Implementation of a tossing ball robot using Reinforcement Learning

Stevedan Ogochukwu Omodolor[1†], Guillem Cornella Barba[1†] and David Hernández Sardà[1†]

[1*]Robotics and Automatic Control, ETSEIB, UPC, Avinguda Diagonal, 647, Barcelona, 08028, Catalonia, Spain.

Contributing authors:
stevedan.ogochu.omodolor@estudiantat.upc.edu;
guillem.cornella@estudiantat.upc.edu;
david.hernandez.i@estudiantat.upc.edu;
[†]These authors contributed equally to this work.

Abstract

In this project, we are implementing a tossing ball robot that learns by means of Reinforcement Learning. The robot is able to determine its optimal actions by using Computer Vision techniques, combined with different solutions, such as Q-learning and DDPG+HER, which have been broadly researched to have a higher accuracy. Simulations have also been done with OpenAI Gym and ROS (Gazebo), prior to the real implementation, to deeply study the best approach and prepare the model for the real scenario.

Keywords: tossing ball robot, reinforcement learning, computer vision, DDPG, HER, Q-learning, Gazebo, openAI, ROS

Introduction

This report includes, on one hand, a research about current-existing solutions similar to the one under study.

On the other hand, this document also includes the design and development of the proposed Reinforcement Learning solution, based on a simulated environment and prototype, which would imitate the real behavior of the solution, with the advantage of easing the iterative process of testing and redoing. Furthermore, after having a working simulation prototype, a real implementation has been done using self-made hardware and software.

The structure of the document is the following: Section 1 corresponds to the the state of art, and contains related literature to the topic, which inspired the prototype development; Section 2 describes the problem statement, in terms of the main aspects to take into consideration while developing the best solution to it; Section 3 describes the proposed solution, in terms of the main idea, and the implemented hardware and software in detail to tackle the scope of the project; Section 4 includes a concise analysis of the achievements and the barriers that have arisen during the development; Section 5 corresponds to the steps that should further be followed in order to have a more complete and robust solution. Finally, Section 5 includes the conclusions.

1 State of Art

This section aims to summarize other prior solutions that tackle the same problem, or a similar one, at the same time that the best characteristics of these solutions are studied.

The first subject of study is the paper "Purposive Behavior Acquisition for a Real Robot by Vision-Based Reinforcement Learning" [1] which implements a very smart mechanism called *Learning from Easy Missions* (or LEM) to speed up the learning time reducing it from exponential to almost linear order.

Second subject is "Teaching a robot to perform a basketball shot using EM-based reinforcement learning methods" [2], in which the authors approach the implementation of a robot arm that learns how to throw balls into a cup using dynamic movement primitives (DMP) to compute the arm's trajectories and expectation maximization (EM) algorithms with and without prior knowledge to learn good weights for the DMPs.

Last but not least, in "TossingBot: Learning to Throw Arbitrary Objects with Residual Physics" [3]. This paper focuses its attention on making a robot capable of throwing objects with totally random shapes and weight distributions, making them follow a predefined trajectory. This implementation consists of a neural network that takes as input a visual observation of objects in a bin and the 3D position of a target landing position. This outputs a prediction of parameters used by two motion primitives for grasping and throwing respectively. Their objective is to optimize this prediction in a way that it results in an object landing on a certain position.

2 Problem statement

The goal of this project is to build a functional prototype of a self-built robot that is able to toss a ball inside a bucket by using Reinforcement Learning. This robot will be located in a non-changing environment. The environment will be recreated in a virtual world for testing and implementation purposes.

2.1 Setup

As mentioned previously, the prototype has been developed both in a real and virtual environment. The setup consists of the robot, a camera, a ball and a bucket. The configuration of the prototype considers the following aspects:

- **Robot location**: the robot base is placed on the ground and has a stationary position. It doesn't move, only being able of changing its aiming direction (yaw) in the same plane as the ground and the shooting speed.
- **Camera**: camera is always at a fix position with respect to the robot base and facing towards the area where the bucket is placed. Thanks to the camera, the robot is able to determine the position of the bucket using a computer vision algorithm.
- **Shooting method**: the robot shoots the ball with a specific velocity and yaw direction thanks to two motors that control these parameters. The pitch angle is invariant and of 60º.
- **Bucket**: it is not stationary and it's four corners are always inside the Field of View (FOV) of the camera.

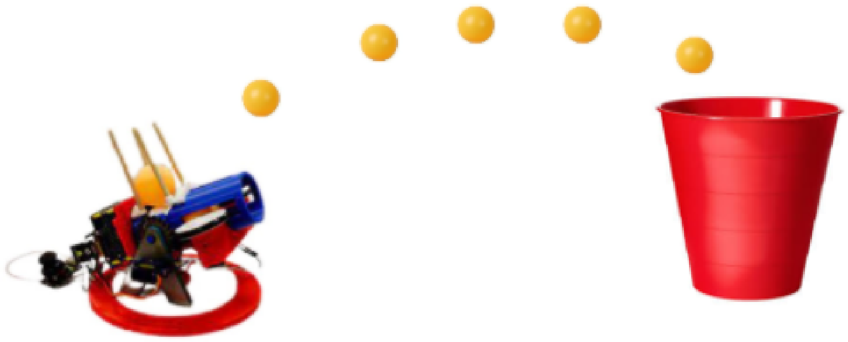


Fig. 1: Simplified environment key components.

2.2 Simulation

For the correct functioning of the prototype in the real world, a virtual environment like the above defined is created to perform the simulations. It consists of a simplification of the real world without loosing important physical information. This simulation consists of:

- **Camera** with similar characteristics of the real camera used.
- **The robot** is represented as a two cylindrical base, one of which rotates in the yaw angle and ball which velocity can be controlled.
- **Bucket** with similar characteristics as the real bucket.

- **OpenAI and Gazebo** used in combination to make interaction with the simulation environment and training easier.

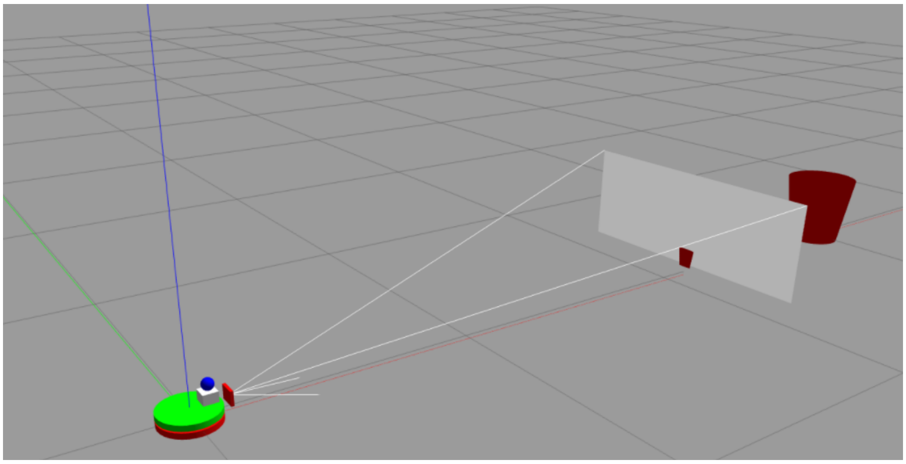


Fig. 2: Simulation Setup.

3 Solution approach

The robot consists of two big parts: Hardware and Software. In this section a tear down of these two groups is done and further details of the parts that compose each of these are explained.

3.1 Robot Hardware

3.1.1 3D design

The robot was designed from scratch using SolidWorks. All the parts were specifically designed to fit the needs of the robotic system. Then, all the components were printed using the ZmorphVX 3D printer and the Zortrax M200 3D printer. The parts are open-source and can be downloaded from [4]

3.1.2 3D Prototype

The electrical scheme was also designed by us, and it contains all the necessary components to perform the basic actions:

- Point towards the bucket using a servo motor mg996r.
- Shoot the ball with a specific velocity by using 2 DC motors controlled by a L298N motor driver.

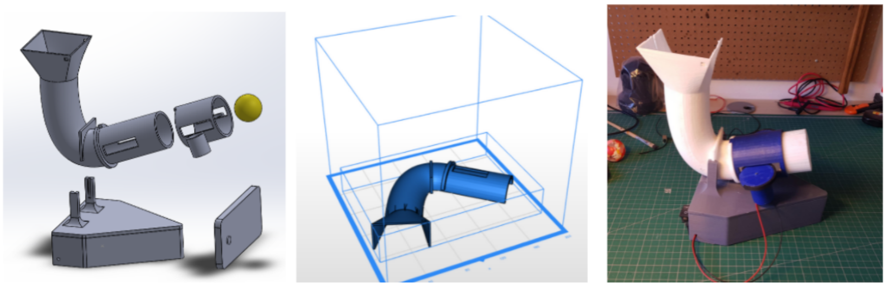


Fig. 3: 3D assembly - 3D printing - 3D final model.

- Receive data from the Python server (that is learning the actions) using an ESP8266 Wi-Fi board, and send feedback to the server to check for a successful UDP communication.
- Capture images from an IP camera to calculate the states of the environment. The device used was a Samsung S8+ smartphone sending data through an IP protocol.
- Power the entire system with a LiPo battery from an RC toy car.

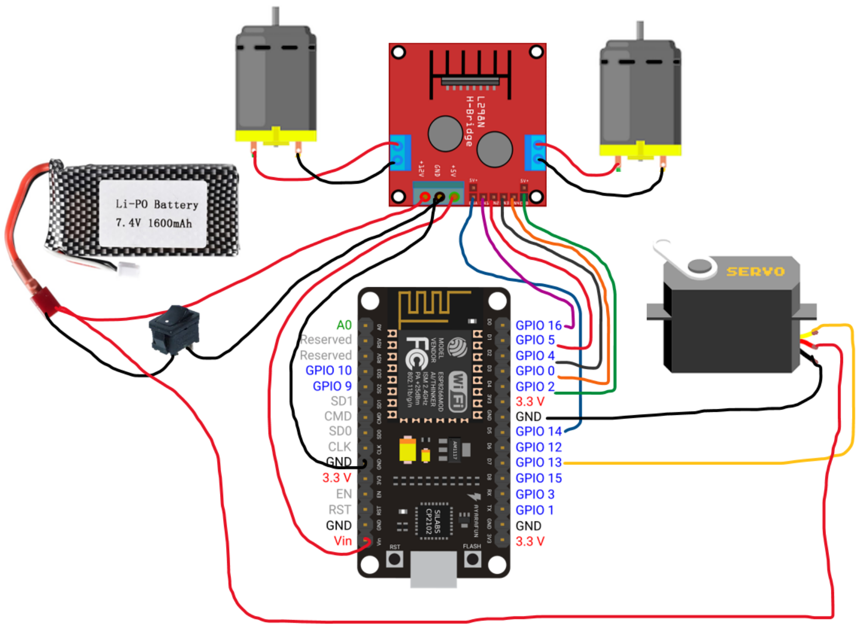


Fig. 4: This graph shows all the electrical components of the system and its adequate wiring

The tutorial from [5] has been very useful to understand the behavior of the motor driver to provide a variable velocity to the DC motors.

3.2 Robot Software

The full code implementation can be seen in [6].

3.2.1 Computer Vision for Image Processing

In order to get the states of the environment, we used a camera to detect the main dimensional properties of the bucket. For that purpose, we followed several steps:

- Connect to an IP Camera using the Android App 'IP Webcam' to get the frames that will be then processed.
- Use color segmentation techniques to extract the red objects from the frames (the bucket is red).
- Use morphological operations to smooth the masked object, particularly use image opening to eliminate blobs and fill holes.
- Then, get the coordinates of the corners of the bucket.
- Finally, the corners will be slightly modified to transform them into the model states (obtaining the features of the NN).

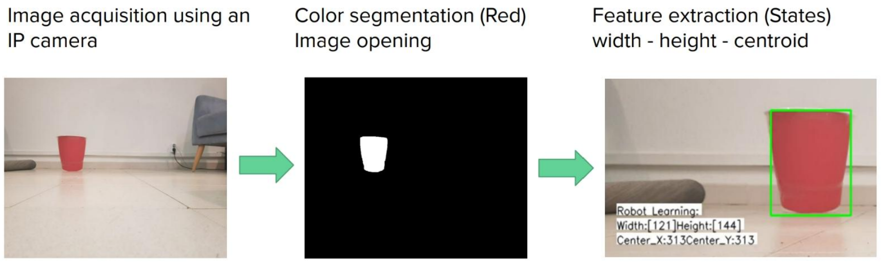


Image acquisition using an IP camera
Color segmentation (Red) Image opening
Feature extraction (States) width - height - centroid
Robot Learning:
Width:[121]Height:[144]
Center_X:313Center_Y:313

Fig. 5: The Computer Vision process used to get the states of the model

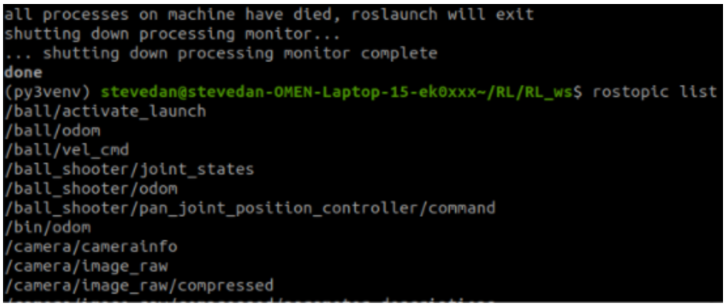
3.2.2 Communication protocol

This step consists on sending the learned actions from the RL model to the robot. To do this, a UDP protocol has been used to send data from the Python server to the ESP8266 Wi-Fi client. The robot's controller (the ESP8266 board) receives this data and maps the actions into feasible outputs to control the motors and the servo.

3.2.3 Simulation environment

In order to simulate the environment, Gazebo was used. The robot shown in Figure 2 is a simplification of the real robot while maintaining similar physical characteristics, some of which are:

- **Command**: The simulation environment has two command, yaw angle executed using a joint controller provided by ROS, and the initial ball launch velocity in which a custom ROS plugin was created. This commands can be seen in Figure 6.
- **Sensors:** To understand the current state of the simulated robot, the following sensor data are available, the joint position and the position of the ball and the bucket(6).
- **Interaction with Gazebo:** In order to interact with gazebo to facilitate the Reinforcement learning training, the following commands were implemented: Reset, pause and unpause simulation and reset controller.



```
all processes on machine have died, roslaunch will exit
shutting down processing monitor...
... shutting down processing monitor complete
done
(py3venv) stevedan@stevedan-OMEN-Laptop-15-ek0xxx~/RL/RL_ws$ rostopic list
/ball/activate_launch
/ball/odom
/ball/vel_cmd
/ball_shooter/joint_states
/ball_shooter/odom
/ball_shooter/pan_joint_position_controller/command
/bin/odom
/camera/camerainfo
/camera/image_raw
/camera/image_raw/compressed
```

Fig. 6: Available rostopic to interact with simulation

3.2.4 Defining the environment with Gym AI

In order to facilitate the use of different reinforcement algorithms with the Gazebo simulation, a customized gym environment was created which include a reset and step functions. The reset does the following: unpauses the simulation, reset the gazebo environment, sets the bin location based on the goal, resets the controllers and finally pauses the simulation in order to return the current state of the environment.

With respect to the Step function which is executed each episode, first the simulation is unpaused, the actions obtained from the policy are then sent to the robot by first moving the angle joint and then launching the ball. Finally, the simulation is paused in order to obtain the current state.

3.3 Reinforcement Learning Algorithm

This section explains the algorithms and strategies used to train the robot. It is worth noting that for every algorithm used, for each episode, one step is performed where the ball is tossed and the process is repeated.

3.3.1 Learning from easy missions

In order to improve the training rate, the whole task was split into multiple parts, where the easy task builds upon the previous. This technique is known as learning from easy mission(LEM) [1]. This was employed to guide the robot in each learning phase and help the training result converge. The learning procedure is set up in a way that the robot learns in easy situations in early stages of training and later learn on more difficult situations.The LEM stages using as reference Figure 7 are as follows:

- **LEM 1:** Fixed pan joint angle + fixed bucket position.
- **LEM 2:**Fixed pan joint angle + fixed bucket position in y direction, free in x direction.
- **LEM 3:** Free pan joint angle + free bucket position in y direction, fixed in x direction.
- **LEM 4:** Free pan joint angle + free bucket position
- **LEM 5:** Free pan joint angle + free bucket position

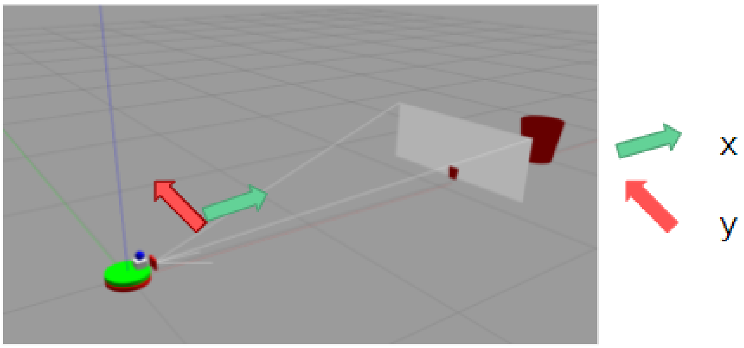


Fig. 7: LEM Reference axes

3.3.2 Q-Learning

Q-learning algorithm uses an action-value function Q^{Π} in order to find an optimal policy Π^{*} The optimal policy will consist of a map that relates the states s and the actions a that maximizes the $Q^{*}(s,a)$ [7]. The definition of the problem based on the Markov Decision Process using the Q-learning is as follows:

- **States:** The state of the system is represented in pixel position of the bucket: center(x,y), width and height $[C_x, C_y, W, H]$.
- **Goal:** Goal is to place the ball in the ball inside the bucket.
- **Actions:** The action space is discretized. The velocity is discretized in 1m/s intervals with range 4 to 7 m/s while the joints angles in 5 ° interval with the range -30 to 60 °. This gives a total of 36 possible actions.
- **Reward:** Reward shaping was used, where the a reward of 1000 is given when the ball is inside the bucket. In order to encourage and guide the learning, reward is giving based on the distance to the basket when the ball falls outside the bucket.
- **Exploration Strategy:** In order to explore the action space, the epsilon greedy algorithms was used.

3.3.3 HER

The idea behind HER is straight forward: after executing each episode, not only the original goal is stored in the replay buffer used for training the Neural nets but a subset of other goals. [8]. With respect to the current implementation, when the ball does not enter the bucket, the bucket is moved close to the ball location. The assumption of where the bucket should be is based on the fact that the elevation angle at which the velocity is shoot is high enough that the ball drops as horizontal as possible. Then the state is relabelled to the current bucket location and the reward is changed according.

3.3.4 DDPG

Deep Deterministic Policy Gradient(DDPG) [9] is a model-free RL algorithm for learning continuous actions. In DDPG, two neural networks are used. First, actor(policy) that proposes the action given a state. Then the critic that predicts if the action is good or bad given a state and an action. In addition, two more target networks are used, target_actor and target_critic, which based on the author adds stability to the training. It also uses an experience replay that contains a list of tuples: $(state, action, reward, next\_state)$. In order to learn, experience from the replay buffer is sampled.[8]

- **States:** The state of the system is represented by the pixel location of the 4 corners of the bucket. State values are normalized before being used as input to the NN or in order to train the NN.
- **Goal:** Goal is to place the ball in the ball inside the bucket.
- **Actions:** The action is continuous and consist of the pan angle of the robot limited within the range (-30, 30)° and velocity within the rage (4,7)m/s .Actions are scaled to the range(-1,1) before been used in training the NN in order to improve results.
- **Reward:** Two reward strategies were employed. First, without the use of HER, reward shaping similar to Q-learning was used. When using Her, a sparse reward system was used, 0 if ball is inside the bucket and 1 if not. This simplified immensely when training in the real robot.

- **Exploration Strategy Action/HER:** In order to explore the action space, the epsilon greedy algorithms was used where Gaussian noise was added to action. In addition, to decide when to use the HER, an epsilon-greedy apporach was also used.

3.3.5 Experimental Procedure

In order to train the simulation and the robot, a cyclical strategy shown in Figure 8 is used. First the model is trained in the simulation in the LEM1. The weights obtained from the simulation is used to train the real robot. The resulting weights are then used to train the simulation again. A video of the experiments and results can be seen in https://vimeo.com/670613697.

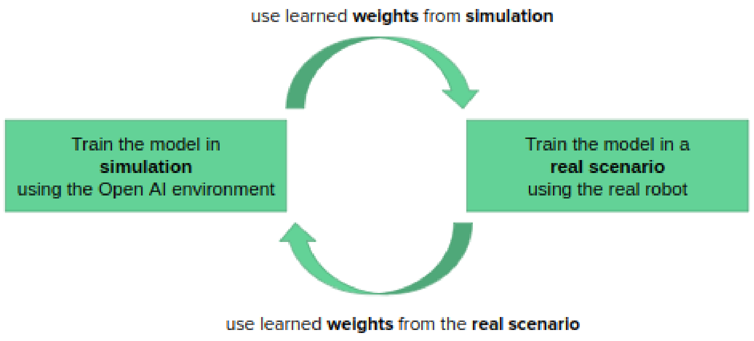

use learned **weights** from **simulation**
Train the model in **simulation** using the Open AI environment
Train the model in a **real scenario** using the real robot
use learned **weights** from the **real scenario**

Fig. 8: Experimental procedure

4 Achievements and barriers

4.1 Simulation

Initially, in order to train the the robot, the Q-learning algorithms with state, action and reward explained previously. The result of the training can be in Figure 9

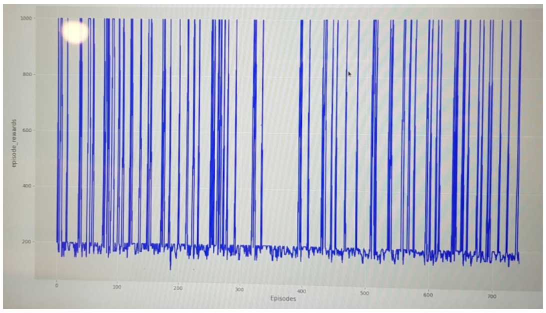


Fig. 9: Training results using Q-learning

As can be observed, the result performed badly and had difficulty converging. This is probably due to high number of states used.

With the poor result obtained from the previous algorithm, DDPG was used because it works well with continuous state and action space. LEM1 was used. DDPG was combined with reward shaping explained in previous section and epsilon greedy for action exploration. Some of the result obtained can be seen in Figure 10, where the image on the left does not converge while the one on the right does.

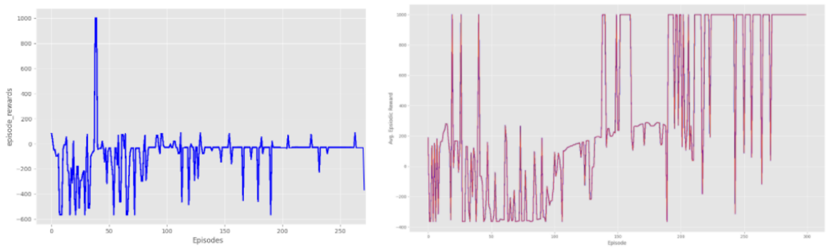


Fig. 10: Training results using DDPG

As can be observed in the previous figure, in some cases the reward converges. Although any slight changes in position affected immersively the result as can be seen in the image on the left.

To combat this problem, DDPG + HER + sparse reward was used. This greatly simplifies the problem, especially during training in the real robot. The training using the HER was done up to LEM2, where the position of the bucket was limited only in the x direction based on Figure 7. Although, it is worth noting that the goal during train was fixed(LEM1). The result of this algorithm can be seen in Figure 11.

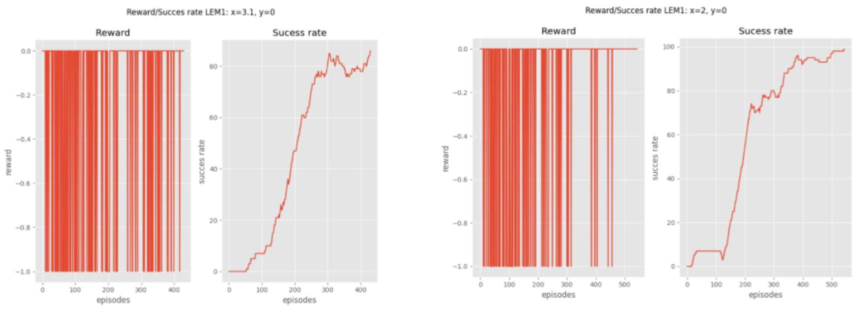


Fig. 11: Training results using DDPG + HER + Sparse Reward

The result shows two different goals(LEM2): Where the weights obtained from the first training was used to train the second goal. As can be observed, the algorithm performed well reaching a success rate(averaged over 100) of over 85% in the first case and up to 90% during the second training.

4.2 Real environment

With respect to the achievements, despite the tedious experimental procedure (tossing the ball, picking it up...and repeat), we managed to do around 80 episodes per experiment. We tried with DPPG alone (LEM 1 conditions) and also combining DDPG + HER (an improved version of LEM 1, equivalent to LEM 2). Since using HER implies modifying the states (moving the bucket in the x position - LEM 2), the training is slower but faster than DDPG working alone in the LEM 2 conditions. As it can be seen in the following figure, after 60 episodes we are getting a success rate around 40(averaged over 100).

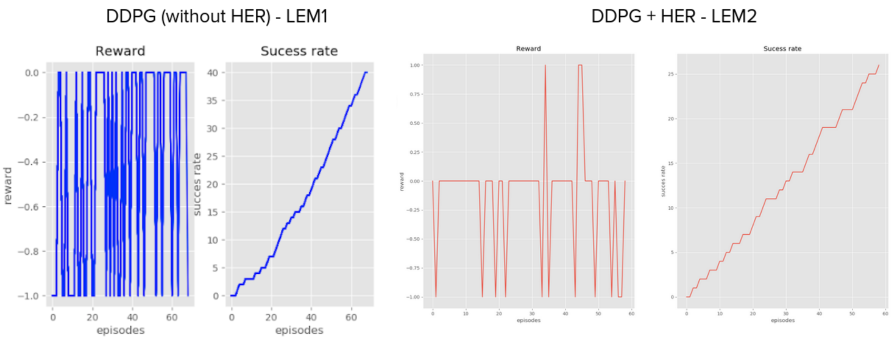
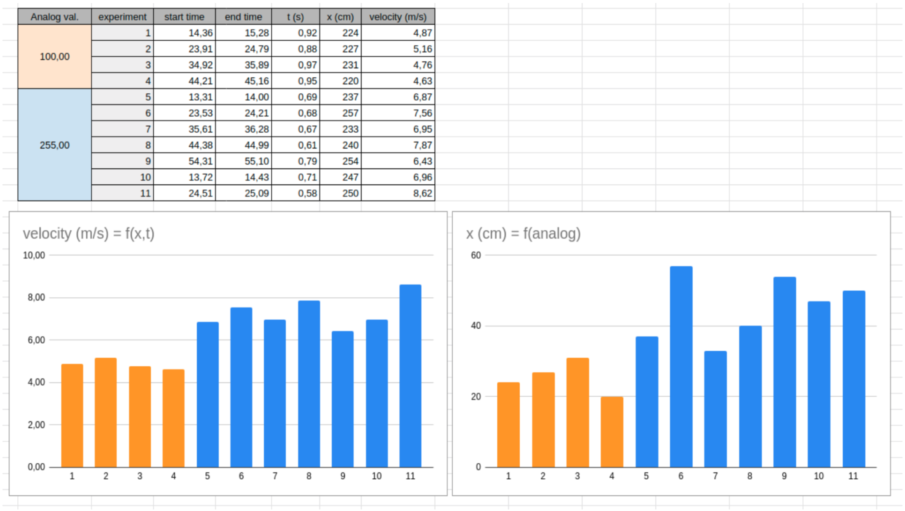


Fig. 12: Training results from the real scenario

With respect to the barriers, in order to know the range of velocities that the robot should use as its actions, an experimental procedure was implemented by throwing several balls and calculating the flying time (t) and the throwing range (x). As it can be seen in the following figure, the x position changes quite a lot, hence the fact that it is complicated to train in the real scenario.



Analog val.	experiment	start time	end time	t (s)	x (cm)	velocity (m/s)
100,00	1	14,36	15,28	0,92	224	4,87
	2	23,91	24,79	0,88	227	5,16
	3	34,92	35,89	0,97	231	4,76
	4	44,21	45,16	0,95	220	4,63
255,00	5	13,31	14,00	0,69	237	6,87
	6	23,53	24,21	0,68	257	7,56
	7	35,61	36,28	0,67	233	6,95
	8	44,38	44,99	0,61	240	7,87
	9	54,31	55,10	0,79	254	6,43
	10	13,72	14,43	0,71	247	6,96
	11	24,51	25,09	0,58	250	8,62

Fig. 13: Experimental results, throwing a ball several times

We are sending an analog value to the DC motors of 100 when we want to have approximately an output velocity of 4m/s, and we are sending 255 analog if we want output velocities around 8 m/s. The friction of the ball with the motors limits the performance of the model, yet it works better including the standard deviation in the algorithm. A better design would be implemented if we started again from scratch, in order to eliminate the friction, for instance, by using a pneumatic actuator with a spring to toss the ball.

5 Future work

In order to enhance the robot's performance, we are proposing several improvements. First to improve the robot accuracy, the robot architecture/design is to be modified to obtain a higher accuracy when throwing the ball (there is a lot of friction currently, hence deviation in the range). We would perhaps use a pneumatic actuator.

Second, repeat the cyclic training procedure several times, i.e, using the weights obtained in simulation for the real scenario, and the weights from the real environment to train again the simulation including the real conditions following the cycle shown in Figure 8.

Finally, to improve training and encourage better exploration, try to use other sources of noise, for instance the adaptive parameter noise instead of the Gaussian noise used.

6 Conclusions

In this report, a robot is taught to toss a ball into a bucket using reinforcement different reinforcement learning techniques. In order to do so, a prototype has to be built from scratch both in the real world and in simulation.

Despite the limited robustness of the robot, the results presented before are quite promising, both in simulation and in the real-scenario. Using DDPG with HER is a good choice for this kind of application where you need trial and error experiments, and its success rate was good and fast. Adding HER is very useful when the states are varying and the model learns faster.

There are some variations that ought to be made to enhance the performance, however, the designed first prototype has a huge potential.

References

[1] Asada, M., Noda, S., Tawaratsumida, S., Hosoda, K.: Purposive behavior acquisition for a real robot by vision-based reinforcement learning. Dept. of Mech. Eng. for Computer-Controlled Machinery. Osaka University (1996)

[2] Michels, T., Hochländer, A.: Teaching a robot to perform a basketball shot using EM-based reinforcement learning methods. TU Darmstad, 1–5 (2012)

[3] Zeng, A., Song, S., Lee, J., Rodriguez, A., Funkhouser, T.: Tossingbot: Learning to throw arbitrary objects with residual physics. Columbia University, Princeton University, MIT and Google, 1–12 (2019)

[4] Cornella, G.: Ball launcher 3d model, https://github.com/gcornella/Ball-Launcher-Reinforcement-Learning. Universitat Politècnica de Catalunya, 1 (2022)

[5] Blog: L298n motor driver – arduino interface, how it works, codes, schematics, https://howtomechatronics.com/tutorials/arduino/arduino-dc-motor-control-tutorial-l298n-pwm-h-bridge/. How To Mechatronics, 1 (2020)

[6] Stevedan Ogochukwu Omodolor, G.C.: Ball launcher 3d model, https://github.com/stevedanomodolor/robot_learning.git. Universitat Politècnica de Catalunya, 1 (2022)

[7] El-Fakdi, A.: Gradient-based reinforcement learning techniques for underwater robotics behavior learning. PhD thesis, University of Girona, Girona

(December 2010). Computer Engineering Department, computer Vision and Robotics Group(VICOROB)

[8] et al, M.A.: Hindsight experience replay. 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA. (2017)

[9] Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning (2019) arXiv:1509.02971 [cs.LG]