UNIVERSITY OF WEST BOHEMIA

Master Thesis

Analysis of the state of table football and prediction of its change based on image data

Pilsen, 2022

Author: Matěj Sieber

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2021/2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Matěj SIEBER**
Osobní číslo:	**A20N0032P**
Studijní program:	**N3918 Aplikované vědy a informatika**
Studijní obor:	**Kybernetika a řídicí technika**
Téma práce:	**Analýza stavu stolního fotbálku a predikce jeho změny na základě obrazových dat**
Zadávající katedra:	**Katedra kybernetiky**

Zásady pro vypracování

1. Seznamte se stávajícími metodami automatické detekce objektů a predikce jejich pohybu.
2. Implementujte vhodnou baseline metodu.
3. Implementuje vhodná rozšíření a vylepšení daného algoritmu.
4. Realizujte a otestujte metody na mechatronickém modelu stolního fotbalu s vhodnými HW a SW prostředky.
5. Zhodnoťte výsledky jak kvalitativně, tak kvantitativně.

Declaration of Authorship

I Bc. Matěj Sieber declare that this thesis titled "Analysis of the state of table football and prediction of its change based on image data" and the work presented is my own. This thesis follows on my work presented in Bachelor Thesis [1].

I declare that I have prepared the Master's thesis independently and exclusively using professional literature and sources, the complete list of which is part of it.

In Pilsen on: ....

Matěj Sieber

Abstract

This work aims to design an agent to play a mechatronic model of table football. This work is divided into two parts theoretical and practical. The theoretical part of this work presents reinforcement learning methods, a simulation environment in ROS in combination with Gazebo, and the results of learning the agent. The practical part introduces methods for player pose and rotation estimation as well as ball position estimation, these are necessary for the model to work. And results of the model in the real-world arena. Most of the code for this thesis was written in Python with the help of OpenCV, NumPy, and PyTorch libraries.

Key words

Object detection, Image processing, Kalman filter, Python, OpenCv, Neural network, Computer vision, Reinforcement Learning

Acknowledgements

I would like to express gratitude to my advisor Ing. Ivan Gruber, Ph.D. et Ph.D. for his support of my work, valuable comments, and supervision of this Master thesis. Next, I would like to thank Ing. Petr Neduchal for helping me with ROS software. I would also like to thank Ing. Miroslav Flídr, Ph.D. for providing space in one of the laboratories for the arena storage. Last but not least I want to thank me for believing in me.

Contents

List of Figures

List of Tables

Acronyms

A2C Actor critic. 15, 16

ARS Augmented random search. 15

DDPG Deep deterministic policy gradient. 15

DQN Deep Q learning. 15

E.g. Example given. 7

fps Frames per second. 32

Gym Framework from OpenAI. 15

PPO Proximal policy optimalization. 15, 16, 34

px Pixel. 26

REINFORCE REward Increment = Non-negative Factor × Offset Reinforcement × Characteristic Eligibility. 11

RL Reinforcement Learning. 3–5, 8, 13, 15, 22, 31, 33–36

ROS Robot Operating System. 13

SAC Soft actor critic. 15

Chapter 1

Introduction

The problem that we will try to tackle is the defense on simplified table football. The number of approaches to tackle this problem is vast. From recording game sessions and trying to mimic the players to the finely tuned expert systems. But the one that stands out is the reinforcement learning approach. The reason is that we do not need to collect or annotate training data. The initial thought was to let the machine play against the machine in silico, monitor the improvements, and then transfer the model and let it play against a human. This had to be slightly altered as the defined game is not symmetrical, thus it would require two models. Altering brought out the possibility to only define the shot set, which would keep the reinforcement learning for defense as it has no prior knowledge. But defined shot set could potentially limit the model's ability to improve.

The choice of reinforcement learning was motivated by papers, that reported successful implementation in a variety of games. One of the most known examples of reinforcement learning is AlphaGo. AlphaGo is the first computer program to defeat a professional human Go player, the first to defeat a Go world champion, and is arguably the strongest Go player in history [2]. Another mentionable article from Deepmind includes advancements in protein folding [3]. Or the newest one which was able to introduce improvements in the field of video compression [4]. This algorithm was firstly used to play computer games. The games are used as a great testbed for creating general-purpose algorithms.

In this work, we focus on exploring the potential of reinforcement learning as it could get better results than the other approaches. And we will try to apply reinforcement learning to the problem of simplified table football, firstly in simulation and then in real world application.

1.1 Master Thesis Objectives

The objectives of this study are:

1. to research current reinforcement learning algorithms
2. to develop a baseline method
3. to further improve the baseline method

4. to use the model in real-world application
5. to evaluate obtained results both qualitatively and quantitatively

1.2 Thesis outline

This thesis is divided into 6 chapters. Chapter 2 brings essential knowledge and introduction to the Reinforcement learning, and presents core concepts in section 2.1 and some limitations in 2.5.

Then the environment used for in silico training is shown in chapter 3 as well as the state-of-the-art algorithms in section 3.2. The section 3.2 also includes the selection of suitable algorithm and its results during training.

Hardware and software parts for the realization of real-world application are presented in 4. The hardware for my work includes the camera and industrial computer. Part of this chapter is also image processing and its results, together with the solution to issues with the industrial computer. Finally, the results for three variants of defending are presented.

Chapter 5 comes with the discussion about results. Comparison of results obtained by in silico training and real-life application. The theory relevance and limitations of the chosen approach. The Master Thesis is concluded in chapter 6.

All the codes and specifications of the environment are accessible in the GitHub repository: `https://github.com/sieberm111/master_thesis_sieberm`

Chapter 2

Reinforcement learning

Reinforcement learning is one of the machine learning areas, based on rewarding desired outcomes and punishing undesired ones. Basic scheme is in Figure 2.1. An agent has a set of actions that result in a new state and reward. Via trial and error is the agent able to understand the environment and act accordingly to maximize the reward.

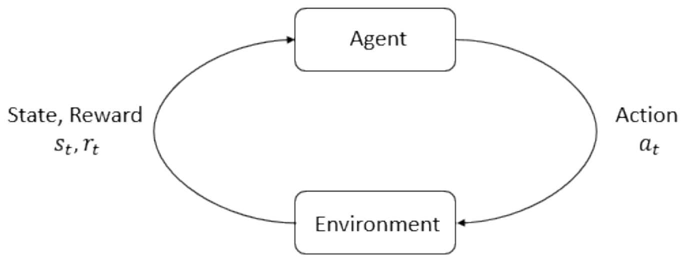


Figure 2.1: Basic RL scheme

Reinforcement learning recalls how people and animals learn by interacting with the environment. This differs from other types of learning it is rather active than passive. RL is sequential (future interactions can depend on earlier ones). It can learn without examples of optimal behavior instead of copying examples it optimizes the reward signal. Concurrently RL is based on the reward hypothesis. The reward hypothesis states that any goal can be formalized as the outcome of maximizing cumulative reward.

Examples	Reward
Fly a helicopter	air time
Video games	win, score
Board games	win

Table 2.1: RL examples

2.1 Core concepts

Now lets introduce some core concepts regarding RL which will be further explained.

- Environment: representing dynamics of a problem
- Reward signal: specifying goal
- Agent: containing
  - Agent state
  - Policy
  - Value function estimate - optional
  - Model - optional

2.2 Reinforcement learning formalisation

Per Figure 2.1, at each time step t agent receives observation O_t, reward R_t, and executes action A_t. The environment receives action A_t and emits new observation O_{t+1} and reward R_{t+1}. Reward is scalar feedback, and represents how well is agent doing at time step t. Goal is to maximise this cumulative reward for the future called return.

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... \tag{2.1}$$

It is possible to define utility of states and actions with expected cumulative reward $\mathbb{E}$ from state s with value $v(s)$.

$$v(s) = \mathbb{E}[G_t|S_t = s] \tag{2.2}$$

And can be further defined recursively.

$$G_t = R_{t+1} + G_{t+1} \tag{2.3}$$

$$v(s) = \mathbb{E}[R_{t+1} + v_{S_{t+1}}|S_t = s] \tag{2.4}$$

Now that the utility of states is known, we can pick action a, that maximize value v. It is important to note that some action may have long term effect and reward may be delayed. In this case it is better to sacrifice immediate reward to gain more long-term reward. A mapping from states to actions is called policy. It is also possible to map reward directly to actions. We have letter v for value function of states and letter q for value function of states and action. Letter q is used for historical reasons.

$$q(s, a) = \mathbb{E}[R_{t+1} + R_{t+2} + R_{t+3}|S_t = s, A_t = a] \tag{2.5}$$

2.3 Inside the Agent

Components of the agent were described in Section 2.1. The inside of the agent is visualized in Figure 2.2. The agent makes predictions and policy based on state, which is influenced by observation. From policy the action is chosen.

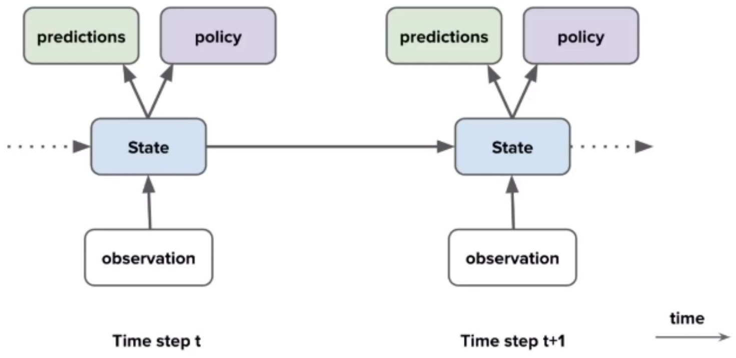


Figure 2.2: RL agent scheme

We can define a history of the agent as equation (2.6). This history is used to construct agent state S_t.

$$H_t = O_0, A_0, R_1, O_1, ..., O_{t-1}, A_{t-1}, R_t, O_t \tag{2.6}$$

2.3.1 Agent state

This brings up the observability of an environment. In the case of a fully observable environment $O_t = S_t$, this condition is rare as the agent usually has limited sensors. But it is useful because if we have a fully observable environment then the process of interaction becomes Markovian. This means that the state contains everything we need to know and adding history does not help. We can define, that process is Markov if equation (2.7) is satisfied. Thus if the state S_t is known the history H_t can be omitted.

$$p(r, s|S_t, A_t) = p(r, s|H_t, A_t) \tag{2.7}$$

Typically, the agent state S_t is compression of H_t as the full observation and agent state is Markov, but they are too large to compute. Similarly the full history H_t is Markov but keeps growing. Usually, we have a partially observable environment, for example, a poker game, where part of the game is hidden from the agent. This is called a partially observable Markov decision process and it is a common occurrence. In this case the agent state S_t is a function of the history, equation (2.8), where u is a state update function.

$$S_{t+1} = u(S_t, A_t, R_{t+1}, O_{t+1}) \tag{2.8}$$

Update function gives us compression of the history and observation, which describes state S_t with reasonable size. For example, if we have a maze as in Figure

2.3. With red squares representing the possible placement of the agent and the goal being the green square. Let us assume the agent is able to inspect its surroundings two squares up, down, left, right creating his S_t. In this case, all these states are the same. So these 3 states are not Markov. So to solve this we can update the function with buffer, which solves this issue. The moment when the agent reaches the top row, his state becomes Markov (and the buffer can be emptied) as he either reached the goal or can determine where he is located because the positions are unique.

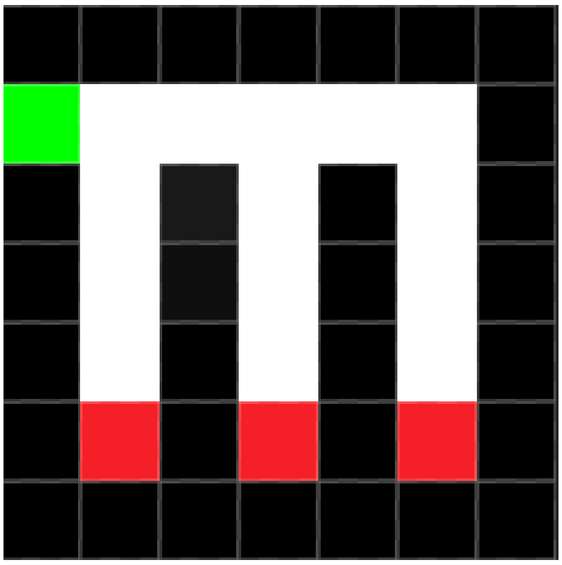


Figure 2.3: Example maze

2.3.2 Policy function

With state S_t of the agent, it is time to describe the policy. Let's dive deeper into the agent and policy function. Policy function conventionally notes as π. There are two types of policy functions, a deterministic policy that maps states to actions shown in equation (2.9) and stochastic policy which is more common and useful, shown in equation (2.10).

$$A = \pi(S) \tag{2.9}$$

$$v_\pi(A|S) = p(A|S) \tag{2.10}$$

2.3.3 Value function

The next component which needs to be explained is the value function. The actual value function is the expected return, shown in equation (2.11). Greek letter $\gamma \,\epsilon\, [0, 1]$ introduces discount factor. The discount factor helps us with the importance of immediate vs long-term rewards. So when $\gamma = 0$ we only care about immediate rewards, if $\gamma = 1$ we care about all rewards, and all are equally important. The best way is between these two extremes to tackle the problem you are solving. The value depends on a policy and we want to optimize it. Value function can be used to evaluate the desirability of states and also can be used to select between actions.

$$\begin{aligned} A(A|S) &= \mathbb{E}[G_t|S_t = s, \pi] \\ &= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...|S_t = s, \pi] \end{aligned} \tag{2.11}$$

Expected return has a recursive from $G_t = R_{t+1} + \gamma G_{t+1}$ and so does the value equation (2.12). Here a $a \sim \pi$ means action chosen by policy π in state s.

$$\begin{aligned} v_\pi(s) &= \mathbb{E}[R_{t+1} + \gamma G_{t+1}|S_t = s, A_t \sim \pi(s)] \\ &= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}|S_t = s, A_t \sim \pi(s)] \end{aligned} \tag{2.12}$$

This is known as Bellman equation [5]. A similar equation hold for the optimal value (2.13). This equation does not depend on policy. Thus is heavily exploited and used to create algorithms.

$$v_*(s) = max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1})|S_t = s, A_t = a] \tag{2.13}$$

Agent approximate value functions, with accurate value function agent, can behave optimally. And with suitable approximation, the agent can behave well, even in big domains.

2.3.4 Model

The last part inside the agent is a model. The model will not be covered in much depth as it is an optional part of the agent and is unused in my work. A model predict what the environment will do next E.g. predict next state $\mathcal{P}$ or next immediate reward $\mathcal{R}$ both examples shown in equation (2.14).

$$\begin{aligned} \mathcal{P}(s, a, s') &\approx p(S_{t+1} = s'|S_t = s, A_t = a) \\ \mathcal{R}(s, a) &\approx \mathbb{E}[R_{t+1}|S_t = s, A_t = a] \end{aligned} \tag{2.14}$$

2.4 Agent categories

This terminology is widely used in academic journals so it is good to know the differences between agents. Value-Based agent has an explicit Value Function, but the Policy Function is based on Value Function. Policy-Based agent has an explicit policy Function but no Value Function. Actor-Critic agent has both Value and Policy functions, actor corresponds to the policy part and acts and the Critic evaluates the policy and updates it based on the Value function and helps to select better policies over time. Model Free agent has one or both Value and Policy functions but does not have a dynamics model. This term is often used in reinforcement learning, but it is not a great division. Model-Based agent has a dynamics model and can have one or both Value function and Policy function. Summary is presented in Table 2.2

Agent category	Policy function	Value function	Model
Value-Based	implicit	✓	✗
Policy-Based	✓	✗	✗
Actor-Critic	✓	✓	✗
Model-Free	optional	optional	✗
Model-Based	optional	optional	✓

Table 2.2: Agent categories

2.5 Exploration and exploitation

This chapter addresses one of the problems of RL. That is how to select the next action a for the learning agent, whether to select one with the best rewards so far and exploit it or to select a sub-optimal strategy and explore action-space [6]. This is one of the differences from supervised learning where data is not in our control, whereas in RL data is under control in some sense. The control is indirect thru actions as data are sequential, it is not possible to just pick some data as it is in clustering, for example. Some of the algorithms are listed below and will be further explained.

- Greedy
- ϵ-greedy
- Policy gradients

2.5.1 Formalising the problem

For simplicity of introduction lets assume following:

- environment: has no state
- actions do not have long-term effect
- other observations can be ignored

Now we defined the multi-armed bandit. This term comes from historical reasons regarding slot machines in Las Vegas as in Figure 2.4 where can be seen, a one-armed bandit.

Figure 2.4: One armed bandit [7]

We can imagine a multi-armed bandit as a set of such slot machines or as a slot machine with multiple levers. Such multi-armed bandit has the following:

- Set of distributions $\mathcal{R}_a | a \epsilon \mathcal{A}$
- $\mathcal{A}$ is a set of actions ("arms"/levers)
- $\mathcal{R}_a$ is distribution on rewards, given action a
- At each time-step t agent selects and action $A_t \, \epsilon \, \mathcal{A}$
- The environment generates a reward $R_t \sim \mathcal{R}_{A_t}$
- The goal is to maximise cumulative reward $\sum_{i=1}^{t} R_i$
- We achieve this by learning policy: a distribution of $\mathcal{A}$

The useful concept is to denote action value for action a as the expected reward equation (2.15). The optimal value is maximization of q overall actions, equation (2.16). Another useful concept is regret, which is equation (2.17). Regret describes how good or bad we are doing. If regret is zero or close to zero we are doing well. The problem is we do not know v_*, it is useful in this scenario where it depicts how much regret does algorithm gain over the learning episode. Therefore maximizing cumulative reward $\equiv$ minimizing total regret.

$$q(a) = \mathbb{E}[R_t | A_t = a] \tag{2.15}$$

$$v_* = max_{a \, \epsilon \, \mathcal{A}} q(a) = max_a \mathbb{E}[R_t | A_t = a] \tag{2.16}$$

$$\Delta = v_* - q(a) \tag{2.17}$$

Greedy and ϵ-greedy use action value estimates $Q_t \equiv q(a)$. All necessary values will be introduced in following section 2.5.2 .

2.5.2 Greedy algorithm

Firstly, let us introduce action value. Action value is expected reward for action a equation (2.15). One simple estimate of function value can be average of the sampled rewards equation (2.18), where $\mathcal{I}(\cdot)$ is indicator function, whether action a was selected.

$$Q_t(a) = \frac{\sum_{n=1}^{t} \mathcal{I}(A_n = a)R_n}{\sum_{n=1}^{t} \mathcal{I}(A_n = a)} \tag{2.18}$$

This can be also calculated incrementally as remembering all time-steps and actions selected is not optimal equation (2.19), where we update the only value of selected action, update learning rate α and increment counter N_t. Learning rate α can be constant, which leads to tracking if the problem is not stationary, or could be dynamic as is and leads to averaging.

$$\begin{aligned} Q_t(A_t) &= Q_{t-1}(A_t) + \alpha_t(R_t - Q_{t-1}(A_t)) \\ \forall a \neq A_t : Q_t(a) &= Q_{t-1} \\ \alpha_t &= \frac{1}{N_t(A_t)} \\ N_t(A_t) &= N_{t-1}(A_t) + 1 \end{aligned} \tag{2.19}$$

Now onto the Greedy policy. It is one of the simplest policies. Action is selected according to equation (2.20). It is a simple action with the highest value of all actions.

$$A_t = argmax\, Q_t(a) \tag{2.20}$$

This algorithm prioritizes exploitation above exploration as is shown in example reward Table 2.3. This is an example where two action a and b are available and rewards are either 1 or 0. Assume that $P(+1|a) = 0.8$ and $P(+1|b) = 0.2$. And if $Q(a) = Q(b)$ then the action is chosen randomly. Now in Table 2.3 are 4 time steps.

time-step	1	2	3	4
a	0	-	-	-
b	-	1	0	0

Table 2.3: Example reward table

After these observations $Q_4(a) = 1/3$ and $Q_4(b) = 0$, so action b will never be selected again and regret keeps linearly rising as greedy algorithm stuck with non-optimal action $\Delta_a = 0$ and $\Delta_b = 0.6$.

2.5.3 ϵ-greedy algorithm

As was shown Greedy algorithm can get stuck on sub-optimal action forever with linear expected total regret. ϵ-greedy algorithm addresses this issue.

$$\pi_t(a) = \begin{cases} (1-\epsilon) + \epsilon/|\mathcal{A}| & \text{if } Q_t = max_b Q_t(b) \\ \epsilon/|\mathcal{A}| & \text{otherwise} \end{cases} \tag{2.21}$$

ϵ-greedy algorithm selects optimal action according to observation with probability $(1-\epsilon)$ and selects other action with probability ϵ shown with policy in equation (2.21). This enables more exploration but ϵ-greedy algorithm still has linear expected regret.

2.5.4 Policy gradients

Lets sidestep action values for a moment. Can we learn policies $\pi(a)$ directly, instead of learning values. For instance, define action preferences $H_t(a)$ and a policy in equation (2.22) called softmax. Softmax is a well defined policy as it normalizes, so all probabilities are positive and sum to 1.

$$\pi_t(a) = \frac{e^{H_t(a)}}{\sum_b e^{H_t(b)}} \tag{2.22}$$

The preferences are not values, they are just learnable policy parameters. The goal is to learn by optimizing the preferences. The idea behind that is to update policy parameters such that expected values increase. Gradient ascent is ideal for that. In the bandit case it is equation (2.23) where θ are the current policy parameters.

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta \mathbb{E}[R_t|\pi_{\theta_t}] \tag{2.23}$$

The issue here is that we cannot compute the gradient of the expected value, because we do not know the expected value. But there is a trick that lets us create a stochastic sample. This trick is called REINFORCE trick introduced in this paper [8] and is shown in equation (2.24).

$$\begin{aligned}
\nabla_\theta \mathbb{E}[R_t|\pi_{\theta_t}] &= \nabla_\theta \sum_a \pi_\theta(a)\mathbb{E}[R_t|A_t = a] \\
&= \sum_a q(a)\nabla_\theta \pi_\theta(a) \\
&= \sum_a q(a)\frac{\pi_\theta(a)}{\pi_\theta(a)}\nabla_\theta \pi_\theta(a) \\
&= \sum_a \pi_\theta(a) q(a) \frac{\nabla_\theta \pi_\theta(a)}{\pi_\theta(a)} \\
&= \mathbb{E}\left[R_t \frac{\nabla_\theta \pi_\theta(A_t)}{\pi_\theta(A_t)}\right] \quad = \mathbb{E}[R_t \nabla_\theta log \pi_\theta(A_t)]
\end{aligned} \tag{2.24}$$

The trick is straightforward firstly we expand the equation, secondly substitute $\mathbb{E}[R_t|A_t = a]$ per equation (2.15). The next step is multiplication by 1, then restructuring the equation. The last step is according to the chain rule. The whole equation (2.24) can be condensed into (2.25), and now we can sample this and create a stochastic gradient ascent on the value of the policy, equation (2.26).

$$\nabla_\theta \mathbb{E}[R_t|\pi_{\theta_t}] = \mathbb{E}[R_t \nabla_\theta log \pi_\theta(A_t)] \tag{2.25}$$

$$\theta = \theta + \alpha R_t \nabla_\theta log \pi_\theta(A_t) \tag{2.26}$$

With stochastic gradient ascent and sampled rewards it is possible to define soft-max policy, equation (2.27) or can be defined case by case in equation (2.28). Cases in equation (2.28) represent firstly, the preference for the selected value will increase, secondly, the preference for all other actions will decrease.

$$\begin{aligned} H_{t+1}(a) &= H_t(a) + \alpha R_t \frac{\partial log \pi_t(A_t}{\partial H_t(a)} \\ &= H_t(a) + \alpha R_t(\mathcal{I}(a = A_t) - \_t(a)) \end{aligned} \tag{2.27}$$

$$\begin{aligned} H_{t+1}(A_t) &= H_t(A_t) + \alpha R_t(1 - \pi_t(A_t)) \\ H_{t+1}(A_t) &= H_t(a) - \alpha R_t \pi_t(a) \quad if \;\; a \neq A_t \end{aligned} \tag{2.28}$$

There is one more useful thing to define and that is baseline subtraction. This will not affect the expected update but will change the variance. This updates defined equation (2.26) to new (2.29) where b represents baseline. However b must not depend on θ or a, but can depend on state S_t.

$$\theta = \theta + \alpha(R_t - b) \nabla_\theta log \pi_\theta(A_t) \tag{2.29}$$

Chapter 3

Experiments

With insight on RL from Chapter 2, lets move to the experiments. The experiments are divided into simulation and real-world application. In the simulation part, I will introduce the simulation environment setup and the results of the experiments. In the real-world application, I will firstly, briefly describe the arena, secondly describe image processing and lastly show experiment results.

3.1 Setup of the environment

Most research papers and articles use the MuJoCo [9] environment, as it is an advanced physics simulator. I was, unfortunately, unable to work in this environment due to insufficient experience and bad documentation. The second choice was ROS [10] in combination with Gazebo [11]. Gazebo comes with a complete toolbox of development libraries to make simulation easy. The gazebo also has a big community, great documentation, and is open source. The Robot Operating System (ROS), is a set of software libraries and tools that help you build robot applications, same as Gazebo, ROS is open source. These tools are suitable for the task of creating robotic table football in combination with Python.

3.1.1 3D Model

To train this model it is crucial to make a 3D model first. The size of the arena and players is relevant to real table football. Basically a cutoff of real table football. Model can be seen in Figure 3.1.

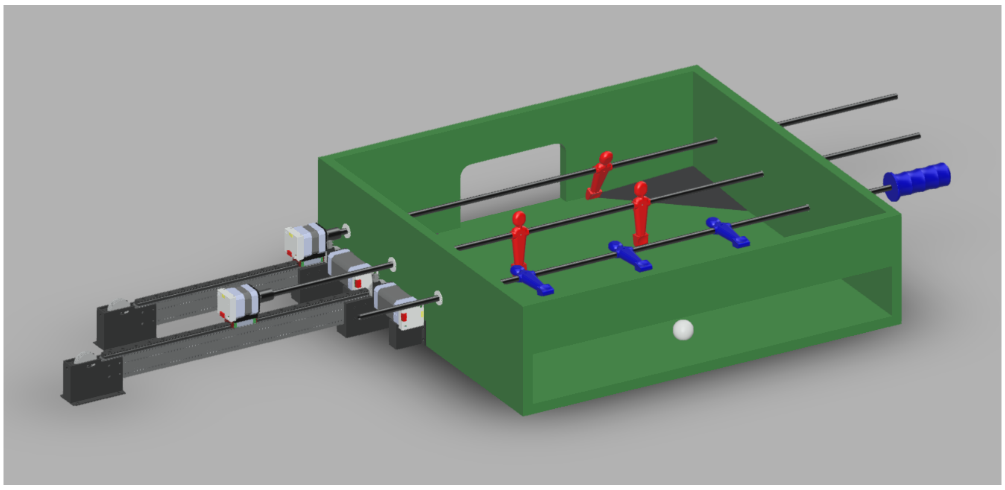


Figure 3.1: 3D model of table football

This model is too detailed, for simulation in ROS it had to be simplified. Simplification is shown in Figure 3.2.

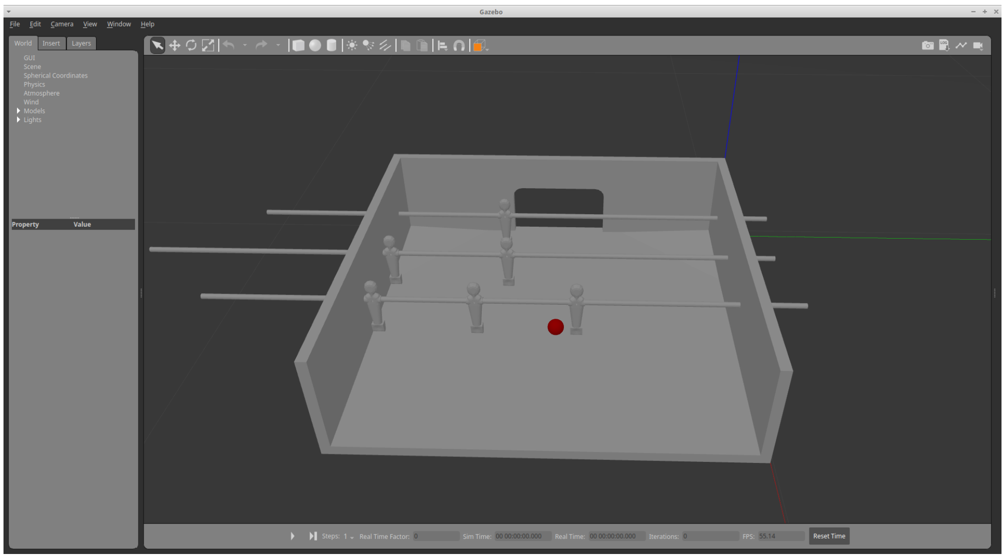


Figure 3.2: Simplified 3D model for ROS

Most of the parameters, such as weight were calculated by Autodesk Fusion 360 [12] after specifying the material, but parameters like friction and controllers had to be specified for the simulation to run smoothly. The gazebo is best suited for real-time applications, which means simulation will usually run at 0.8-1.5x times the real-time. This can be partly solved with the usage of Simulation time, where update frequency does not depend on real-time but on computer hardware capabilities. All experiments were run locally on my personal computer for optimization and to prevent any errors. An effort was made to run experiments on Metacentrum, which has much higher computing power and is suited for such tasks. To run experiments on Metacentrum, a Docker image had to be made. Docker is the name for open source software that aims to provide a unified interface for isolating container applications. More about Docker is available at their documentation [13]. Unfortunately, it was not possible to run experiments on Metacentrum due to numerous problems.

3.2 Tested algorithms

With the simulation environment running, it is possible to define the attacker's input. Two sets of shots were defined, a validation set, and a training set. The training set contains direct shots, pull shots, and a few misses as it is common that human attacker sometimes misses. The validation set has the same types of shots as the training set without the misses. Due to controller settings in Gazebo, the shots were not always the same, but as it turned out it was not problematic but rather useful as it introduced some random noise to the set. With the shot set defined, it was possible to start testing RL algorithms.

3.2.1 State-of-the-art algorithms

An overview of the current state-of-the-art algorithms is presented in Stable Baselines [14]. It is a library with good documentation and GitHub repository [15]. These algorithms are great as a starting point and can be further customized. Some of the algorithms available include the following.

- Augmented random search (ARS) [16]
- Actor critic (A2C) [17]
- Deep deterministic policy gradient (DDPG) [18]
- Deep Q learning (DQN) [19]
- Soft actor critic (SAC) [20]
- Proximal policy optimalization (PPO) [21]

These algorithms work in Gym interface [22]. Gym provides unified framework for RL and more general AI training. Thus it was necessary to wrap the Gazebo simulation in Gym environment for successful training. Communication with Gazebo was realized with internal rosservices and self-made reference links. Unfortunately, the internal rosservices do not provide a symmetric coordination system, it was necessary to create linear transformation to have data symmetric around 0.

3.2.2 Selecting of suitable algorithms

The experiments require a reward function. The first version of the reward function was a negative reward for player movement and receiving a goal. It is best shown in pseudo-code 3.1.

```
def calcualte_reward(ball_pos, player_pos):
    reward = 0
    if player_pos.defender +- epsilon == ball_pos:
        reward += 1

    action_penalty = how much players moved from last position
    reward -= action_penalty

    if ball_pos == goal:
        reward -= 100
        new_game()

while True:
    [ball_pos, player_pos] = GetDataRosModel()
    reward = calculate_reward()
    [goalie_pos, defender_pos] = RLmodel.predict([ball_pos,
   player_pos])
    SendDataToRos([goalie_pos, defender_pos])
```
Listing 3.1: Negative reward function pseudo code

The first experiment was a simple over-fitting of the model on one shot as a proof of concept that the training algorithm and overall setup is working. The defenders had a disabled rotation for faster convergence of results. This led to the selection of desired algorithms. So from mentioned algorithms in section 3.2.1 that seem to be good starting point only A2C and PPO were suitable. And algorithm of choice was PPO, as it was less fragile with the introduction of clipping the gradient update [21]. Other algorithms turned out to not learn efficiently or they are not compatible with the input and output formats that are desired. The input data format is continuous values attackers pose and rotation $(-1 : 1)$ and ball coordinates $(-1, 1)$. The output format is discrete values $0 - 99$ for defenders and the same for the goalie. The progression of the reward function in over-fitting is in Figure 3.3, the episode is composed of either 64-time steps or shorter if the ball leaves the arena. The success rate of defending was 80%. Since this was initial test of convergence the validation set was not tested. The success rate is only 80% because the shot was noisy, due to controllers as was mentioned. All figures are smoothed with running average of 20 samples.

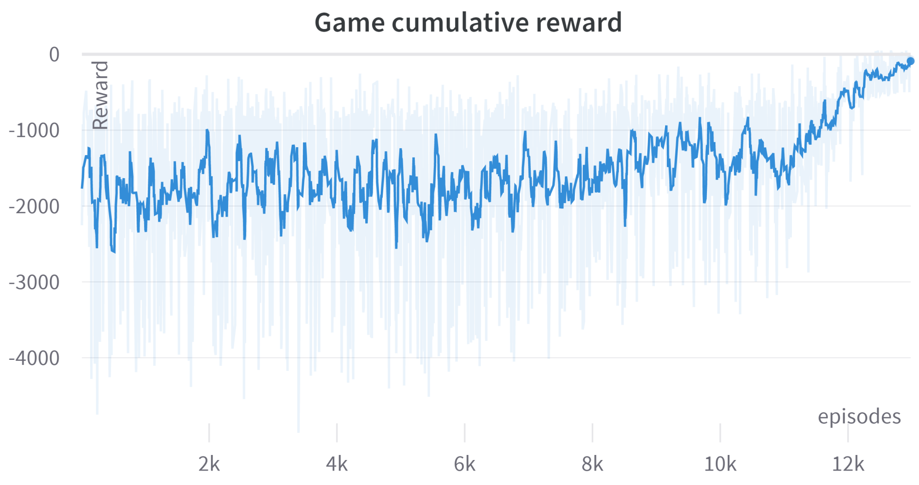


Figure 3.3: Cumulative reward of over-fit model on one shot (PPO model)

3.2.3 Experimenting with reward function

Further experiments with negative reward function, full training set, and disabled rotation, did not yield sufficient results as the model became stationary and was able to defend a portion of shots and discarded reaction to the rest of the set. And the progression of reward function in Figure 3.4 and 3.3 are similar.

Metric	Data
Success rate training set	23.3 %
Success rate validation set	46.6 %

Table 3.1: Results with negative reward function

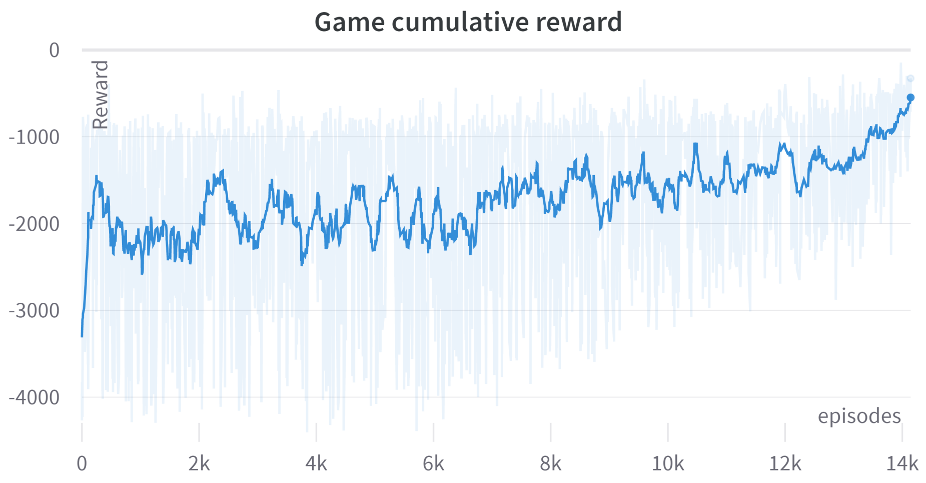


Figure 3.4: Cumulative reward on training data set with negative reward function

The setup of learning is shown in pseudo-code 3.1, the reward function will be further described similarly with only the depiction of the reward function. The second version of the reward function 3.2 was meant to balance positive and negative rewards such that the reward would reflect on the fact, whether the defense was successful and managed to either keep the ball in the field or shot it behind the attack.

```
def calcualte_reward(ball_pos, player_pos):
    reward = 0

    if player_pos.defender +- epsilon == ball_pos:
        reward += 10
    if player_pos.goalie +- epsilon == ball_pos:
        reward += 10

    action_penalty = how much players moved from last position
    reward -= action_penalty

```

```python
    if ball_pos == goal:
        reward -= 200
        new_game()

    if ball_pos == in_field:
        reward += 100
        pass

    if ball_pos == out_of_filed:
        reward += 100
        new_game()
```

Listing 3.2: Second version of reward function

Metric	Data
Success rate training set	33.4 %
Success rate validation set	49.3 %

Table 3.2: Results with augmented reward function

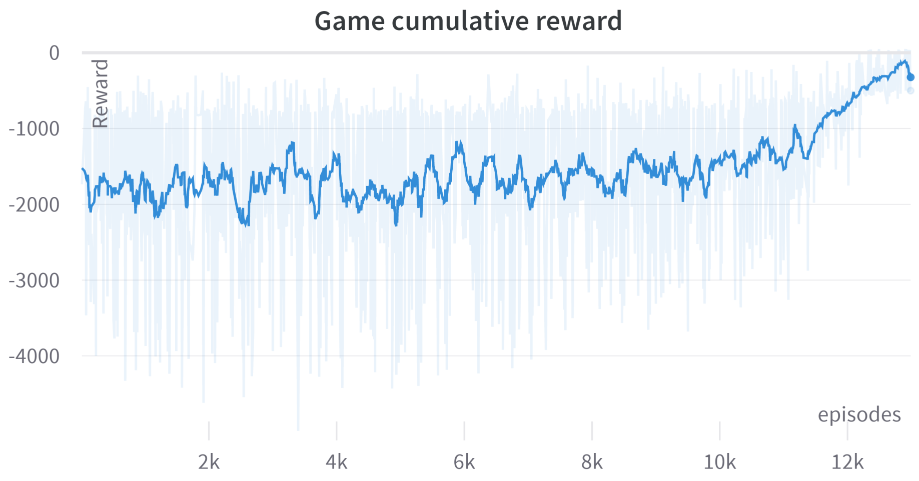


Figure 3.5: Cumulative reward of model on training data set with augmented reward function

This change did not solve the issue of players becoming stationary. This is apparent from Figure 3.5, where the trend shows over-fitting. Success rate is evaluated in Table 3.2. After the analysis of the reward function, the rewards remained mostly negative and it is apparent that the players became stationary because the gradient policy was descending. In chapter 2.5.4 is this algorithm shown and because of the negative reward, the algorithm was gradient descent instead of gradient ascent. The agent after exploring action space chose to exploit the most beneficial position and avoided additional negative rewards with a stationary position. This led to maximizing overall reward. The cumulative reward was similar to the on in Figure 3.3. The next step was to change the reward function to be positive most of the time. The biggest negative impact on the reward function was the action penalty and it was removed. The new reward function is shown in code snippet 3.3. The reward function in 3.3 also presents a new positive reward for goalie and defenders to be close to each other, this lets the players cover more of the goal line.

```
def calcualte_reward(ball_pos, player_pos):
    reward = 0

    if player_pos.defender +- epsilon == ball_pos:
        reward += 10
    if player_pos.goalie +- epsilon == ball_pos:
        reward += 10

    if player_pos.defender +- epsilon == player_pos.goalie:
        # reward for covering area close together
        reward +=10

    if ball_pos == goal:
        reward -= 500
        new_game()

    if ball_pos == in_field:
        reward += 200
        pass

    if ball_pos == out_of_filed:
        reward += 500
        new_game()
```

Listing 3.3: Third version of reward function

The third version of the reward function proved to be more suitable as the rewards were changing from positive to negative, this can be seen in Figures 3.6 and 3.7. In case of a goal, the overall reward was negative otherwise it was positive. This led to altering between gradient ascend and descend. Gradient ascend set preferences for chosen action to be higher and for others to be lower. Gradient descent set preferences for chosen action to be lower and for other actions to be higher. This led to the incorporation of rotation into the model. Up to this experiment, the rotation was disabled.

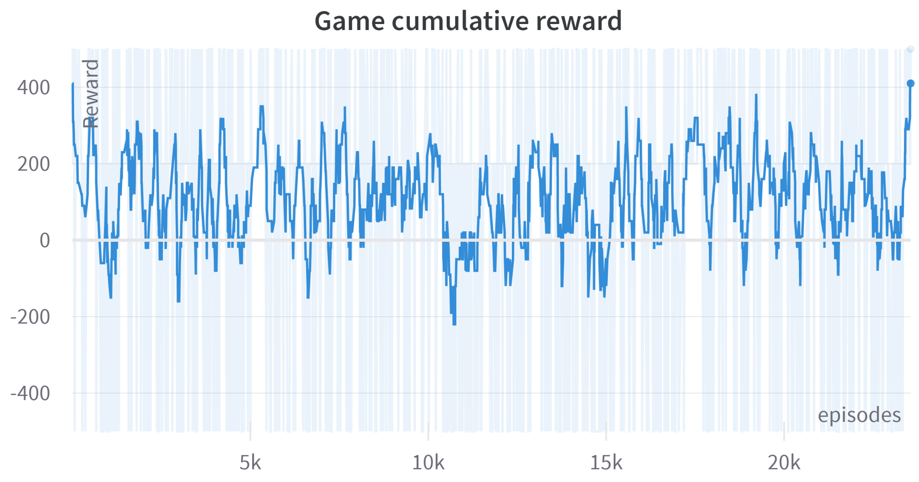


Figure 3.6: PPO model with positive reward after 2.5 milion time steps

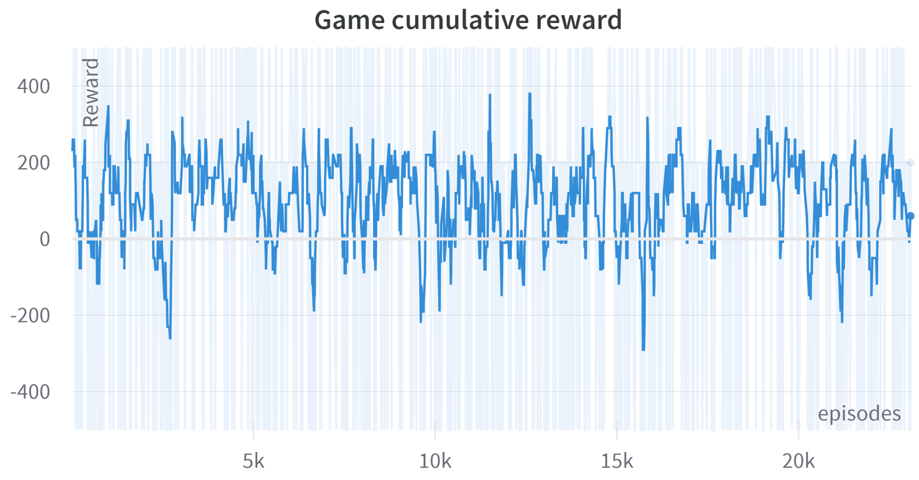


Figure 3.7: PPO model with positive reward after 4.5 milion time steps

With rotation introduction, the model performed worse as the state space that the agent had to explore became bigger. The change in reward function this time was to reward the position of players only if they had low rotation and would be still able to catch the ball. The drop in reward function can be seen in Figure 3.8. Other metrics can be seen in Table 3.3. Due to these results the rotation had to be discarded.

```
def calcualte_reward(ball_pos, player_pos):
    reward = 0

    if player_pos.defender +- epsilon ==  and rotation.defender ==
    +-0.1 : #rotation in rad
        reward += 10
    if player_pos.goalie +- epsilon == ball_pos and rotation.
   goalie == +-0.1 : #rotation in rad:=
        reward += 10

    if player_pos.defender +- epsilon == player_pos.goalie:
        # reward for covering area close together
        reward +=10

    if ball_pos == goal:
        reward -= 500
        new_game()

    if ball_pos == in_field:
        reward += 200
        pass

    if ball_pos == out_of_filed:
        reward += 500
        new_game()
```
Listing 3.4: Reward function with enabled rotation

Metric	Data
Success rate training set	35.2 %
Success rate validation set	30.3 %

Table 3.3: Results on model with implemented rotation

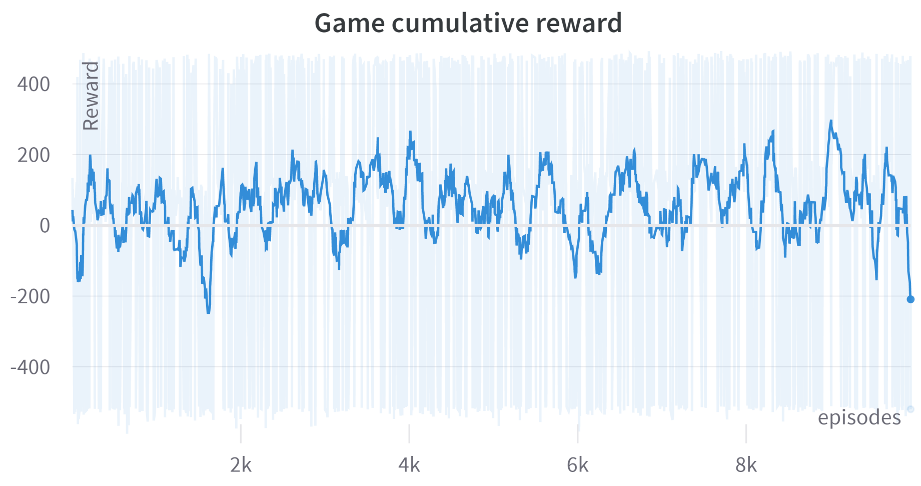


Figure 3.8: Cumulative reward of model with enabled rotation

3.2.4 Model with best performance

The best performing model is with disabled rotation and reward function from code snippet 3.3. The evaluation of the model will be based on reward function, training, and validation set. And will be compared to another reference than static players, because results from Table 3.1 show that the results are not sufficient. The comparison will be with Random action chosen.

From Figures 3.6 and 3.7 it looks like the model is not learning however overall average value across all episodes is slowly rising as is shown in Table 3.4. This is a reflection of fact that RL models are sometimes trained over up to one billion time steps. Evaluation of validation and training set will be the success rate of defending and can be seen in Table 3.5 on the training set and in Table 3.6 on the validation set. Model trained for 500K steps performed only marginally better on training data set and worse than random on validation data set. This is most likely caused by exploration of action space and effort to find the optimal actions. However the model trained for 4.5M steps was performing better on both data sets. Reasons for Random action having such performance are described in Chapter 5.3. Performance in the real arena will be evaluated using the success rate of defending as well.

Model	Average reward per episode
Model after 500k time steps	51
Model after 1M time steps	65
Model after 1.5M time steps	95
Model after 2.5M time steps	105
Model after 3.5M time steps	112
Model after 4.5M time steps	124

Table 3.4: Average reward over time steps

Shot	Success rate 500k steps	Success rate 4.5M steps	Random action
Pass shot 1	77 %	84 %	88 %
Pass shot 2	89 %	86 %	83 %
Pass shot 3	96 %	100 %	93 %
Pull shot 1	83 %	93 %	71 %
Pull shot 2	96 %	97 %	80 %
Pull shot 3	97 %	96 %	83 %
Pull shot 4	97 %	98 %	81 %
Straight shot 1	34 %	83 %	39 %
Straight shot 2	36 %	61 %	50 %
Straight shot 3	40 %	77 %	36 %
Straight shot 4	40 %	85 %	38 %
Straight shot 5	62 %	70 %	62 %
Straight shot 6	42 %	47 %	38 %
Diagonal shot 1	36 %	29 %	30 %
Diagonal shot 2	61 %	76 %	54 %
Miss 1	97 %	100 %	95 %
Miss 2	97 %	98 %	95 %
Overall Success rate	69.4 %	81.2 %	65.6 %

Table 3.5: Success rate of defending on training data set

Shot	Success rate 500k steps	Success rate 4.5M steps	Random action
Straight shot val 1	91 %	73 %	67 %
Straight shot val 2	7 %	62 %	41 %
Straight shot val 3	0 %	15 %	25 %
Straight shot val 4	37 %	55 %	39 %
Straight shot val 5	91 %	81 %	79 %
Pull shot 1	43 %	68 %	61 %
Pull shot 2	77 %	74 %	75 %
Overall Success rate	49.4 %	61.2 %	55.3 %

Table 3.6: Success rate of defending on validation data set

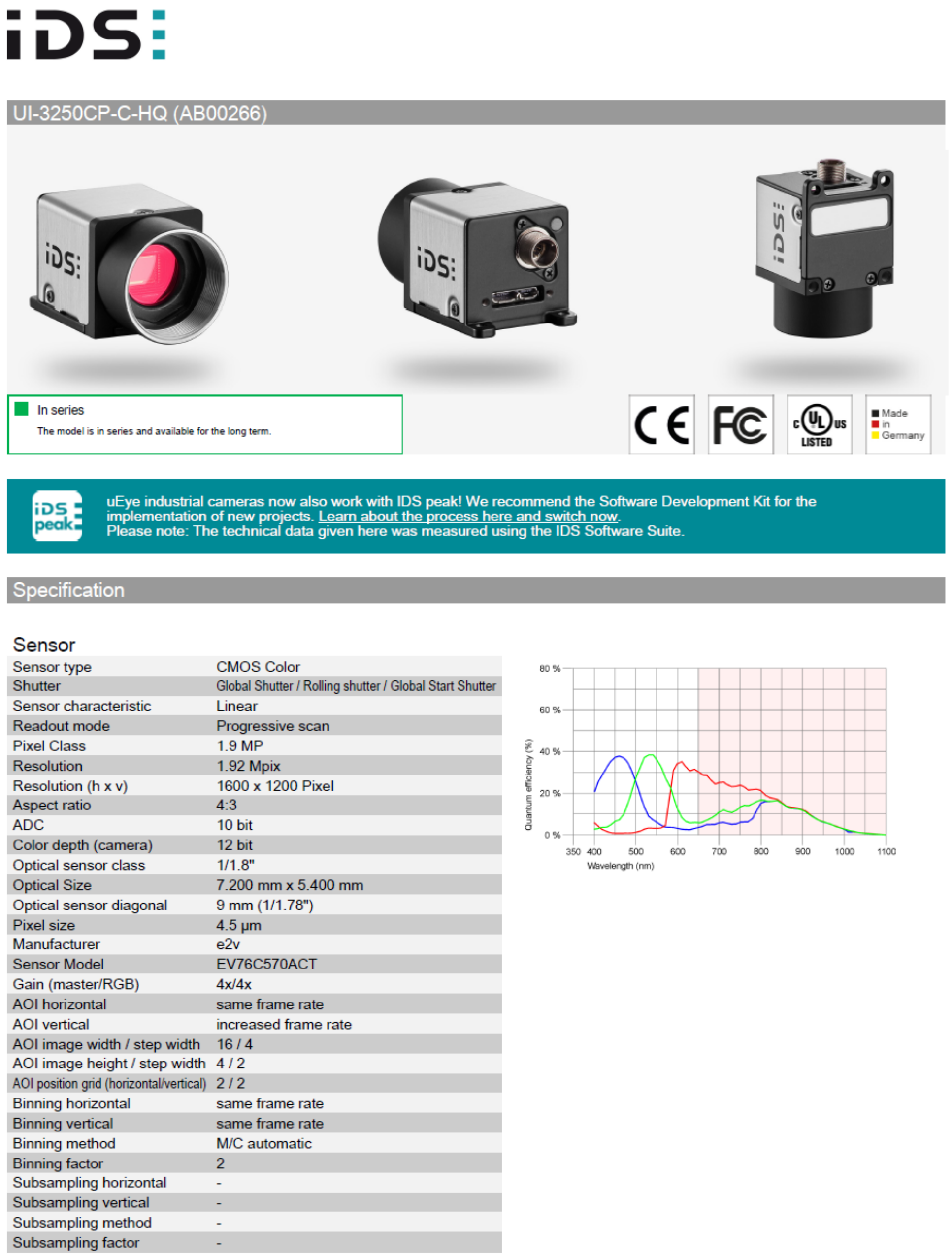
Chapter 4

Realisation of the Arena

In this chapter, I will elaborate on the software and hardware. Hardware parts for my work are a camera and a partly industrial computer. The hardware also includes arena assembly and electric actuators. The arena assembly and electric actuators setup and tuning is in my colleague's work [23]. Regarding software, the Python flow chart of the program will be shown, how calibration was realized and how fast was the whole program. As well as results for random actions, trained models and expert system.

4.1 Camera

The camera chosen for this task is an industrial camera from IDS [24]. This camera was paired with suitable lens. To operate this camera IDS driver and pyuey bindings for uEye API [25] had to be installed. I also installed the uEyeCockpit, which is software from IDS that provides GUI for setup, saving camera settings to file, and initial experiments with the camera. The camera has the following specifications shown in Figure 4.1. All listed software from IDS can be downloaded from [24], but registration is mandatory.



UI-3250CP-C-HQ (AB00266)

🟩 In series

The model is in series and available for the long term.

Made in Germany

uEye industrial cameras now also work with IDS peak! We recommend the Software Development Kit for the implementation of new projects. Learn about the process here and switch now.
Please note: The technical data given here was measured using the IDS Software Suite.

Specification

Sensor

Sensor type	CMOS Color
Shutter	Global Shutter / Rolling shutter / Global Start Shutter
Sensor characteristic	Linear
Readout mode	Progressive scan
Pixel Class	1.9 MP
Resolution	1.92 Mpix
Resolution (h x v)	1600 x 1200 Pixel
Aspect ratio	4:3
ADC	10 bit
Color depth (camera)	12 bit
Optical sensor class	1/1.8"
Optical Size	7.200 mm x 5.400 mm
Optical sensor diagonal	9 mm (1/1.78")
Pixel size	4.5 µm
Manufacturer	e2v
Sensor Model	EV76C570ACT
Gain (master/RGB)	4x/4x
AOI horizontal	same frame rate
AOI vertical	increased frame rate
AOI image width / step width	16 / 4
AOI image height / step width	4 / 2
AOI position grid (horizontal/vertical)	2 / 2
Binning horizontal	same frame rate
Binning vertical	same frame rate
Binning method	M/C automatic
Binning factor	2
Subsampling horizontal	-
Subsampling vertical	-
Subsampling method	-
Subsampling factor	-

Figure 4.1: Camera data sheet from vendor [26]

I had to change the parameters from factory settings to suit this task. The most influential parameters are presented in Table 4.1.

Parameter	Set Value
Width	800 px
Height	600 px
Pixelclock	128 Hz
Framerate	50 fps
Exposure	20 ms
Long exposure	ON
Colormode	ON
GainBoost	0

Table 4.1: Used camera settings

The lowering of resolution from 1600x1200 px to 800x600 px enabled for higher FPS and faster pattern matching. The connection to this camera is a USB 3.0 micro-B that allows for high transfer speeds and volumes of data. To mount this camera, a frame was built from aluminum profiles and a holder had to be 3D printed because the camera does not have any standard mount. The model of the holder can be seen in Figure 4.2. Final product is in Figure 4.3. The image taken with the camera with my settings is in Figure 4.4

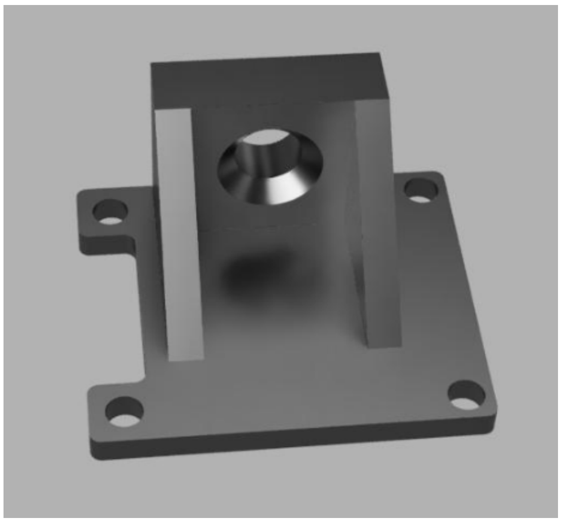


Figure 4.2: Holder for camera

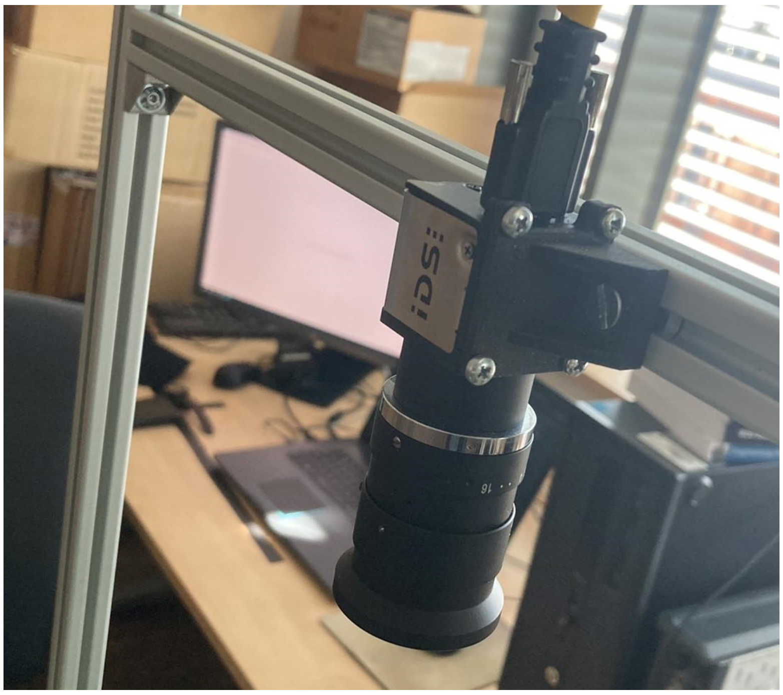


Figure 4.3: Aluminium profile frame with mounted camera

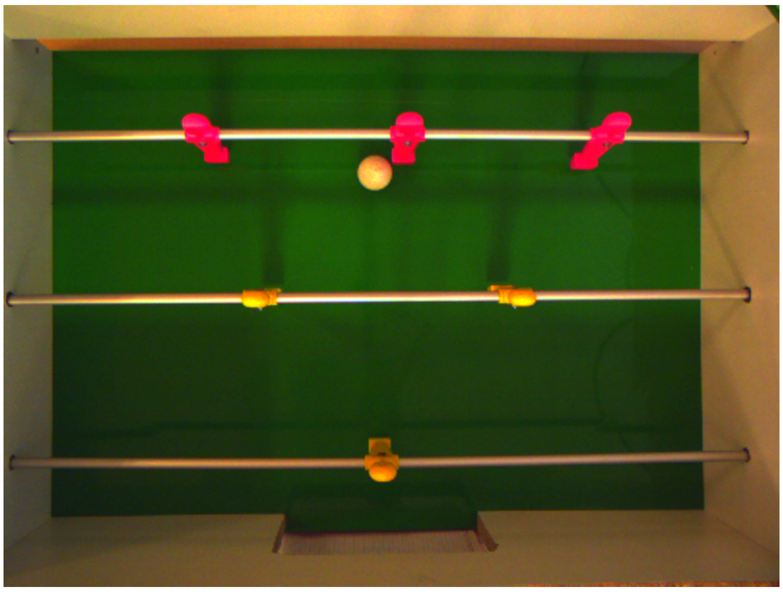


Figure 4.4: Image of arena taken with IDS camera with my settings

4.1.1 Image processing

This section is dedicated to image processing and this topic was core in my Bachelor's Thesis [1]. So only the used algorithms will be mentioned. Image processing was involved in several tasks, such as defense and goalie rotation calibration, tracking of ball, and tracking of attacking players.

Rotation calibration

Calibration of players is realized while players rotate at stable velocity, with frame cropping and HSV thresholding. Firstly frame is cropped on specified coordinates to have only the defense or goalie player in the frame, secondly, a HSV thresholding is applied. HSV color space can be seen in Figure 4.5. HSV color space is more convenient for thresholding.

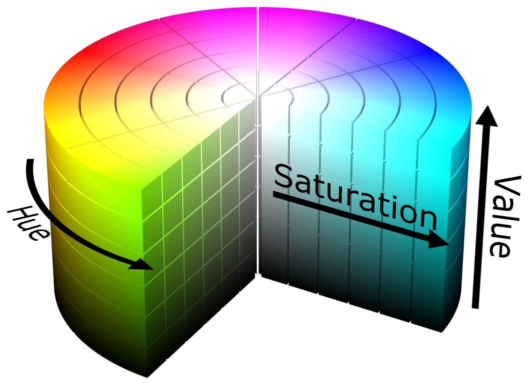


Figure 4.5: HSV color space [27]

The mask after thresholding is summed over the white pixels resulting in size of calibrated player, examples of different rotations can be seen in Figure 4.6.

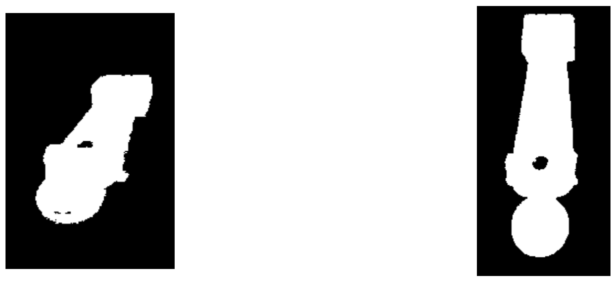


Figure 4.6: Examples of thresholding

The sum of masked players has a maximum and minimum depending on rotation. This fact was used to send information to the industrial computer that players have reached their minimum. The minimum was angle with a stable offset from 0° that was compensated. The calibration began with a collection of samples over a fixed time to evaluate the minimum, afterwards the signal was sent to stop rotation and end calibration. Graphs from both calibration are in Figures 4.7 and 4.8.

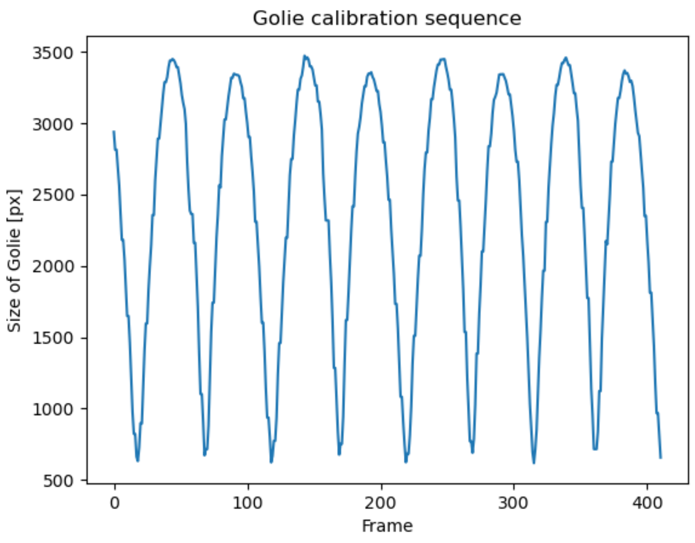


Figure 4.7: Calibration of goalie

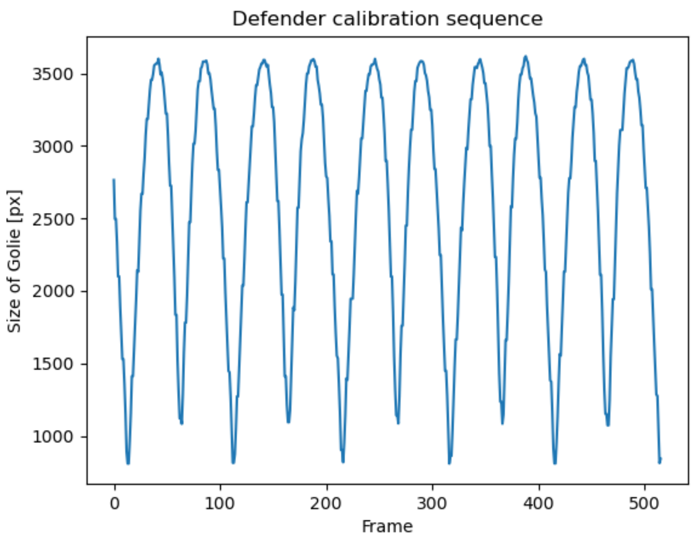


Figure 4.8: Calibration of defender

The same principle of thresholding is applied to the pose estimation of attack players. Further improved with binary image moments returning the centroid of the blob. The centroid of the blob gives us information about the linear and rotational position. Cropped attacking row can be seen in Figure 4.9. Cropping the whole row yields good results in rotation estimation however it is noisy in linear motion. For linear motion, only a thin strip around the player's bar was cropped, solving the noise issue. Graph of linear tracking can be seen in Figure 4.10 where players were moved by hand from one side to another and roughly to the center. Graph of rotation tracking is in Figure 4.11. Both rotation and linear motion of attack players are normalized to range $(-1, 1)$.

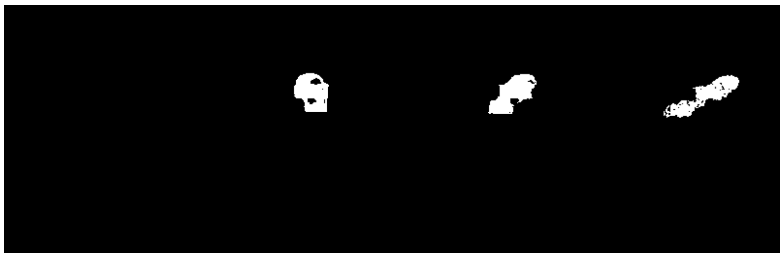


Figure 4.9: Cropped image of attack players

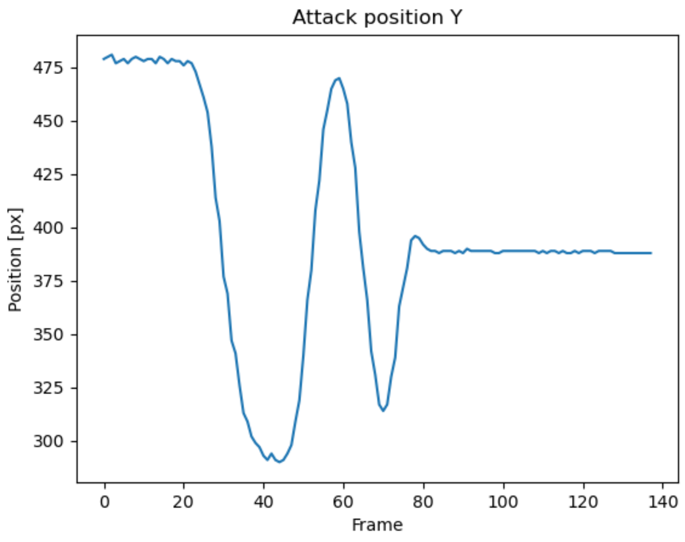


Figure 4.10: Example of tracking linear motion for attack players

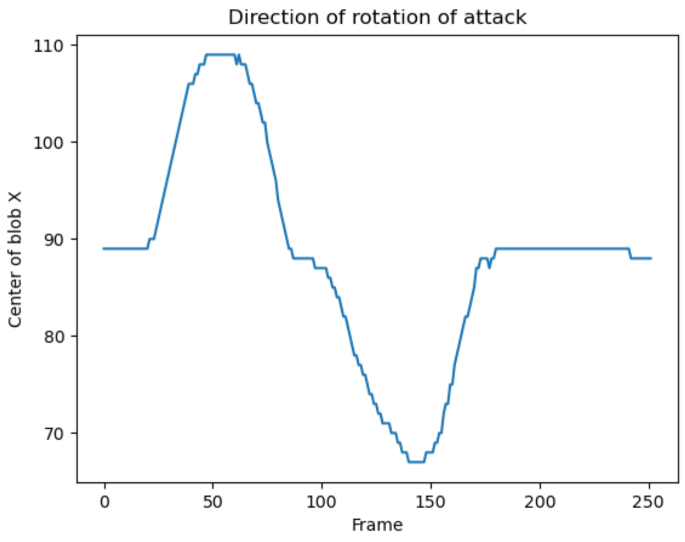


Figure 4.11: Example of tracking rotation for attack players

Ball tracking

For ball tracking, pattern matching was used. Pattern matching is a method for the localization of smaller templates in an image. This method is based on evaluating all pixels and locating the best match. Output is also a matrix with values representing a match for the center pixel (a higher value represents a better match). The output matrix is usually thresholded for noise cancellation. In my case, there is only one ball in the arena so the location of the ball is coordinates of max value. In documentation [28] are various criteria for match evaluation. The best one for my application is TM-CCOEF-NORMED (Template Matching Correlation Coefficient Normed) as its range of match is limited between 0-1 (0 is no match and 1 is an absolute match). The frame from the camera was cropped because the white walls of the arena were interfering, cropped image is in Figure 4.12. This method can find the ball correctly 97% of the time and with ideal illumination even higher. The speed of the whole setup from 1000 frames processed is on average 0.0235s.

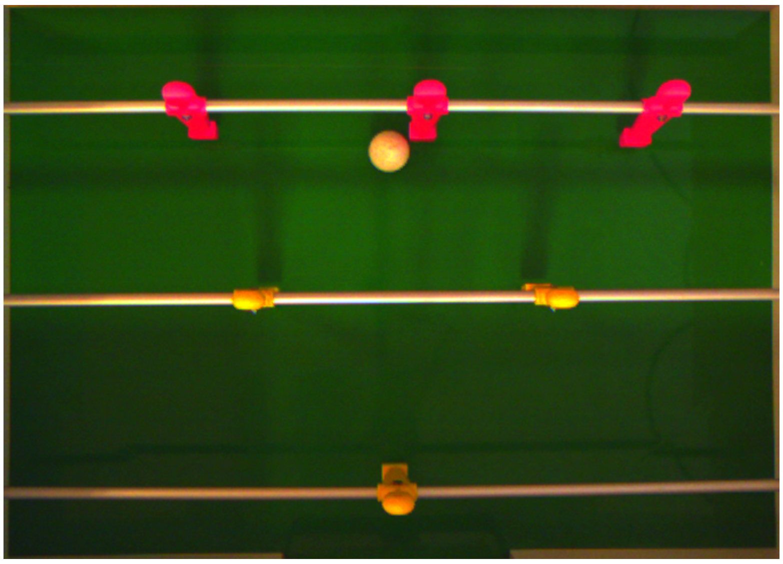


Figure 4.12: Cropped image for pattern matching of ball

4.2 Software setup

The initial idea was to export RL model as well as image processing onto the industrial computer. And the Python script would run in REXYGEN software [29]. This was not possible as the camera is not working with Linux. The industrial computer uses a Linux Debian 10 and Python in version 3.7.5. Because of this, the resolution was to let the image processing and RL model run on my computer and to apply network protocol to send data over to the industrial computer. The protocol of choice was Modbus TCP/IP, as it is implemented in both REXYGEN and Python and is easy to configure. This modification required the creation of ports, that will be transferring messages between my computer and the industrial computer. The created ports are presented in Table 4.2.

Port	Write	Port	Read
0	Linear Setpoint Defender	10	Start Game (start of main loop)
1	Rotation Setpoint Defender	11	Start Calibration Defender
2	Linear Setpoint Goalie	12	Start Calibration Goalie
3	Rotation Setpoint Goalie	13	Error state
4	Initialization of Defender DONE		
5	Initialization of Goalie DONE		
6	Start motors		
7	Defender switch regulators		
8	Goalie switch regulators		

Table 4.2: Modbus port choice

With specified states, it was time to design the program. Best summary is flow chart in Figure 4.13. Camera initialization consists of loading a .ini file specifying fps, resolution, and other adjustable parameters for the camera. The start of motors represents starting REXYGEN and simultaneously it starts linear calibration of defense and goalie. Calibration is done by reaching the end switch. Rotational calibration is done via camera as is described in 4.1.1 and lastly, the start game as the name suggests starts the game.

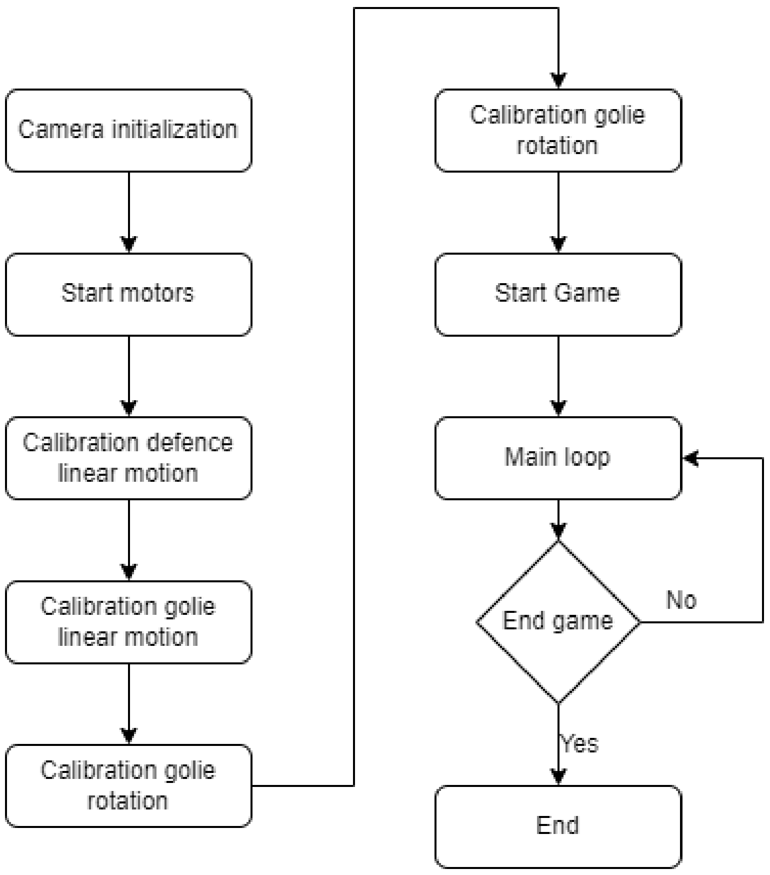

Camera initialization
Start motors
Calibration defence linear motion
Calibration golie linear motion
Calibration golie rotation
Calibration golie rotation
Start Game
Main loop
End game
No
Yes
End

Figure 4.13: Flow chart of Python code

4.3 Performance on real arena

The performance in the real arena was measured by the success rate of defending as is mentioned in 3.2.4. Performance was measured on Random output, Model output, and Expert system output for comparison. The static player baseline was not tested. The reason for that is the shots are not strictly defined but created by the player, which aims for open space. In this scenario, static players would fail. The Expert system was created on my Bachelor thesis baseline [1], with the use of Kalman Filter [30]. The Kalman filter was set up by my colleague Martin Jandík and is described in his Master Thesis [23]. Evaluation is based on 50 shots from slow straight shots to pull shots to capture the whole spectrum of shots and utilize all players. Results are presented in Tables 4.3 and 4.4.

Model	Success rate of defense
Random	28%
RL model	62%
Expert system	70%

Table 4.3: Success rate on real arena with no rotation

Model	Success rate of defense
Random	30%
RL model	40%
Expert system	66%

Table 4.4: Success rate on real arena with rotation

The success rate on RL and the Expert system is lower with rotation enabled. This is due to the occasional own goal by the defending players. The evaluation was meant to be across more than 50 shots, unfortunately, one of the motors stopped working.

Chapter 5

Discussion

The objectives of the thesis consisted of four subtasks:

1. to research current reinforcement learning algorithms
2. to develop a baseline method and further improve it
3. to use the model in real-world application
4. to evaluate obtained results both qualitatively and quantitatively

5.1 Baseline method and improvements

The research started with RL introduction and formalisation in Chapter 2. Later a list of state-of-the-art algorithms were presented in Section 3.2. Of these algorithms, only PPO algorithm was suitable and had measurable improvement. Developing a baseline method required the specification of the reward function. It also included the creation of training data set and validation data set, which were later used for model performance evaluation. The baseline method was a negative reward function presented in pseudo-code 3.1. This baseline method proved successful after over-fitting to one shot. Further experimenting with the reward function in Section 3.2.3 proved the relevance of the theory. The final version of the reward function was designed with respect to gradient ascend policy characteristics.

5.2 Deployment of the model on the real arena

For deployment, it was necessary to extract all the information the model requires from the camera. While deploying the model to the real arena two major problems occurred. The first one was the fact that electric actuators did not work the way the manufacturer specified. The second was the incompatibility of the camera with the industrial computer and the incompatibility of models trained on different versions of Python. These problems were solved with network communication of my computer and an industrial computer. The protocol used for this was Modbus TCP/IP.

5.3 Summary of results

Incremental improvements were seen, but a breakthrough did not occur. This is mostly due to my lack of experience with reinforcement learning. Another obvious limitation is the need to create a data set of shots. Despite all this, the final model performed quite well. The model also showed characteristics that were incorporated into the reward function, such as defending close to each other and staying in front of the ball, this is a qualitative observation. Quantitative results in Table 5.1, are success rates for real arena and simulation. Even though in simulation the model looks like it is quite close to random action on the validation set, in the real arena it performed much better. This is possible since simulation actuators are faster, and thus the random action could perform so well while the players were flying from one side to other. This fast movement was not possible on the real arena, where trained model had the advantage of tracking the ball. The next possible explanation is that the validation set is fixed and cannot aim for open space, whereas in the real arena the aim was always open space. The initial thought was to use the same regulators in simulation and the real arena. This was not possible due to the late arrival of electric actuators.

Model	Success rate of defense in arena	Success rate of defense in model
Random	28 %	60.5 %
RL model	62%	71,2 %

Table 5.1: Comparison of success rates

Chapter 6

Conclusion

This thesis is about reinforcement learning and its applicability to real-world application. In our case the simplified mechatronic table football The purpose was to explore the potential of reinforcement learning and show the qualities of this approach.

We selected a suitable algorithm, defined a data set, and trained the model with multiple reward functions. We found out that reinforcement learning requires long training periods and can produce sufficient results.

With proper hardware setup, we were able to use it in a real-world application, while retaining the results learned in simulation.

To sum up the thesis, it is a good insight into reinforcement learning and its possibilities. The property of retaining the results gained in simulation is promising for real-world applications. I think we have opened new questions, and further experiments with this approach will be necessary, to fully unlock the reinforcement learning potential.

6.1 Future work

Some of the ideas and improvements for the future work are listed below.

- Further experiment with reward functions.
- Automatic shot creation.
- Tune the Docker image to run on cloud servers.
- Change the simulation to include camera and make end to end system.
- Create a symmetrical game, to enable in silico RL agents to play against one another. Eliminating the need to create a data set of shots.

Bibliography

[1] Matěj Sieber. *Detekce pohybu míčku pro mechatronický model stolního fotbalu.* ZČU, 2020.

[2] Deepmind. *Alpha Go home page.* URL: https://www.deepmind.com/research/highlighted-research/alphago (visited on 04/10/2022).

[3] Deepmind. *Alpha Go protein folding.* URL: https://www.deepmind.com/blog/alphafold-a-solution-to-a-50-year-old-grand-challenge-in-biology (visited on 04/10/2022).

[4] Deepmind. *Alpha Go MuZero.* URL: https://www.deepmind.com/blog/muzeros-first-step-from-research-into-the-real-world (visited on 04/10/2022).

[5] R.E. Bellman. *Dynamic Programming. Dover.* Springer US, 1957. ISBN: 978-0-387-30164-8.

[6] Tor Lattimore and Csaba Szepesvári. *Bandit Algorithms.* Cambridge University Press, 2020. DOI: 10.1017/9781108571401.

[7] None. *The slot machine picture.* URL: https://creativecommons.org/licenses/by-sa/3.0/ (visited on 04/10/2022).

[8] R. J. Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine Learning* 8 (1992), pp. 229–256.

[9] MuJoCo. *MuJoCo homepage.* URL: https://mujoco.org/ (visited on 01/10/2022).

[10] Robot Operating System. *ROS homepage.* URL: https://www.ros.org/ (visited on 02/10/2022).

[11] Gazebo. *Gazebo simulator homepage.* URL: https://gazebosim.org/home (visited on 02/10/2022).

[12] Autodesk. *Autodesk Fusion 360.* URL: https://www.autodesk.cz/products/fusion-360/ (visited on 04/10/2022).

[13] Docker. *Docker documentation.* URL: https://docs.docker.com/ (visited on 05/11/2022).

[14] Antonin Raffin et al. "Stable-Baselines3: Reliable Reinforcement Learning Implementations". In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.

[15] Stable baselines 3. *Stable baselines 3 GitHub repository.* URL: https://github.com/DLR-RM/stable-baselines3 (visited on 05/10/2022).

[16] Mingyang Geng et al. “Learning data augmentation policies using augmented random search”. In: *CoRR* abs/1811.04768 (2018). arXiv: 1811.04768. URL: http://arxiv.org/abs/1811.04768.

[17] Volodymyr Mnih et al. “Asynchronous Methods for Deep Reinforcement Learning”. In: *CoRR* abs/1602.01783 (2016). arXiv: 1602.01783. URL: http://arxiv.org/abs/1602.01783.

[18] OpenAI. “Deep Deterministic Policy Gradient algorithm.” In: (). URL: https://spinningup.openai.com/en/latest/algorithms/ddpg.html (visited on 05/10/2022).

[19] Volodymyr Mnih et al. “Playing Atari with Deep Reinforcement Learning”. In: *CoRR* abs/1312.5602 (2013). arXiv: 1312.5602. URL: http://arxiv.org/abs/1312.5602.

[20] OpenAI. *Soft actor critic algorithm.* URL: https://spinningup.openai.com/en/latest/algorithms/sac.html (visited on 04/10/2022).

[21] John Schulman et al. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).

[22] OpenAI. *OpenAI Gym.* URL: https://gym.openai.com/envs/#algorithmic (visited on 05/10/2022).

[23] Martin Jandík. *Návrh manipulátorů pro řízení autonomního pohybu hráčů stolního fotbalu.* ZČU, 2022.

[24] IDS. *IDS vendor official site.* URL: https://www.ids-imaging.us/ (visited on 05/11/2022).

[25] IDS. *Python bindings for uEye API.* URL: https://pypi.org/project/pyueye/ (visited on 03/10/2022).

[26] IDS. *UI-3250CP-C-HQ camera data sheet.* URL: https://en.ids-imaging.com/download-details/AB00266.html (visited on 05/11/2022).

[27] None. *The HSV color model mapped to a cylinder.* URL: https://creativecommons.org/licenses/by-sa/3.0/ (visited on 04/10/2022).

[28] *Dokumentace Open CV.* 2019. URL: https://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/template_matching/template_matching.html (visited on 04/27/2020).

[29] REXYGEN. *Official REXYGEN site.* URL: https://www.rexygen.com/ (visited on 05/12/2022).

[30] R. E. Kalman. *A New Approach to Linear Filtering and Prediction Problems.* Transaction of the SME-Journal of Basic Engineering, 1960.