Lebry: A Robot That Plays Basketball Via Reinforcement Learning

Victor Centellas, *Student, MUAR,* and Juan Gallostra, *Student, MUAR,*

Abstract—This paper introduces Lebry, a robot that plays basketball. In order to achieve the goal there are several devices that interact. First of all a camera collect a set of images of the screen and are processed using a Image Recognition Algorithm in order to detect the basket and ball position. After that, the position of the ball and the basket are the input elements of the Q-Learning algorithm which decides according to the input which is the best action to take. The action is sent to the servomotors which perform the desired movement.

Index Terms—Q-Learning, Basket, Robot, Inverse Kinematics, Computer Vision, Image Recognition, Raspberry Pi, Arduino, Reinforcement Learning, Robot Learning

1 Introduction

Many self learning applications have been created to help humans with ordinary tasks in recent years. However the reinforcement learning can also be applied to a robot and learn how to play games beating the humans. There is plenty of bibliography related to this field, as for example Giraffe [1], a chess engine that uses self-play to discover all its domain-specific knowledge. Another example could be BetaPoker [2], a reinforcement learning for heads-up limit poker.

With the technological advances, mobile phones have reach our day to day and robots have to be able to interact with them. At this point we can introduce Lebry, a robot that plays a basketball game using a mobile phone. The aim of the game is to basket a ball from a set of different points achieving the biggest score. This can be carried out implementing a basic Q-Learning algorithm in cooperation with a set of systems that interact between each other. Lebry has been designed and constructed from scratch. Beginning with a 3D model created using a Computer Aided Design software and printed using 3D printers. The construction have been done taking into account the interaction between the other elements, assuring no interference between them. Finally, Lebry has been tested and self-trained in order to beat humans playing the basketball game.

The rest of the paper is organized as follows. In the next section, we introduce de proposed approach of the different modules that have to interact to achieve our goal. Section 3 details the developed system, differentiating between the robot with its design, construction and inverse kinematics; the computer vision part composed by the ball, basketball and score detection and the learning, describing the algorithm and how the training have been performed. The results obtained are summarized in section 4. Finally, Section 5 gives a summary and future work in order to improve the robot capabilities.

2 Proposed Approach

The first step towards the robot construction is to design the different subsystems and propose how they are going to interact between each other. The system overview is presented in Figure 1 and consists of four different modules. This modules are:

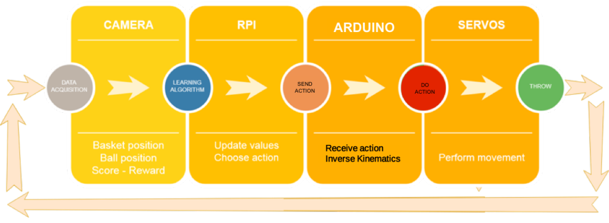

Fig. 1. System overview composed of four modules.

- **Camera:** For the estimation of the ball and basket positions some data need to be collected. A camera is suspended from the air taking photos of the screen. This photos are going to be processed by a Computer Vision algorithm, detailed in Section 3.2 in order to extract the desired information.
- **RaspberryPI:** Is the core of the robot and have several key functions. On one hand, extracts information of the photos taken by the camera using Image Recognition software. On the other hand, has implemented the Q-Learning algorithm which decides the actions to be taken and finally sends these actions to Arduino board through the Serial Port.
- **Arduino Board:** In our original approach, this functions were also performed by the RaspberryPI. However due to some manufacturing problems in the RPi board, the servos were not working. For that reason

we decided to use the Arduino board, which receives from the Serial Port the actions to be taken. There are two different types of actions:

1) Place the arm to the ball position.
2) Perform the throw of the ball.

And perform the action using the implemented Inverse Kinematics of the robot sending to the servo the desired angle to be moved.

- **Servomotor:** 3 different servomotors are placed on the robot. With them the robot can achieve every necessary position needed to reach the ball positions and basket. All of them are connected to a 5V power supply and to the Arduino board using the control signal.

3 Developed system

In this section we are going to explain in a deeper way the different systems mentioned in Section 2. The system is composed by three main parts: The robot itself, the computer vision and the learning parts.

3.1 Robot

The robot itself has been built from scratch. A step by step guide describes all the elements needed for the construction and the inverse kinematics are derived in order to move the robot joins to the ball position, according to the data obtained from the camera.

3.1.1 Design and construction

In this subsection we will explain the construction of the robot step by step. The robot will be placed in a wooden platform. The different parts of the robot have been printed using a 3D printer. For that a 3D model of the robot have been designed using SolidWorks, as can be seen in Figure 2.

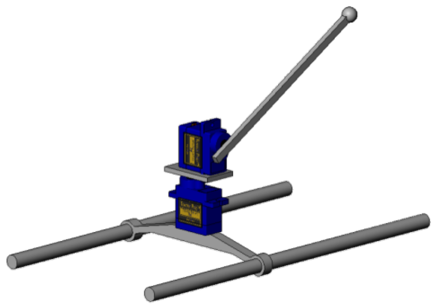
Fig. 2. 3D model of the robot, designed using SolidWorks.

The first step is to place the guides that allow the horizontal movement of the robot when the ball has to be thrown. To achieve that two guide stops have been sticked on the wood. Before placing the guides inside its limits the robot base has to be mounted on the guides by placing them between the holes made for it. On the bottom part of the base there is a servo motor placed in order to allow the rotation along the vertical axis.

In order to finish the robot construction the second part of the robot has to be placed on its base. A small platform is placed on top of the servo in order to allow another servo to be placed in top of that. The goal of this servo is to allow the movement of the robot's arm. Finally it has to be noted that in order to allow the arm's end to be detected by the phone screen a wired has to be connected between the top of the arm and the ground. The final structure of the robot can be seen in Figure 3.

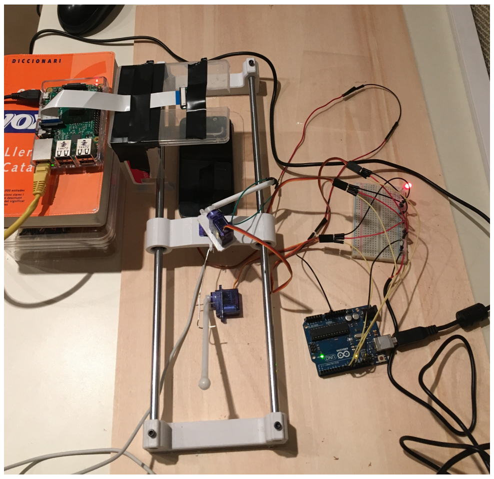
Fig. 3. Final structure of the robot.

3.1.2 Inverse knematics model

The ball position is calculated taking as reference the bottom right border of the screen. In order to move the joints to the desired position of the ball the inverse kinematics are required. To do that, the forwards kinematics of the robot must be derived and are stated by the following equations.

$$x = l_3 cos\left(\alpha\right) sin\left(\beta\right) \quad (1)$$
$$y = l_2 + l_3 cos\left(\alpha\right) cos\left(\beta\right) + l_1 cos\left(\gamma\right) \quad (2)$$
$$z = z_1 + l_3 sin\left(\alpha\right) \quad (3)$$

where x is the coordinate perpendicular to the guides, y the one along they, z states the vertical height, α is the inclination of the arm, β is the angle related to the horizontal orientation of the platform, γ states the orientation of the servomotor in charge of moving the robot along the guides, l_3 is the arm's length, l_2 is the length of part that connects the servomotor support and the base of the robot, l_1 is the length of the servomotor support and z_1 is the mobile thickness. The obtained inverse kinematics are:

$$\alpha = arcsin\left(\frac{z - z_1}{l_3}\right) \quad (4)$$

$$\beta = arcsin\left(\frac{x}{l_3 cos\left(arcsin\left(\frac{z - z_1}{l_3}\right)\right)}\right) \quad (5)$$

$$\gamma = arcos\left(\frac{y - l_2 - l_3 cos\left(\alpha\right) cos\left(\beta\right)}{l_1}\right) \quad (6)$$

with $l_1 = 4.5$ cm, $l_2 = 9$ cm, $l_3 = 7.5$ cm and $z_1 = 7.6$ cm. The obtained angles are in rad and need to be converted to degrees.

3.2 Computer Vision

The Computer Vision algorithms explained in this section are a key part of the presented project. This is true because both the states and the rewards, whose correct computation is crucial to train the robot, are obtained with Computer Vision techniques. This algorithms are the only means by which the robot is able to sense the environment and know what is happening around. This is the reason why there is a need to find robust and reliable algorithms.

Talking more specifically, as already explained in section 2, these algorithms are in charge of:

1) Detecting the position of the ball and the basket in the screen, required to know the state of the environment where the robot operates.
2) Detecting the current score of the game, required to get feedback for the actions.

Figure 6 presents the main screen of the game as seen with a smartphone or tablet. This is what the camera sees, and from where the ball and basket have to be detected. Both algorithms are explained in the following sections. We implemented them in Python, using the Python bindings of the OpenCV library [4]. For the score recognition one we also used the Python bindings of Google's Tesseract OCR library [5].

3.2.1 Ball and Basket detection

The algorithm that we used to extract the coordinates of the center of the ball and the basket is presented in Algorithm 1. There are some parameters that we had to tune by hand in order to achieve the desired results of robustness and reliability. This parameters mainly depend on and help in addressing the errors derived from the poor camera resolution, its position with respect to the smartphone screen and the noise present in the captured image. The main parameters that we had to tune were the filter sizes, the regions of interest (ROIs) and the bounds for the areas of the basket and the ball.

The main idea behind the algorithm we developed is to first smooth the image, reduce the noise and make the interest parts bigger with the morphological operation. We have to point out here that we eroded instead of dilating because after binarizing both the ball and the basket appear in black. Finally, we extract the centers from the contours, where m is notation for moment and the subindex specifies which moment we are referring to. Figure 4 presents the detected ball and basket for a particular frame captured by the camera with the final setup.

3.2.2 Score detection

When it comes to the score detection, we developed two different algorithms but ended using only the most reliable one. The first approach was to use Google's Tesseract OCR Library to extract the number of the current score and use it to detect if the throw had been successful. This worked great when working with screen captured games but led to unreliable results when using it in the real setup. This is why we developed the second algorithm, much simpler but that proved to work better in the real setup. This second approach, instead of computing the actual score, was focused on detecting only when Lebry had missed a throw. Figure 6 presents the differences in the rendered screen during the game and when the game is over. It can be observed that when a throw is missed the score turns blue and the message *Current Best* is printed to the screen. To detect this game over situation we defined a new region of interest that covered the area of the message. We binarized and preprocessed this region of interest and after that, if the total amount of black pixels was over a threshold we concluded that the message had been rendered and hence Lebry had missed a throw. If this was not the case we decided to assume that the throw had been successful.

Before entering the last part of the project, the learning algorithm, it is worth pointing out that the developed robot

Algorithm 1 Ball and Basket detection algorithm

```
 1: procedure DETECT CENTER($frame$)
 2:     Binarize with an adaptive gaussian threshold
 3:     Crop image to ROI
 4:     Apply median blur filter of size 5 to ROI
 5:     Erode ROI with a radius 5 circle
 6:     Compute contours
 7:     for contour in contours do
 8:         Compute contour moments
 9:         Compute contour area
10:         if lower bound < area < upper bound then
11:             $x_{center} = \frac{m_10}{m_00} + x^0_{ROI}$
12:             $y_{center} = \frac{m_01}{m_00} + y^0_{ROI}$
13:             center ← $(x_{center}, y_{center})$
14:             return center
15:         end if
16:     end for
17:     center ← $None$
18:     return center
19: end procedure
```

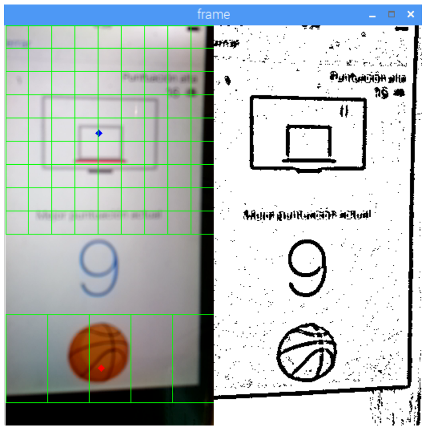

Fig. 4. Ball and basket detection with the developed algorithms.

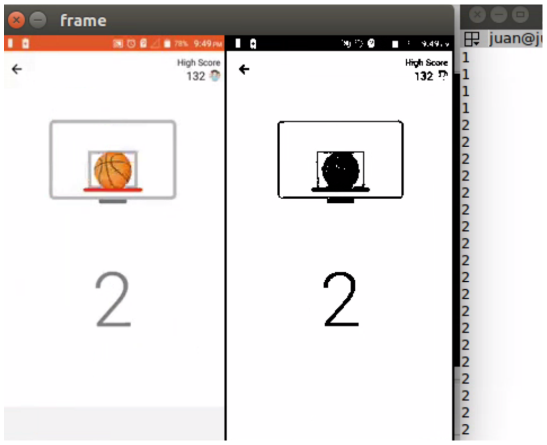

Fig. 5. Score detection performed on a frame of the game.

and algorithms only allowed us to detect the ball and basket once every 7 seconds. This made it impossible for us to achieve that Lebry worked in real time. As a consequence, we had to reduce the scope of the project and focus on the first stage of the game, the first 10 baskets, where the basket is not yet moving.

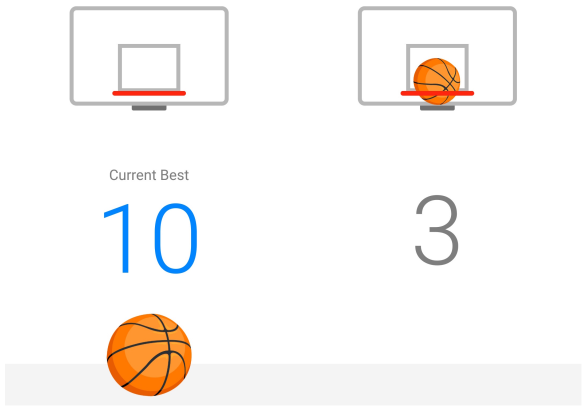

Fig. 6. Comparison of the rendered screen when its game over (left), and duing the game (right).

3.3 Learning

The last stage towards achieving a fully functional working robot able to fulfill the objectives of the project was to define and implement its learning strategy. Sections 3.3.1 and 3.3.2 go in detail about this aspect of the project.

3.3.1 Algorithm

We chose Q-Learning [3] as the learning algorithm due to its simplicity and verified efficiency. Furthermore, it is a very general algorithm that can be applied to many different situations and learning problems.

The main equation that governs the learning under the paradigm of Q-Learning is

$$Q(s, a) = r(s, a) + \gamma \cdot \max\left(Q(s', a')\right) \tag{7}$$

Where $Q(s, a)$ tells us the value of performing action a when being in state s, $r(s, a)$ is the reward obtained after performing action a from state s, γ is the discount factor and $\max(Q(s', a'))$ is the predicted maximum future reward. This means that to be able to implement Q-Learning we had to define a set of states, actions, rewards and initial Q-values for the matrix $Q(s, a)$. We initially set the discount factor γ to 0.5 to equally favour exploration and chosing the known best action. However, we realised that, in this particular problem the evoultion of the states is independent. This is so because the ball appears at a random position after each throw so we finally set the discount factor to 0. We also decided to initialise the Q matrix to zero.

Regarding the actions, states and rewards, they demand a longer explanation. The states have to be chosen in such a way that results in the robot having enough knowledge of its environment to fulfill the desired tasks. In this case, it goes without saying that the two key values that the robot needs to be able to play the game are the ball position and the basket position. This is why we defined the state as a combination of this two values. Furthermore, since our original idea was to play the full game, not only the first 10 baskets, we also decided to include in the state the basket's position in the previous frames. To reduce the number of states and the complexity of the problem we decided to discretise both the basket area and the ball area. After some fine tunning the basket area was discretised to a 9×9 grid and the ball area in 20 divisions. Figure 4 shows an example of the discretisation of both areas. Instead of using the exact coordinates of the ball and the basket we finally defined the state as the grid coordinates combination of both the ball and the basket.

For the actions we decided not to use define them as a set of vectors representing movement directions of the arm, which would have been the obvious choice. Instead we defined them as a set of angles. This angles would be the final values of the base after the throw with respect to its resting position. We decided to use 18 actions that covered the angles ranging from 62 to 45. With the base at 53 degrees the arm was parallel to the longest side of the screen. This has an interesting effect which is that the same action results in a different effect or throw trajectory depending on the starting position of the throw.

Finally, we set the rewards to be 100 if the throw was successful and -100 otherwise.

3.3.2 Training

After having defined all the required parameters to implement the learning Lebry had to start training in order to learn how to play the game. Since the training process was slow due to the time constraints between throws and there are a vast amount of possible values for the paramateres (discretisation values and its combinations, set of actions, etc.) we only trained a small subset of the possible models to find, amongst them, the one that yielded the best results. It is worth noting that, although slow, a positive aspect of the built setup and training process of Lebry is that, once started, it was fully automated and required no human interaction. This is a consequence of the game restarting on its own when losing a game without having to press any buttons or perform any action. This eased a lot the training process because after starting the robot it could carry on the training process on its own.

From the different models that we trained the one that achieved the best results was using a set of 18 actions, a ball area discretisation of 20 divisions and a grid of 9×9 for the basket area.

4 Results And Conclusions

After two hours of continuous training Lebry was able to complete a full run of 10 consecutive baskets (https://youtu.be/baSNCdxkE-A). We achieved this result discretizing the basket area with a 9x9 grid and the ball area with 20 divisions. We set the number of actions to 18, with the final angle of the base ranging from 45 degrees to 62 degrees, both angles included. However, this does not mean that Lebry started to perform always that well. We achieved an average score of between 5 and 7 throws. Due to how we defined the actions, as already explained in section 3.3.1, throws from the right of the basket were harder to learn. This was due to the range of angles used to define the actions. When throwing from the right of the basket there were fewer actions that resulted in a successful throw and most of the available actions resulted in ball trajectories that were too diagonal even for the more extreme positions.

A part from the problem with the actions just mentioned above we also found that the feedback provided to the robot wasn't as reliable as we woukd like. The main problem here was that there were some times were the robot failed to throw the ball but instead of giving no reward the robot was receiving the reward of the previous action. This lead to inaccuracies in the learned model that affected the final performance.

All in all, we were to able to reach the objectives set at the beginning of the project. However, by improving the computation of the reward we could have learned a better model and hence increase the average number of consecutive successful throws.

5 Future work

From our point of view, there are two main lines of work that stand out when considering further work on this project. The first of them is to improve the efficiency of the detection algortihms. This would allow to perform not only the detection of the ball and the basket, but also the tracking of the basket once it starts moving after 10 consecutive baskets. As explained in section 3.2.1, with the current robot setup and the implemented algorithms 7 seconds are required to pass between two consecutive state computations. This would be the first step towards making Lebry able to cross the barrier of 10 baskets and hence learn the full game. The computation of the reward should also be improved to reduce the number of false positives and false negatives.

Secondly, once the algorithms and robotic setup have been modified and are able to operate in real time then the learning algorithm should be extended to also learn how to properly throw when the basket is moving. We think this could be achieved by extending the state of the system. A part from the current position of the basket we would also include in the state its position in some of the previous frames so that basket trajectory is taken into account when learning to play. The main issue with this approach is the huge amount of states that appear as a result since it grows exponentially as more frames are included in the state. The size of the Q-Matrix is $(n \times m)^k \times nb \times na$, where $n \times m$ is the number of cells in the basket area, k is the amount of frames used to derive the state, nb is the number of divisions in the ball area and na is the number of actions. If this possible extension of the learning algortihm is not viable then another learning strategy might have to be considered.

References

[1] M. Lai, *Giraffe: Using Deep Reinforcement Learning to Play Chess* London, England: Imperial College of London, 2015.

[2] A. Tung, E. Xu and J. Zhang, *BetaPoker: Reinforcement Learning for Heads-Up Limit Poker* California, USA: Standford University, 2014.

[3] Watkins, C.J.C.H, *Learning from delayed rewards*, PhD Thesis. England: University of Cambridge, 1989.

[4] Itseez, *Open Source Computer Vision Library*, online. https://github.com/itseez/opencv, 2018.

[5] Google, *Tesseract Open Source OCR Engine*, online. https://github.com/tesseract-ocr/tesseract, 2018.

Victor Centellas MUAR Student.

Juan Gallostra MUAR Student.