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

Author: Giménez Suárez, Pablo Santiago
Directors: López López, Álvaro Jesús and Tobías, Ignacio de Rodrigo

***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.**

***Index Terms*—Artificial Intelligence, Reinforcement Learning, Pick & Place, Gym, Baselines, Computer generated trajectories.**

I. Introduction

ARTIFICIAL Intelligence as a discipline began to develop in 1950 with Alan Turing's article "Computing Machinery and Intelligence", in which he developed a way of assessing whether a machine was capable of thinking and which was later called the "Turing Test". Alan Turing played a major role in the Second World War, providing an unprecedented technical advantage by allowing German communications to be deciphered by brute force calculation. He is rightly considered one of the fathers of automatic computing. However, it was not until six years later with John McCarty, Marvin Miskey and Claude Shannon, the fathers of modern artificial intelligence, that the term was coined at a conference in Darmouth, USA. [1]

One of the most important applications of artificial intelligence is robotics. It is in this speciality where AI can shine the most, providing these algorithms made up of ones and zeros with a physical body that allows them to interact with the environment, opening up a range of possibilities in industry. One of the most important references in this field is undoubtedly Isaac Asimov, the father of robotics, who proposed his three fundamental laws:

- "A robot shall neither harm a human being nor, by inaction, allow a human being to be harmed."
- "A robot shall comply with commands given by human beings, except for those that conflict with the first law."
- "A robot shall protect its own existence to the extent that this protection does not conflict with the first or the second law."

II. State of the Art

Nowadays, 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. [2]

A. Types of learning

There are three types of learning in the world of artificial intelligence: [3]

- 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. [4]

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.

B. Industrial robotics

Today, industrial robots are being widely used in almost all production lines. The two main virtues that separate them from typical industrial manipulators are that they are multifunctional and reprogrammable, allowing them to adapt to the process. Because of this, a number of characteristics have been identified to differentiate between the different types of robot:

- Degrees of freedom. This determines the level of complexity of the robot's movement.
- Working area. Some robots have a large range and others have a limited range and is generally determined by the size of the links and the number of DoF *(Degrees of freedom)*.
- Load to be supported. Both the volume and weight of the objects to be lifted have a direct impact on the size and robustness of the robot used.
- Programmability level. Not all robots have high processing capacity.

C. Types of industrial robot

The main types of robots in industry would therefore be:

- Cartesian robot.
- Scara Robot.
- Cyclic Robot.
- Six-axis robot/PUMA.
- Double arm robot.

.

Furthermore, in recent years, so-called collaborative robots have been gaining momentum, which are specifically designed to work in the same environment as people, with reduced speeds and lower torque in the motors. Some examples of this technology have been mentioned in the section I, such as the *Swifty, GoFa or the YuMi*.

III. 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.

IV. 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. [5].

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.

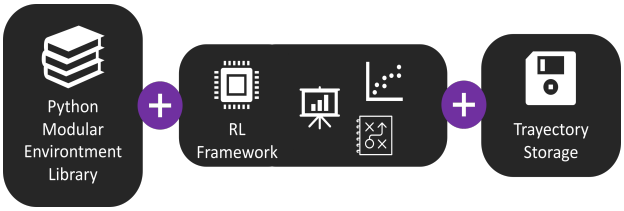

Fig. 1. Schematic of the system architecture.

V. Add robot to robot-gym library

When adding a robot to the library, a series of clearly defined steps must be followed, however, it is recommended to have experience with 3D model management and robotic dynamics to avoid getting stuck, especially in the first steps that may not be very intuitive.

- The CAD files are downloaded from the respective web or official repository.
- (OPTIONAL) The downloaded files, either the *assembly* or the different elements of the robot separately, have to be opened in *Autodeks Fusion 360* to:
  - Verify the integrity of the meshes.
  - Verify the layout of the different elements.
  - Verify the physical properties designated to each component.
- From *Autodesk Fusion 360* we move to *SOLIDWORKS* which is the main tool for CAD processing.
  - Install the *SW2URDF* add-in.
  - We add coordinate axes in those points of the robot that we are going to use as joints. The logical thing to do is to follow the rules proposed by Denavit Hartenberg [6], but it is not essential in this case.
  - We initialize the complement and configure it by providing all the information requested regarding the axes, the solids to be used and their relationships.
- After following the steps of the *SOLIDWORS* add-in we will obtain several files including the .stls and the .urdf file of the robot. Of all these folders, we are only interested in the *meshes* and *urdf* folders.
- Next in the terminal, we go to the location of the binary files folder according to the proposed mujoco file structure with the following command:
  cd $HOME/.mujoco/mujoco210/bin
- With these files and Mujoco installed in our system, we can export the format to xml with the help of the *compile* function contained in Mujoco.

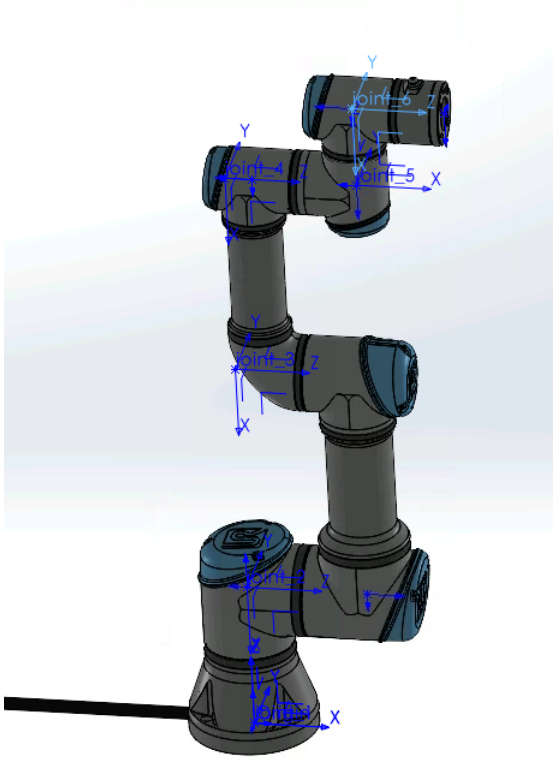

Fig. 2. Example of a robot with added axes and relevant points

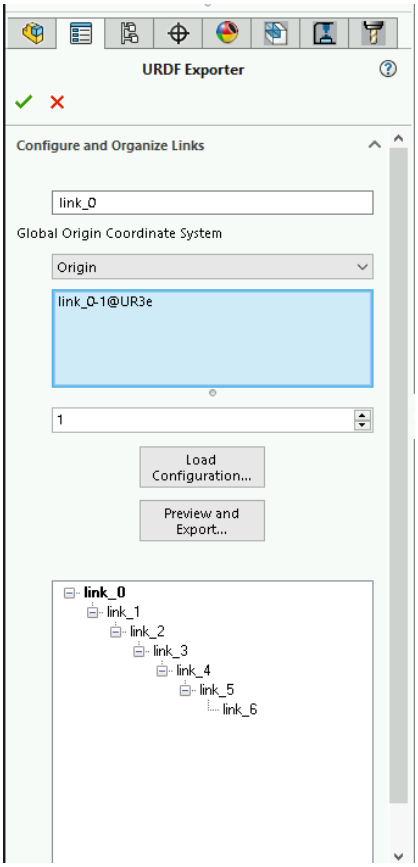

Fig. 3. Example of use of the SW2URDF add-on - 1

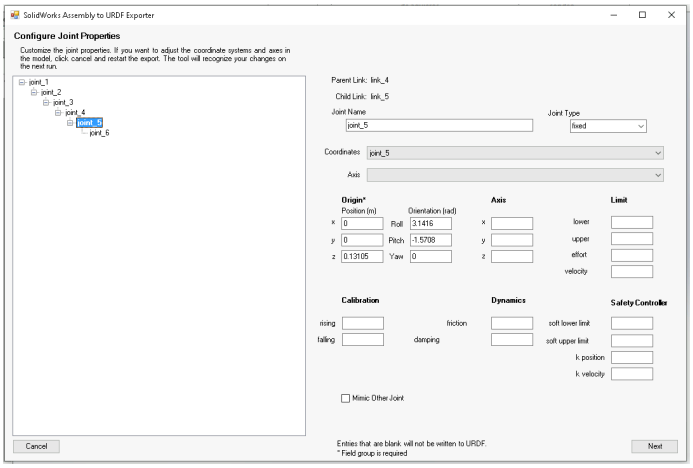

Fig. 4. Example of use of the SW2URDF add-on - 2

Simply copy all *.stl* files and the corresponding *.urdf* file into the same directory and execute the command:
***./compile** 'Source urdf file location'. 'Location and name of output xml file'*
An example would be: **./compile $HOME/UR3e.urdf $HOME/UR3e.xml**

- From here on all that is necessary is to insert the files in the appropriate location and maintain the format presented in the library.
  - The *.stl* files are located in the subdirectory *.../robot-gym/robot-gym/envs/assets/stls* and are saved in a folder with the name of the robot. It is recommended to copy the *arrow.stl* file present in other robots to be able to visualize the target orientation in the corresponding learning environments.
  - As for the *.xml* file, it should be saved in the subfolder *.../robot-gym/robot-gym/envs/assets/* in a folder with the name of the robot, just like the mesh files. It is strongly recommended to take a look at how the files have been structured to other robots and replicate the format. This will be done in the future automatically with an add-on.
  - Then, go back to the subfolder *.../robot-gym/robot-gym/envs/* add another folder with the name of the robot and add the Python files that define the operation of the robot's environment inheriting from the base class.
  - Finally, it remains to include these resources in the corresponding __init__.py files, add the configuration of the robot and register it.

VI. Operating dynamics

This section will explain the dynamics of the environments that make up the library when they are used.

All the implemented environments use the bases of reinforcement learning with the extended environment-agent cycle.

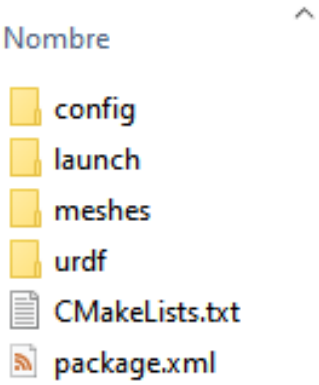

Fig. 5. Output of the SW2URDF add-on

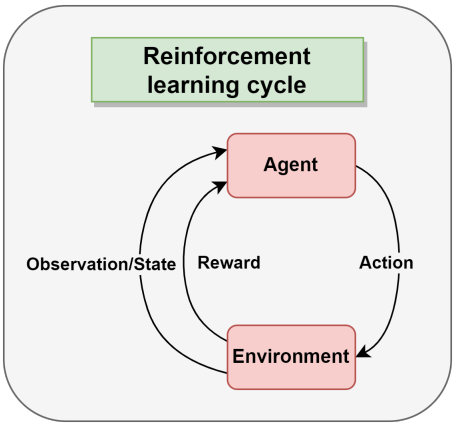

Fig. 6. Reinforcement learning cycleo

This cycle is repeated throughout the learning process in what are called *steps*. The Figure 7 illustrates this cycle and the internal processes performed by the environment in each of them.

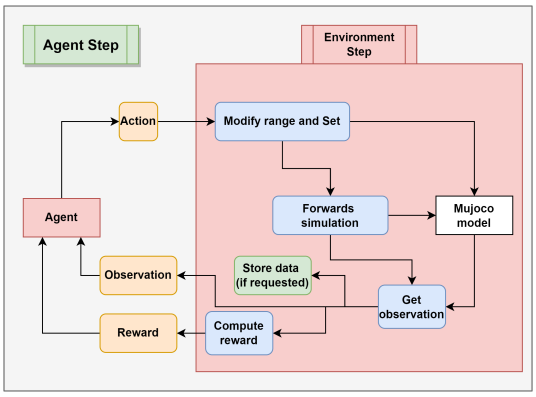

Fig. 7. Steps of the environment and the agent

These *steps* are limited within an episode. An episode comprises the number of *steps* between which the environment is initialized and the target is reached or a limit is reached. The steps limit is configured in the environment registry along with the Env-ID. The Figure 8 summarizes a training episode.

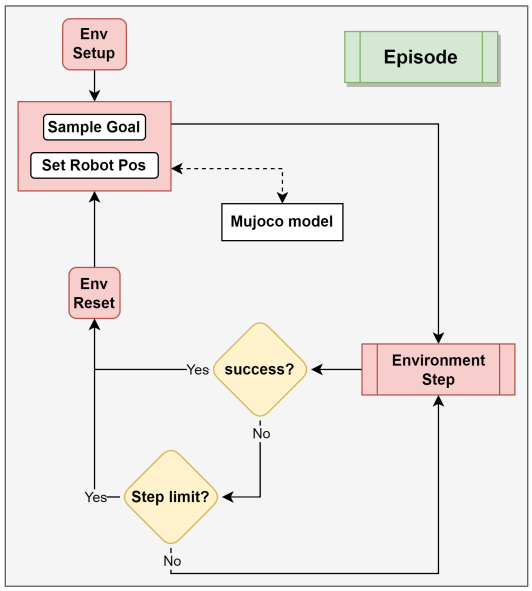

Fig. 8. Summary diagram of a training episode

VII. Results

To verify the functionality of the development environments a short code has been made. It simply loads the environment and executes random actions on it to verify that the resetting of the environment and the application of the actions is consistent with what is expected.

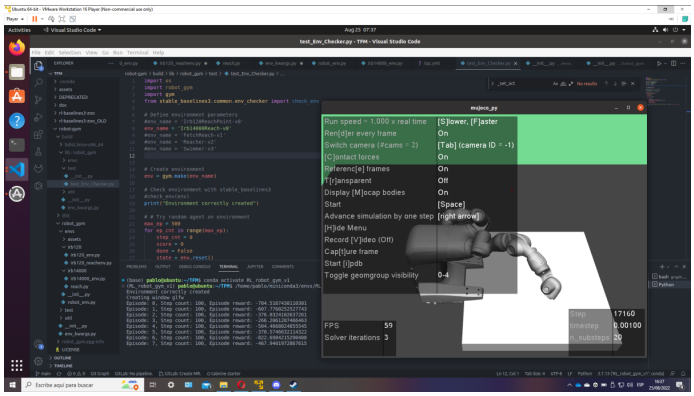

Fig. 9. Example of environment validation

Once this was done, different learning tests were run on the environments. In the following sections we will go into detail on three of them.

During the learning process the user can monitor the progress with the help of tensorboard and with the information presented in the command line:

As for the hyperparameters used during training, they vary depending on the algorithm used. In general, the ones proposed by *rl-baselines3-zoo* [5] in other environments and those obtained in the optimization of the same with the optuna tool

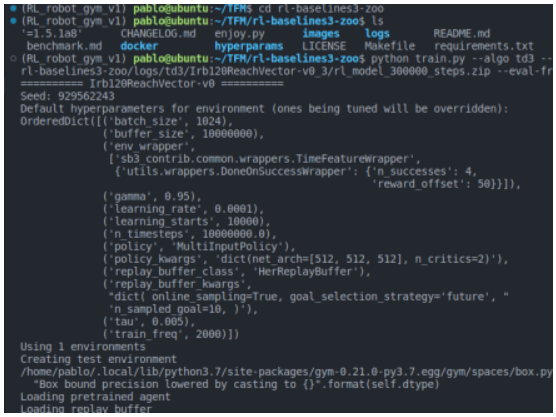

Fig. 10. Information presented to the user when initializing learning

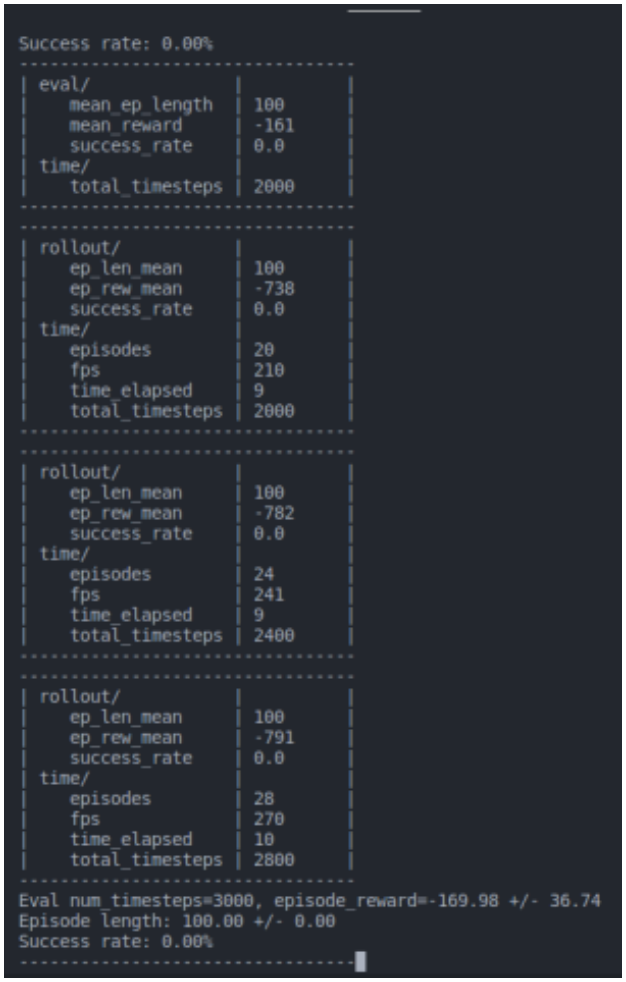

Fig. 11. Information presented to the user during training

implemented in this library. It is important to highlight the wrappers used, which include a wrapper by execution time limit of sb3-contrib and a wrapper called *'DoneOnSuccess'* that allows to finish the episode once the objective has been reached.

To validate the trained agents, they were executed with the help of another of the functions of *rl-baselines3-zoo* [5] which allows to run them with generation seeds other than training. Once it is verified that in these tests the robot is able to reach the objectives, generated with the same rules as in the training, the agent is considered to be validated.

Once the trained agents were validated, we proceeded to implement them in real robots. For the tests we used the Irb14000 present in the laboratory and the agent explained in the subsection VII-A. The results were almost optimal. Unfortunately, due to the use of an intermediate library it was not possible to realize the smooth trajectory but at intervals. This should be solved in the future.

A. *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*.

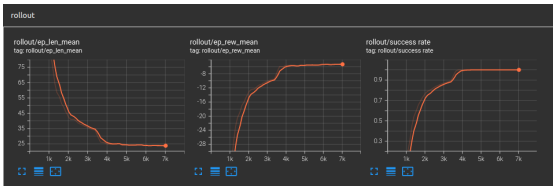

Fig. 12. IRB14000 - Reach - DDPG - SPARSE - Training

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 12, in just ten thousand episodes we have achieved a hit rate of 100%.

B. *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 13, it can be observed that a hit rate of 100% is not achieved at any time and that the limit value oscillates around 65%.

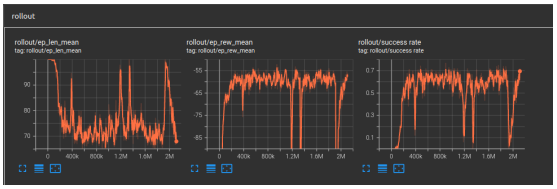

Fig. 13. IRB14000 - ReachVec - DDPG - SPARSE - Training

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

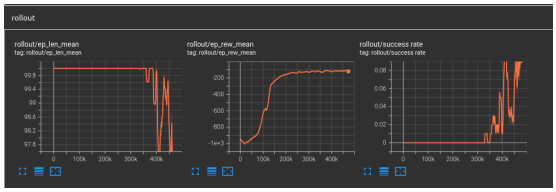

Fig. 14. IRB14000 - ReachVec - TQC - DENSE - Training

VIII. 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 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.

IX. 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.

References

[1] E. Trillas, *LA INTELIGENCIA ARTIFICIAL : MAQUINAS Y PERSONAS*, ser. Temas de Debate Series. DEBATE, 1998. [Online]. Available: https://books.google.es/books?id=0igNAAAACAAJ

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

[3] J. L. Gonzalez. (2022) Tipos de aprendizaje automático. [Online]. Available: https://medium.com/soldai/tipos-de-aprendizaje-automtico-6413e3c615e2

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

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

[6] L. e. l. Wikipedia. (2022) Denavit–hartenberg parameters. [Online]. Available: https://en.wikipedia.org/wiki/DenavitHartenberg_parameters