que los archivos de malla. Se recomienda encarecidamente echar un vistazo a cómo se han estructurado los archivos a otros robots y replicar el formato. Esto se hará en el futuro de forma automática con un complemento.
  - A continuación, se debe volver a la subcarpeta *.../robot-gym/robot-gym/envs/* añadir otra carpeta con el nombre del robot y agregar los archivos de Python que definan el funcionamiento del entorno del robot heredando de la clase base presentada en la Sección A.1
  - Por último, queda incluir dichos archivos en los archivos __init__.py correspondientes, añadir la configuración del robot y registrarlo tal y como se explica en la Sección D.1.2

D.1.2. Configurar un robot en roboy-gym

Configurar y registar un robot en robot-gym es relativamente sencillo. Para registrarlo únicamente hace falta editar el fichero correspondiente a la Sección A.7 siguiendo el mismo formato que el presentado.

```
26 register(
27     id='Irb120_ReachPoint',
28     entry_point='gym_abb.envs:Irb120ReachEnv',
29     kwargs=kwargs_dicts['Irb120_ReachPoint'],
30     max_episode_steps=100,
31 )
```

Figura D.5. Registro de entornos

Como se puede ver en la Figura D.5, el parámetro 'id' corresponde al nombre que se quiera usar para llamar al entorno desde fuera de la librería, tal y como se ve en la Sección A.8. La configuración se registra con el parámetro 'kwargs' y se encuentran ampliamente explicados en la Sección A.6. Lógicamente los parámetros cambiaran en función de la definición del entorno de aprendizaje.

Esto te permite tener diferentes entornos en base a la configuración que se le da a los entornos ya implementados.

D.2. Uso de RL-Baselines3-zoo

Para conocer la librería en detalle y en caso de tener alguna duda respecto a su utilización, los desarrolladores de la misma han dejado una documentación semi-extensa: Documentación RL-Baselines3-zoo

Los comandos más relevantes son *enjoy.py* y *train.py*. Un ejemplo de uso que se ha empleado en el proyecto es:

python train.py –algo td3 –env Irb120ReachVector-v0 –tensorboard-log $HOME/TFM/tmp/stable-baselines/ -i /home/pablo/TFM/rl-baselines3-zoo/logs/td3/Irb120ReachVector-v0_3/rl_model_300000_steps.zip –eval-freq 1000 –eval-episodes 10 –n-eval-envs 2 –save-freq 100000 –save-replay-buffer

- *'algo'*: Nos permite seleccionar el algoritmo de aprendizaje entre aquéllos que estén implementados en la librería. (Es importante que los hiperparámetros estén definidos en la subcarpeta correspondiente)
- *'env'*: Nos permite especificar el 'id' del entorno a entrenar
- *'tensorboard-log'*: Nos permite elegir la ubicación donde se almacenará la información de supervisión del proceso de aprendizaje proporcionada por tensorboard. Para poder analizarla basta con acceder al directorio y ejectuar el comando a continuación:

  tensorboard –logdir .
- *'i'*: Nos permite retomar un agente ya entrenado para continuar su entrenamiento.
- *'eval-freq'*: Nos permite fijar cada cuantos episodios se evaluará un agente en entrenamiento con entornos independientes a la primera semilla de generación.
- *'eval-episodes'*: Número de episodios de evaluación por entorno.
- *'n-envs'*: Número de entornos de evaluación simultáneos.
- *'save-freq'*: Frecuencia de episodios con la que se guarda el entorno en entrenamiento.
- *'save-replay-buffer'*: Nos permite guardar el buffer de datos generado para *HER*

Bibliografía

[20] *Anaconda Software Distribution*, ver. Vers. 2-2.4.0, 2020. dirección: https://docs.anaconda.com/.

[Ach18] J. Achiam, «Spinning Up in Deep Reinforcement Learning,» 2018. dirección: https://spinningup.openai.com/.

[Ber17] F. Berzal. «Breve historia de la inteligencia artificial: el camino hacia la empresa.» (2017), dirección: https://www.cesce.es/es/w/asesores-de-pymes/breve-historia-la-inteligencia-artificial-camino-hacia-la-empresa.

[Cop13] B. Copeland, *Alan Turing: El pionero de la era de la información* (Noema Series). Turner, 2013. dirección: https://books.google.es/books?id=CSpZAgAAQBAJ.

[Gon22] J. L. Gonzalez. «Tipos de aprendizaje automático.» (2022), dirección: https://medium.com/soldai/tipos-de-aprendizaje-autom%C3%A1tico-6413e3c615e2.

[NR21] G. K. Nigora Gafur y M. Ruskowski. «Dynamic collision avoidance for multiple robotic manipulators based on a non-cooperative multi-agent game.» (2021), dirección: https://arxiv.org/pdf/2103.00583.pdf.

[Por] A. M. Porcelli, «La intelifencia artificial y la robótica: sus dilemas sociales, éticos y jurídicos,» en *Derecho global. Estudios sobre derecho y justicia*.

[Raf+21] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus y N. Dormann, «Stable-Baselines3: Reliable Reinforcement Learning Implementations,» *Journal of Machine Learning Research*, vol. 22, n.º 268, págs. 1-8, 2021. dirección: http://jmlr.org/papers/v22/20-1364.html.

[Raf20] A. Raffin, *RL Baselines3 Zoo*, https://github.com/DLR-RM/rl-baselines3-zoo, 2020.

[Riv19] O. Rivlin. «Reinforcement Learning for Real-World Robotics.» (2019), dirección: https://towardsdatascience.com/reinforcement-learning-for-real-world-robotics-148c81dbdcff.

[Rue22] J. F. V. Rueda. «Aprendizaje supervisado y no supervisado.» (2022), dirección: https://healthdataminer.com/data-mining/aprendizaje-supervisado-y-no-supervisado/.

[Sur20] A. Suran. «On-Policy v/s Off-Policy Learning.» (2020), dirección: https://towardsdatascience.com/on-policy-v-s-off-policy-learning-75089916bc2f#:~:text=Off-Policy%20learning%20algorithms%20evaluate,in%20both%20ways)%2C%20etc..

[TET12] E. Todorov, T. Erez e Y. Tassa, «MuJoCo: A physics engine for model-based control,» en *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2012, págs. 5026-5033.

[Tew20] U. Tewari. «Which Reinforcement learning-RL algorithm to use where, when and in what scenario?» (2020), dirección: https://medium.datadriveninvestor.com/which-reinforcement-learning-rl-algorithm-to-use-where-when-and-in-what-scenario-e3e7617fb0b1.

[Tri98] E. Trillas, *LA INTELIGENCIA ARTIFICIAL : MAQUINAS Y PERSONAS* (Temas de Debate Series). DEBATE, 1998. dirección: https://books.google.es/books?id=0igNAAAACAAJ.

[Wik22a] L. e. l. Wikipedia. «Aprendizaje por refuerzo.» (2022), dirección: https://es.wikipedia.org/wiki/Aprendizaje_por_refuerzo.

[Wik22b] L. e. l. Wikipedia. «Cinemática directa.» (2022), dirección: https://es.wikipedia.org/wiki/Cinem%C3%A1tica_inversa.

[Wik22c] L. e. l. Wikipedia. «Cinemática inversa.» (2022), dirección: https://es.wikipedia.org/wiki/Cinem%C3%A1tica_directa.

[Wik22d] L. e. l. Wikipedia. «Denavit–Hartenberg parameters.» (2022), dirección: https://en.wikipedia.org/wiki/Denavit%E2%80%93Hartenberg_parameters.