BACHELOR THESIS

Active Vision for Humanoid Soccer Robots using Reinforcement Learning

Department of Informatics
MIN Faculty
Universität Hamburg

Florian André Vahl

florian.vahl@uni-hamburg.de
B. Sc. Informatik
Student ID number: 7031630

Thesis Advisors: Dr. Mikko Lauri
Marc Bestmann

Abstract

In this thesis, an active vision approach is trained using reinforcement learning to control the head of a humanoid robot in the RoboCup Humanoid League domain. To train the approach a lightweight simulation is constructed to simulate a partially observable Markov decision process based on recorded high-level game data and the actions of the agent. The agent was trained using Proximal Policy Optimization and rewarded for a higher world model confidence. Different observation and action space configurations were evaluated to improve the performance of the policy. The best policy was compared against an existing entropy-based approach as well as the static scanning pattern currently used by the Hamburg Bit-Bots.

Zusammenfassung

In dieser Arbeit wird ein Algrithmus zum aktiven Sehen mit Hilfe von bestärkendem Lernen trainiert, um den Kopf eines humanoiden Roboters in der RoboCup Humanoid League zu steuern. Zum Trainieren des Ansatzes wurde eine leichtgewichtige Simulation erstellt, welche einen teilweise beobachtbaren Markov-Entscheidungsprozess, basierend auf aufgezeichneten Spieldaten und den Aktionen des Agenten, simuliert. Der Agent wurde mit Hilfe von Proximal Policy Optimization trainiert und für eine höhere Sicherheit im Weltmodell belohnt. Verschiedene Beobachtungs- und Aktionsraumkonfigurationen wurden evaluiert, um die Leistung des Agenten zu steigern. Die beste Strategie wurde mit einem bestehenden entropiebasierten Ansatz sowie mit der statischen Scanbewegung, welche derzeit von den Hamburg Bit-Bots verwendet wird, verglichen.

Contents

List of Figures

List of Tables

1 Introduction

In this thesis, a novel active vision approach to control the head of a humanoid soccer robot will be developed and evaluated. The goal is to maximize the certainty of the robot's world model using the orientation of the robot's head. For this purpose, a neural network policy is trained using reinforcement learning in a custom-built simulation.

Observing the environment around a mobile robot is crucial for its function. Many robots feature complex vision systems [Fie+19; SBB18; Bar+19; BWL20] for these types of measurements. Said systems may rely on a camera system that has a limited field of view and is therefore not able to observe the whole environment at once. The limitations can be technical or artificial as in the RoboCup humanoid soccer competitions [7; 6], where they are used to enforce a humanoid-like design. Technical limitations can be related to the available computational resources, as a full 360-degree video includes more information that needs to be processed. This can especially be an issue for embedded environments, where computational resources are limited. A full 360-degree video is also only suitable for close proximity observations as all magnification needs to be done digitally which is limited by the camera's sensor resolution as well as optical qualities of the lens. Using a lens with a larger focal length mitigates the issue by enabling observations across longer distances while keeping the resolution the same. But this comes at the cost of a smaller field of view, which needs to be positioned well to get the maximum information from the environment.

Many robots, especially humanoid ones like the Wolfgang robot platform [Bes+19] (see Section 2.7) which is used for this thesis, are capable of changing their viewpoint by orienting the camera towards interesting regions of their environment. This way of observing the environment is similar to the way humans visually observe the state of their environment by dynamically positioning their eyes and head to gain a better view. Often not only a single, but multiple features of the environment are being observed. Given the limits, the robot, therefore, needs to find a compromise between multiple often conflicting goals. This needs to be done to keep the world model up to date as well as possibly enable better high-level decisions. Formally, the problem can be formulated as a partially observable Markov decision process.

Current methods used to control the camera orientation often use static patterns consisting of a sequence of joint positions [1], conventional tracking methods to track a certain feature of interest [Vil+14] or entropy-based active vision approaches [Mah+19; Rui+07]. An emerging field of study includes the usage of reinforcement learning for

such tasks [KTR21]. In this case, a neural network policy is trained based on a reward which describes how well the policy performed a certain task. It can be used to reward a more accurate representation of the world state in the robot's world model caused by better observations.

The main research question in this thesis is how a reinforcement learning-based active vision approach performs in the RoboCup soccer domain and how its performance can be improved. To answer this question a novel reinforcement learning-based approach is introduced. It uses recorded high-level game data from different RoboCup competitions in a custom build simulation to mimic in-game scenarios. An agent is trained in these scenarios to perform the set of partial observations that leads to the highest world model confidence. PPO [Sch+17] is used as the reinforcement learning algorithm which optimizes the agent's policy. In the end, it is evaluated which observation and action space configurations, as well as neural network sizes, lead to the best performance. The best configuration is compared against an entropy-based approach as well as a simple static scanning pattern to answer our research question. The implementation of the simulation environment, as well as the training code and configurations, are publicly available on GitHub[1].

[1] `https://github.com/Flova/BA/`

2 Fundamentals

This Chapter introduces the basic concepts and approaches used later on in the thesis. The topics range from the field of active vision in Section 2.1 over a description of reinforcement learning in Section 2.2 to more domain-specific topics like ROS in Section 2.6 or the used robot model in Section 2.7.

2.1 Active Vision

Vision or more generally nearly every other form of perception can be performed not only as a passive observation but also as an active process. In this process, the parameters of the observation process, meaning e.g. the viewpoint or camera settings, are dynamically changed to gain better observations. For visual observations, this process is called *active vision*. It is a growing field of interest in the field of robotics [CLK11].

Ruzena Bajcsy describes it as an active process, which not only includes the passive analysis of input data but also the actions that help to gather this data by changing the state of the passive sensor. Her motivation is phrased with the quote "We don't just see, we look." [Baj88].

The field is also closely related to the medical and psychological analysis of similar behaviors in nature. Rapid eye movement between multiple points, known as saccades, reflects the priorities of the visual system, attention, and later memory of a scene [Rol15]. It is also shown that our viewpoint is largely dependent on the task and context [RBH07]. These findings show the importance of active vision-like behaviors in nature.

2.2 Reinforcement Learning

In reinforcement learning, an agent is trained using a reward to perform certain tasks in an environment. In contrast to supervised learning, no labeled data is used to train the model of the agent directly. Instead, the agent needs to interact with the environment and gather information in regards to which actions lead to the highest reward and modify its policy accordingly. The interaction between the agent and the environment can be seen in Figure 2.1. The environment provides the agent with a state and a reward in a

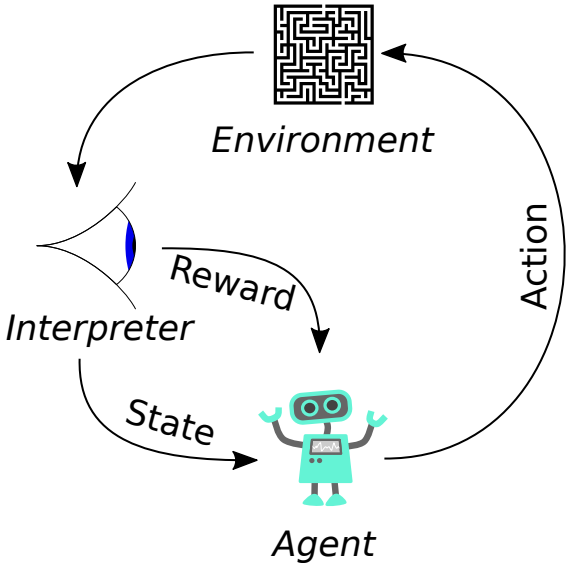

Figure 2.1: Illustration of reinforcement learning. Figure from [10].

given time step. The agent then needs to determine an appropriate action that is passed back into the environment. This influences the next state of the environment and the cycle repeats.

This process can be formally defined as a Markov Decision Process (MDP), which itself is defined as the 4-tuple (S, A, P, R) where S stands for the set of possible states, A declares a set of actions, P probabilistically maps the current state under respect to the action a to the new state, and R defines the immediate reward given for that transition. A policy π defines the agent's behavior, by probabilistically mapping states of the environment to actions that are executed in the said states [SB18]. It is optimized during training to maximize the reward given in this Markov Decision Process.

Many real-world reinforcement learning problems are not fully covered by the standard Markov Decision Process as only a partial state of the environment is observable. This is called a *partially observable Markov decision process* (POMDP). This generalization of the standard MDP extends the tuple in the definition with Ω which is a set of observations, and O which probabilistically maps a state of the environment and taken action together to an observation $o \in \Omega$ which is then provided to the agent. In many cases, the set of observations Ω as well as the set of actions A are not discrete, but continuous. They are then referred to as observation- and action-space respectively.

In reinforcement learning, we need to optimize not only for the current reward returned by the MDP in a given step but also for future rewards. For this, the discounted reward is used. It takes future rewards into account and is defined as

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.1}$$

where γ is the *discount rate*. It determines the present value of future rewards. It can be included in the value function $V_\pi(s)$ which represents the future or discounted reward for a given state s if the policy π is followed. This can be used to define the $Q_\pi(s, a)$ function which returns the value after the action a has been performed in state s when the policy π is used [SB18].

To determine how the an action a effects the future reward in a given state s, we need to calculate the advantage function $A_\pi(s, a)$. The function is defined as:

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s) \tag{2.2}$$

All of these functions may only be approximated, which would lead to them being noted as $\hat{A}$, $\hat{Q}$, and $\hat{V}$ respectively. These approximated functions, as well as the policy π, are often modeled using neural networks. This is called *deep reinforcement learning*.

To estimate the advantage function A the *generalized advantage estimation* [Sch+16] can be used. It compares the usually expected rewards after the state s modeled by the value function $V(s)$ and subtracts them from the combination of the reward given in the rollout after taking the action a and the value of the next state $V(s+1)$ after taking that action a. This estimation might have biases because realistically only $\hat{V}$ is available. One could also only compare the states $\hat{V}(s)$ with the sum of rewards given during the rollout after the given state, but this has a high variance because it is also influenced by other actions during the rollout. Therefore, the hyperparameter λ is introduced to give a tradeoff between both approaches. A hyperparameter named γ is used to balance the immediate and future rewards. The estimated advantage function can therefore be formally defined as:

$$\delta^{\hat{V}_t} = r_t + \gamma \hat{V}(s_t + 1) - \hat{V}(s_t) \tag{2.3}$$

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^{\hat{V}} \tag{2.4}$$

On-policy reinforcement learning algorithms collect all data used for the optimization of the policy using the newest policy. They do not include samples from previous rollouts in the rollout buffer, which would be called off-policy reinforcement learning. This means the rollout buffer is discarded each time a new version of the policy is used to collect the rollout buffer. The training is split into two alternating phases. In the first one, a rollout

buffer, containing information regarding actions, states, and rewards, is collected in the training environment. In the second one, the policy is optimized based on the data in the rollout buffer. The optimization is done in an iterative way where the rollout buffer is split into batches that are optimized sequentially. An epoch is finished when all batches of the rollout buffer are processed. The number of epochs, meaning the number of times each sample in the rollout buffer is used for the optimization, is a hyperparameter.

Model-free reinforcement learning algorithms have no explicit model of their environment and are not able to see how a given action affects the environment prior to executing the action. Model-based algorithms on the other hand include a model of their environment and are explicitly able to reason how their actions will affect the environment.

2.2.1 Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a model-free on-policy reinforcement learning algorithm [Sch+17]. It is used for deep reinforcement learning with the policy gradient method, where the policy is directly optimized using stochastic gradient descent (SGD) or other related gradient-based optimization algorithms.

An issue with on-policy algorithms is, that the optimization of the policy on the rollout buffer might lead to a new policy that fails to perform in the environment. This is fatal as all data for the rollout buffer is collected with this new policy, so the algorithm might not be able to recover. Proximal Policy Optimization tries to prevent this by limiting the changes applied to the parameters of the policy in each optimization phase. This is achieved by clipping the advantage-based objective function, prohibiting large policy updates.

The clipping advantage-based objective function is defined as

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{2.5}$$

where θ represents the parameters of the policy π and $\hat{\mathbb{E}}$ stands for the average over a finite set of batches. Generalized advantage estimation [Sch+16] is used to estimate the advantage function $\hat{A}_t$. The function r_t denotes a probability ratio, which compares the probability of choosing a given action in the given state between the new policy π_θ and the old policy $\pi_{\theta_{\text{old}}}$. It is defined as

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \tag{2.6}$$

where a and s stand for the action and the state respectively. Therefore, comparing the old policy to itself would result in $r_t(\theta_{\text{old}}) = 1$. If the new policy prefers the action a more than the old one in the given state s its r_t would be greater than 1. Similarly, it would be less than 1 if the action became less likely. We need this comparison between the current policy and the old policy because the old one was used to gather the rollout buffer, and the approximate advantage $\hat{A}$ is estimated based on the rollout buffer. This means that if the given action got more likely in the given state under the new policy, the advantage that was estimated for that action, should influence the policy optimization more and if it became less it should influence the optimization less. Even though PPO is an on-policy algorithm it happens during the optimization that the current policy and the old policy which was used to collect the rollout buffer differ. This is the case because the optimization was already executed on previous batches and that changed the policy parameters. We, therefore, need to take this difference into account.

Clipping is applied to the probability ratio r in equation 2.5 to keep the policy updates small and prevent hazardous large updates. The hyperparameter ϵ is introduced to define the clipping range. It is usually kept around 0.2. The *min* term is added to deactivate the clipping if the new policy has a much higher probability of selecting an action with a negative advantage than the old policy in the current state. This is not desirable and a larger optimization step might be necessary to prevent that divergence.

The final loss function does not only include the clipping advantage-based objective function $L^{\text{CLIP}}(\theta)$ but also a supervised mean squared error loss for the value function L^{VF} and an entropy bonus S which is used to encourage exploration. The value function can be trained in a supervised way because the value of a given state in the rollout is known by summing up all the following rewards. Weighting the components with the two hyperparameters c_1 and c_2 results in the following equation.

$$L(\theta) = \hat{\mathbb{E}}_t \left[L^{\text{CLIP}}(\theta) - c_1 L^{\text{VF}} + c_2 S_{[\pi_\theta]}(s_t) \right] \tag{2.7}$$

In reinforcement learning, an agent needs to explore its environment by not always following the most valuable action in a given state. To achieve this exploration, PPO samples the actions from a probability distribution during the rollout. Often a Gaussian distribution is used for this purpose, but other probability density functions forming e.g. a beta distribution are also possible. To do this, the policy not only predicts the action value itself but also a variance. Based on this output the action is sampled. This gives the policy some control over the amount of exploration done in a given state. During deployment the sampling is disabled and the action is used directly. To backpropagate through the sampling the reparameterization trick is applied [KW14]. This results in the sampling being formulated as

$$x = \mu + \sigma \cdot \text{sample}(N(0, 1)) \tag{2.8}$$

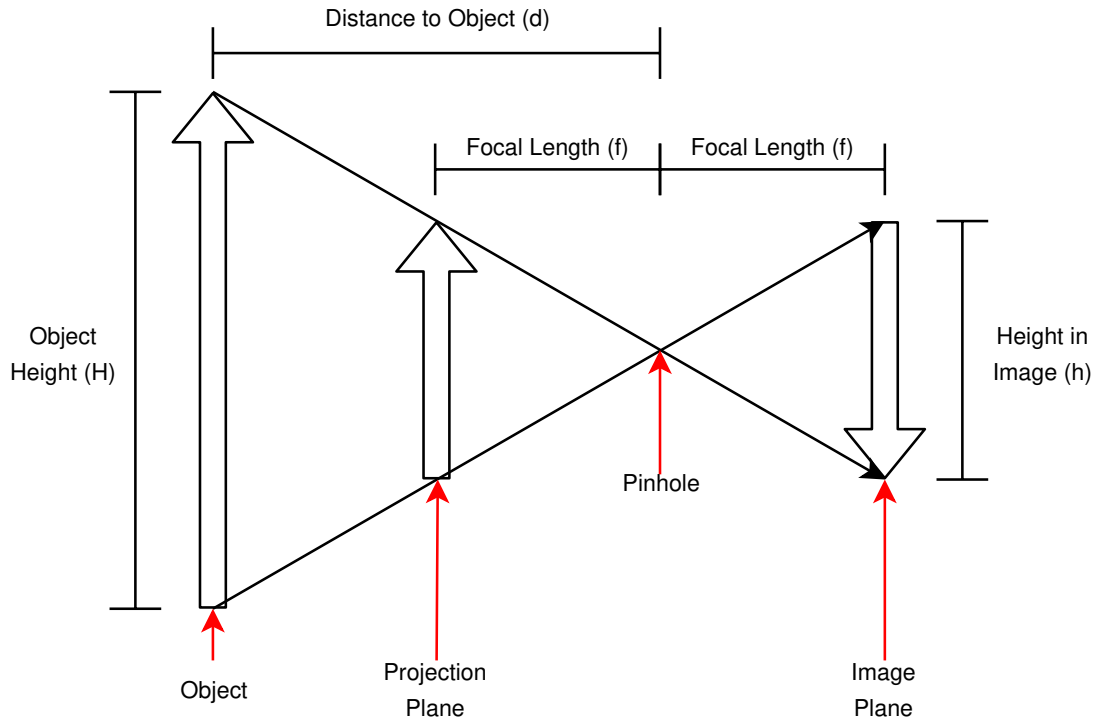

Figure 2.2: Projection of an object onto the projection- and image-plane when following the pinhole camera model. Figure from [Gül19].

2.3 Camera Model

The pinhole camera model can be used to describe how light from a 3D scene is projected on the 2D plane of pixels inside the camera. It is later on used to calculate the visibility of objects for our high-level simulator (see Section 4.1). The basic concept of an object being projected onto the image plane can be seen in Figure 2.2 for one image dimension.

The standard way of describing and estimating the intrinsic camera calibration has been introduced by Zhang et al. [Zha00]. The intrinsic model of the pinhole camera is described by the following camera matrix, where f_x, as well as f_y, represent the focal length scaled by resolution in x or y axis respectively and c_x and c_y represent the corresponding image centers.

$$K = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \tag{2.9}$$

To get the image pixel coordinates of a 3D point p we need to multiply the point with K. We also assume that the coordinates of p are given in the camera's frame of reference.

$$p_{img} = K \cdot p \tag{2.10}$$

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x \cdot f_x + z \cdot c_x \\ y \cdot f_y + z \cdot c_y \\ z \end{pmatrix} \tag{2.11}$$

We need to normalize u and v with w to get the image coordinates x_{img} and y_{img} from the results of the equation above.

$$x_{img} = \frac{u}{w} = \frac{u}{z} \tag{2.12}$$

$$y_{img} = \frac{v}{w} = \frac{v}{z} \tag{2.13}$$

Rounding this to integer values results in the pixel coordinates.

2.4 RoboCup

The RoboCup [Kit+97] [5] is an annual robotics competition held at different places around the world. It was first officially hosted by Nagoya, Japan in 1997 after a Pre-RoboCup was held at the International Conference on Intelligence Robotics and Systems in the previous year. Its main goal is to encourage robotics research in such a way that a team of humanoid robots is capable of winning a game of soccer against the current human world champions in 2050. In the years since its foundation, the competition also expanded in other areas by hosting competitions in different leagues that focus on e.g. industrial or rescue robotics. But the main focus is still on the domain of robot soccer, which is seen as a real-world benchmark that is also understandable and interesting for the public audience. The robot soccer part of the RoboCup is further split into different sub-competitions. The small and midsize league mainly focus on advanced strategies without the constraints of a humanoid robot design. There are also dedicated simulation leagues, that focus on the design of software without the power and computation constraints of a real-world robot. The humanoid and standard platform league both focus on humanoid robots. This means that the robot designs need a human-like layout with a single head, two arms, and two legs. Other constraints regulate the ratio of the body length to the head and leg lengths, the width, the height, the foot size, and the length of

Figure 2.3: Three robots competing in the Humanoid Kid Size League. The Hamburg Bit-Bots in red use their Wolfgang-OP robot while Team Rhoban in blue plays with their SigmaBan robot platform. The differences in regard to e.g. size and proportions can be clearly seen.

the arms. There are also constraints regarding the sensors that are integrated into the robots [6]. Only sensors that are comparable to human senses such as cameras, force-torque sensors, microphones, accelerometers, and gyroscopes are allowed, while other commonly used ones like e.g. lidars or compasses are prohibited. In addition to that, all robots need to be completely autonomous and all computation during the game needs to happen on the robot's hardware. While the standard platform league (SPL) uses the standardized NAO robot (shown in Figure 2.4) which was designed by SoftBank Robotics (previously Aldebaran Robotics), more experimental and team-designed robots are used in the humanoid league. This enables more research regarding the mechanical design of the robot. In addition to that, it enables the usage of more computationally intensive approaches due to a more flexible choice of computer hardware [Bes+21]. But this comes at the cost of less portable software approaches due to different hardware configurations. The league is further split into the Humanoid Kid Size and Adult Size Leagues, which group the different robot designs by their size. The Hamburg Bit-Bots team where this thesis is mainly focused on competes in the Humanoid Kid Size League [Bes+19]. A picture from a Humanoid Kid Size League game can be seen in Figure 2.3. During the COVID-19 pandemic, many leagues introduced ways of playing in simulation. More information about the simulation of games regarding the Humanoid League can be seen in Section 2.5.

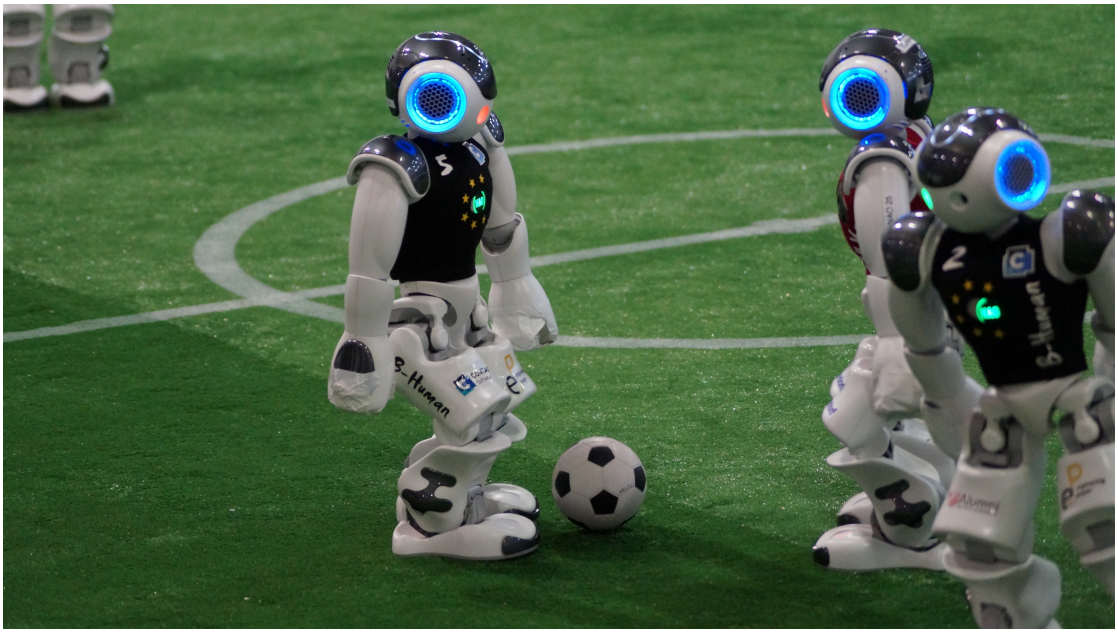

Figure 2.4: A Standard Platform League (SPL) game during the German Open 2019, which is a local RoboCup tournament.

The RoboCup domain includes many different research fields. They range from high-level behavior and strategy [Che+19] to low-level hardware control [BGZ19]. To compete successfully in a RoboCup soccer game, a robot needs a reliable motion component, that is capable of walking, kicking, and recovering from falls, which happen frequently in this rough environment. It also needs to sense and model the game's state including information regarding the position and dynamics of the robot itself as well as other players or the ball. Based on that information high-level tasks like path planning or the game strategy are executed. The robots also need to communicate with each other to succeed in this multi-agent environment.

To reach the goal of winning against the current world champion in 2050 [Bur+02] the rules are adapted and the difficulty of the competition is increased incrementally when certain milestones are reached. The milestones are based on metrics like kick distance, walking speed, or the number of goals.

2.5 Webots

The Webots robotics simulator was first introduced in 1996 [Mic04]. It is internally based on a fork of the ODE physics engine and allows physically detailed simulation of different kinds of robots as well as their environment.

Figure 2.5: Two robots competing in a simulated game during the RoboCup 2021.

Due to travel restrictions and the COVID-19 pandemic the RoboCup 2021 Humanoid League Kid Size League tournament (see Section 2.4) was completely simulated using the Webots simulator as well as a newly developed automated referee [7]. An example situation from a simulated game during the world cup can be seen in Figure 2.5. The number of robots in a game was reduced from four robots per team to three robots per team. This was done to reduce the computation needed to simulate a complete game with two half times of ten minutes each. The calculation was done in the cloud via AWS. Each robot ran in its own docker container on a different machine, communicating with the simulation server via the network. The teams needed to provide a robot model in the *.proto* format which describes the design of their real robot. This digital representation of the real robot includes not only the mechanical layout but also all sensor and actuator placements as well as their specifications. This was done to adapt the simulation as close as possible to the real robots and minimize the divergence between the code-base for the simulated and real robots.

Because the simulation speed with six robots in a single environment is significantly lower than real-time, the complete game state is logged for a later visualization after the simulation is completed. This visualization is then presented to the public and the teams. Besides that, the teams are able to log data regarding the internal state of each robot.

2.6 ROS

The Robot Operating System (ROS) is an open-source middleware that runs on top of an operating system like Ubuntu or Windows and provides a framework with different utilities to build software for robotics applications [Qui+09].

ROS includes tools for visualization and interaction with the robot as well as a message-passing system to connect different nodes, which the software is split into. In addition to that, default solutions for many standard tasks like forward kinematics, inverse kinematics, and path planning are provided. The targeted languages are mainly C++ and Python. Splitting the software into the nodes allows code to run concurrently even across the network on different computers. This is an advantage in contrast to monolithic software architectures. Debugging and testing are also simplified as not all nodes need to run at a given time to test a component. Messages are published in a predefined layout onto different communication channels called topics. A node can subscribe and publish to multiple topics. Standardization of the messages as well as keeping the nodes small helps with the interchangeability of code between different robot architectures, domains, laboratories, or companies. Besides the messages, there are also bidirectional communication interfaces called services and actions that are mainly used for remote function calls. They mainly differ in the backchannel, as the actions return a continuous progress indication in contrast to the single return message of a service.

The Hamburg Bit-Bots RoboCup team uses ROS since 2017 [BHW17], replacing the previous self-developed framework. Their software architecture inside the ROS ecosystem can be seen in Figure 2.6.

2.7 Robot Platform

The robot platform used in this thesis is the Wolfgang-OP [Bes+21]. It is designed for the RoboCup environment and features an industrial C-Mount camera from Basler (Basler acA2040-35gc) with a global shutter as well as an Intel NUC (NUC8i5BEK) and an Odroid-XU4 for processing. The processing capabilities are extended with an Intel Neural Compute Stick 2 Vision Processing Unit (NCS2 VPU) for efficient neural network inference. Its mechanical design features 20 degrees of freedom in its joints. The shoulder roll and neck pitch joints are equipped with elastic elements to avoid fall damage when the robot falls to the side. In addition to that, the knee joints are supported by torsion springs forming PEAs (Parallel Elastic Actuators [Ver+16]). The joints are powered by Dynamixel Servos [2] from the MX-64, MX-106, and XH540 series. Standing upright the robot's height measures 0.8m with a weight of 7.5kg. An image of the standing robot can be seen in Figure 2.7.

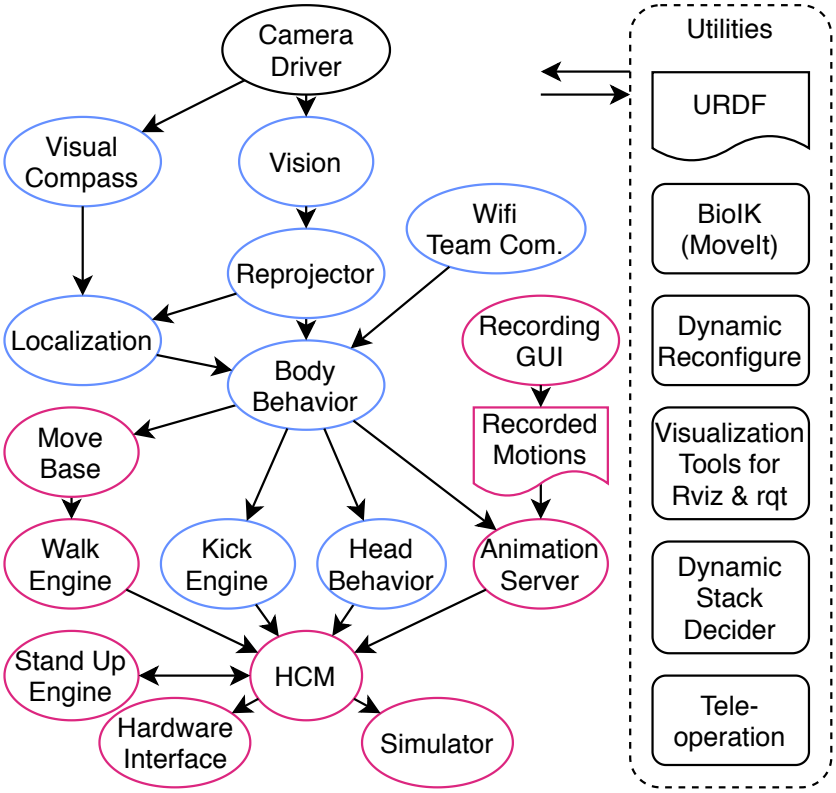

Figure 2.6: Simplified layout of the Hamburg Bit-Bots software stack. Only important nodes and their communication channels are shown. The vision system and the high-level behavior (blue) are specialized for the RoboCup domain, while parts of the motion system (red) are directly usable in other domains. Figure from [Bes+21].

Figure 2.7: A humanoid Wolfgang-OP robot from the Hamburg Bit-Bots. Image used with permission of the owner.

3 Related work

Previous work regarding active vision approaches from the RoboCup soccer domain is presented in Section 3.1. Both entropy and reinforcement learning-based techniques are covered. In addition to that, active vision from other domains which also use reinforcement learning will be presented in Section 3.2.

3.1 Active Vision in the RoboCup Soccer Domain

Active vision has been researched before in the field of humanoid soccer robots. First of all, a simple entropy-based approach from the MRL Team will be described in Section 3.1.1. A similar, but more complex one will be highlighted in Section 3.1.2 and improved in Section 3.1.3. In the end, an end-to-end approach using reinforcement learning is described in Section 3.1.4.

3.1.1 MRL Team Description Paper 2019

The MRL RoboCup team described its active vision approach in their 2019 team description paper [Mah+19]. Their technical description details an entropy-based approach that is computed on the fly on the robot's hardware. They motivate it by stating that other teams used a larger FOV or a higher frame rate in combination with faster head movements to gather more information in a given time. But the maximum FOV is limited in the RoboCup humanoid league, faster head movements destabilize the robot, and the faster framerates also need more computational resources. They, therefore, describe a dynamic viewpoint selection based on information regarding landmarks used for the self-localization and the ball position. Their greedy algorithm selects the head orientation that minimizes the entropy represented by the visibility or invisibility of the ball and the said landmarks. A visualization of some sample situations with a high or low entropy can be seen in Figure 3.1. The description of their approach does not describe the exact layout of their action space, but it can be assumed that it is discretized or some sort of sampling is applied given that the algorithm checks for different positions and ranks them according to their entropy. They also detail, that the entropy of the different classes is weighted to prioritize the ball as the main objective. More complex dynamics, such as leaving the viewpoint which is optimal based on our partial internal world state

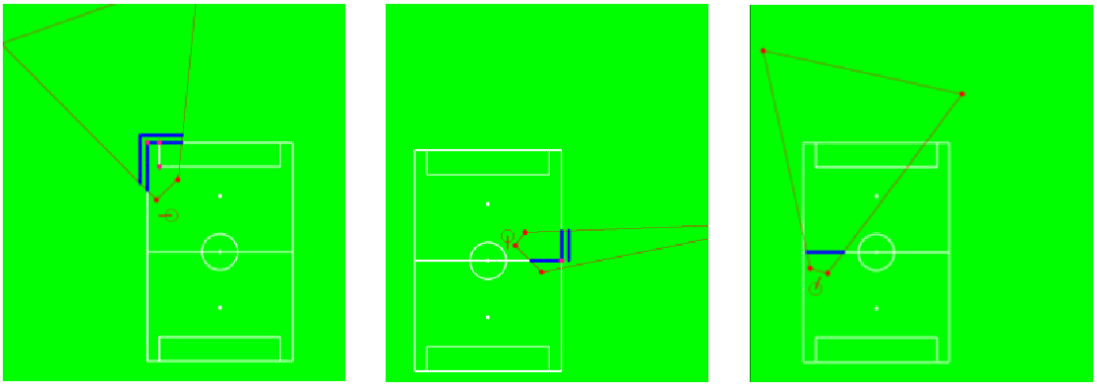

Figure 3.1: This Figure shows the observed landmarks used to calculate the entropy of the environment in different states. Figure from [Mah+19].

to gather information regarding the currently unobserved part of the environment that might lead to a lower overall entropy are not described or modeled. Because of the short nature of the team description papers, only the used approach is described, lacking further evaluation and comparison. A detailed evaluation of a reimplementation of their approach can be seen in Section 5.1 of this thesis.

3.1.2 Entropy-Based Active Vision for a Humanoid Soccer Robot

Seekircher et al. from the Standard Plattfom League RoboCup Team B-Human also proposed an entropy-based system that uses a greedy algorithm to control the head of a Nao robot [SLR10]. Their approach uses the Monte Carlo exploration algorithm to sample possible actions. The explored space not only considers different actions, but also the belief of robots pose which are taken from the robots particle filter based self-localization. The different particles of the filter are used as different possible robot poses that represent the belief of the robot's pose. The actions are sampled for each of the possible positions in such a way that the overall entropy of the state estimate regarding static landmarks and the ball is minimized. To model the probability for a measurement at a given state not only the visibility from the camera model but also distance and the characteristics of the B-Human vision system are modeled. The future influence of the actions in a given state is not modeled in the Monte Carlo exploration for performance reasons. Overall they were able to significantly optimize upon their baseline passive head movements. This can be seen in Figure 3.2

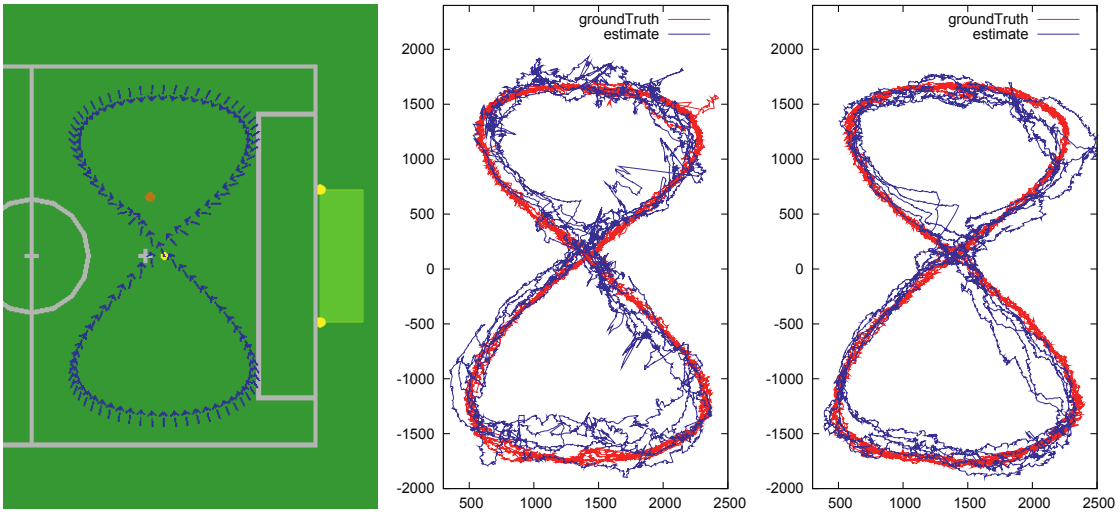

Figure 3.2: Experiment comparing the estimated robot position with both passive (center plot) and active (right plot) camera control. Figure from [SLR10].

3.1.3 A Dynamic and Efficient Active Vision System for Humanoid Soccer Robots

Mattamala et al. [Mat15] optimized the approach described in Section 3.1.2 by using a Priori Information. This is done by building pre-calculated look-up tables for static landmarks used in self-localization. A self-localization particle filter is run with simulated observations to retrieve which viewpoints lead to a better self-localization in a given pose on the field. The degrees of freedom in the head are reduced for this calculation and only include the tilt joint because moving the pan joint is approximately equivalent to a rotation of the robot. This assumption is only true if the robot is precisely upright or the camera orientation is given in a global frame of reference. Even if it is provided in a global frame of reference, tilting of the robot might result in rotations along the uncontrolled roll axis of the head. This also affects the observable area of the field even if the camera center still points to the same position. An example lookup table can be seen in Figure 3.3.

The head orientation from the look-up tables is also dynamically adapted during the execution on the robot. First of all only look-up table values within the head joint constraints are chosen. In addition to that, head movement is penalized so that the entropy reduction must outweigh the action penalty. Other robots are considered as occluding obstacles and their occlusion of localization features is also considered. Last but not least the ball is added to the summed entropy and the best action is selected.

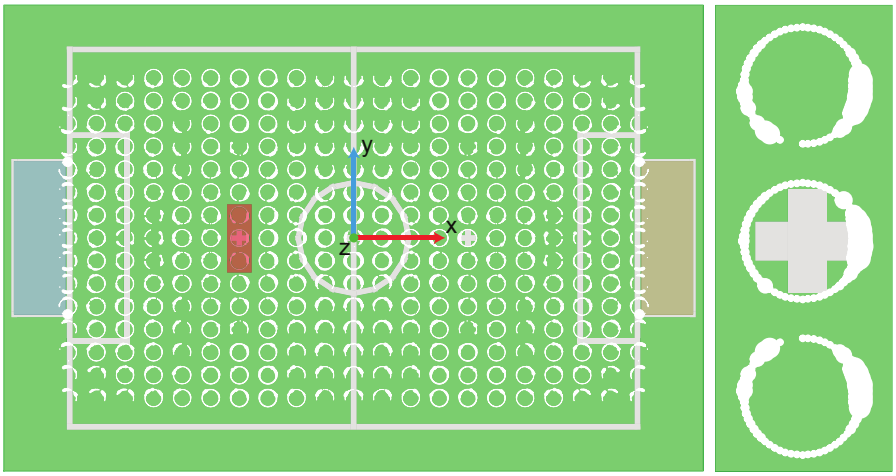

Figure 3.3: Example lookup table showing a grid of field positions each containing scores assigned to a different orientations noted as the width of the circles boundary. Figure from [Mat15].

3.1.4 Real-time Active Vision for a Humanoid Soccer Robot Using Deep Reinforcement Learning

Recently a deep reinforcement learning-based end-to-end approach for active vision in the RoboCup domain was proposed by Khatibi et al.. [KTR21] Their main objective is to keep the ball as well as landmarks used for the self-localization constantly in view. The proposed architecture uses the unprocessed camera images as its observations and controls the head joints with its actions. Their approach is less reliant on other components such as self-localization as it is only using visual features. But this approach also has multiple drawbacks. First of all, the camera image is directly used instead of higher-level information extracted by e.g. an object detection algorithm. This means that in addition to the objective of finding the best viewpoint an object detector for the ball and field features is implicitly trained. This is not only less sample efficient due to the lower sample efficiency of reinforcement learning compared to supervised learning, it is also less efficient during inference as the implicit object detection is only used inside the hidden layers of the policy and a second object detection is needed for the normal ball localization used by the rest of the robot. This being said, there are also benefits to using the image directly as information is lost with each abstraction in the software stack and an end-to-end approach reduces these abstractions. Their approach is trained using the Webots Simulator (see Section 2.5) and the deep Q-Learning reinforcement learning algorithm [Mni+15]. The MDP makes it hard to integrate real-world observations and the visual appearance of the simulated images differs significantly, so sim2real transfer needs to be investigated further as it is not included in the paper.

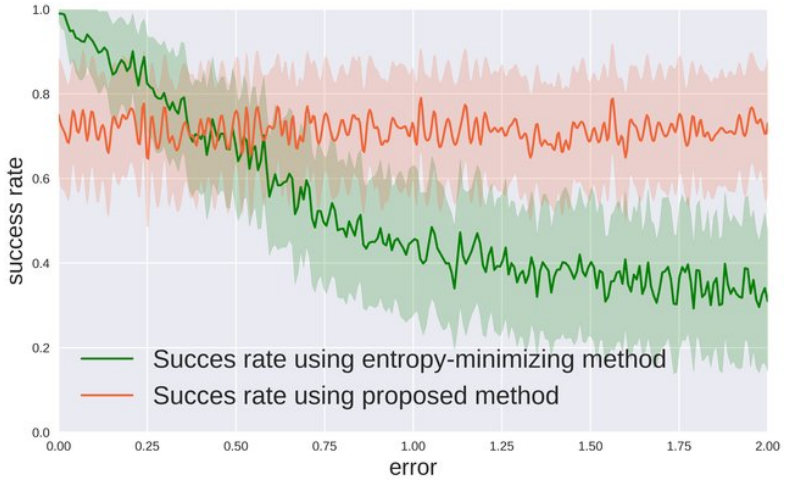

Figure 3.4: Comparison of the entropy based and the proposed method for different localization errors. Figure from [KTR21].

The paper compares itself to the entropy-based approach described in Section 3.1.1. The comparison is mainly focused on the visibility of self-localization features. It is investigated how uncertainty in the self-localization of the robot affects the performance of both approaches. The results can be seen in Figure 3.4. A success rate is used as the metric. They define it as the fraction of features that are observed by the policy divided by the number of possibly observable features for the given robot pose, head joint constraints, and field of view.

In their conclusion, they also suggest the usage of PPO which is used in this thesis to enable a continuous action space.

3.2 Active Vision using Reinforcement Learning

In this Section, two reinforcement learning-based approaches from outside the RoboCup domain are shown. The first one learned an active visual attention model using reinforcement learning. In the second one, a robot performs vision-based object manipulation. An active vision policy is learned to get a better view in case objects are occluded.

3.2.1 Recurrent Models of Visual Attention

In this approach from Google DeepMind, a *Glimpse Sensor* is moved across a given image instead of moving the camera in an active vision fashion. [Mni+14] So technically speaking the approach is a visual attention model, but its workings are quite similar to

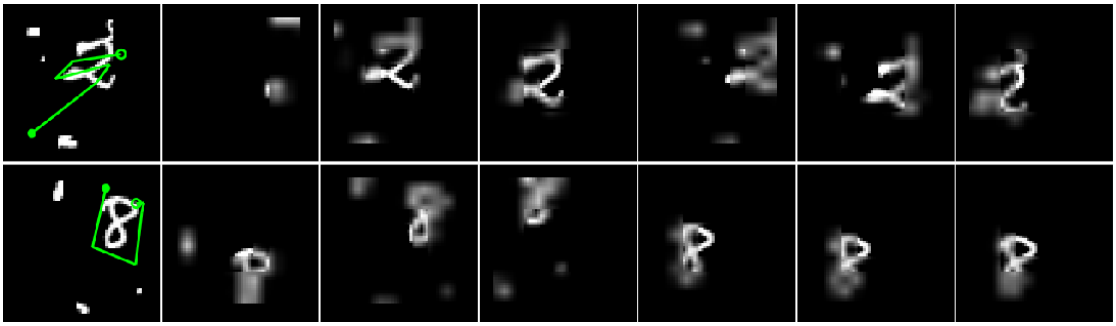

Figure 3.5: This Figure shows the path of the Glimpse Sensor over two images from the Cluttered Translated MNIST dataste. The path (green line) and full resolution image are shown on the left, while the different observations of the Glimpse Sensor are shown on the right. Figure from [Mni+14].

the active vision approaches we are concerned with. They also define their approach as a partially observable Markov decision process (POMDP), because the Glimpse Sensor sensor is not able to observe the whole image at once. Only a part of the available information is given to the agent to reduce the computational cost of processing the complete image at once in full resolution.

They, therefore, define a virtual sensor that is movable across the image and returns a stack of different feature maps. This stack includes low-resolution feature maps that cover a larger area around the position of the virtual sensor as well as narrow high-resolution ones. The observations of this sensor are passed into a recurrent neural network which is able to move the sensor across the image to search for useful features. The action space includes the control of the glimpse sensor's movement as well as a task-dependent output. A simple classification is used in the paper for the task-dependent output. The policy consists of a recurrent neural network. This closed loops interaction between the glimpse network and the environment is not differentiable, so a reinforcement learning-based approach is necessary for the normally supervised classification task. But a hybrid supervised loss can be added to train the task-dependent output in the last step of the epoch. A positive reward is given if the agent succeeds with its classification and no further reward shaping is applied, so the reward is quite sparse. For the training, the REINFORCE policy gradient reinforcement learning algorithm is used.

The proposed approach is evaluated using the MNIST dataset as well two newly proposed datasets that are based on MNIST called Translated MNIST and Cluttered Translated MNIST. The Translated MNIST introduces random translations to the MNIST digits. In addition to that, some artifacts are added to form the Cluttered Translated MNIST. The performance of an agent on the Cluttered Translated MNIST dataset can be seen in Figure 3.5.

3.2.2 Reinforcement Learning of Active Vision for Manipulating Objects under Occlusions

Active vision is important for many tasks in the robotic domain. In the approach presented by Cheng et al. [CAF18] a policy is trained to optimize the viewpoint of a camera during an object manipulation task while taking care of occlusions. The policy for the object manipulation and camera control are trained jointly forming a single policy network. The training environment is based on the *FetchPush-v0* environment from the OpenAI Gym. It is simulated using the MuJoCo physics simulator [TET12]. As a robot model, the Baxter robot is used. Due to the occlusions and limited field of view of the camera they also need to define their process as a POMDP. To simplify the problem a static pretraining for the vision-based object manipulation was proposed. It was accomplished by disabling the active vision action space in the pretraining. Multiple different architectures for the network itself were proposed. They can be seen in Figure 3.6.

Their evaluation shows that an efficient pretaining without the active vision can be done and that the active vision system improves the object manipulation performance if occlusions are introduced. In addition to that, it is highlighted that the model which used the raw image without the RCNN completely failed to perform the object manipulation. Also, models which used only the RCNN data for the gripper object manipulation performed better than the ones that used the additional image information. This leads to the conclusion that the active vision improved their result, even if it was later on added to a non-active vision policy. In addition to that one can see the benefits of using high-level data such as the RCNN detections in this case.

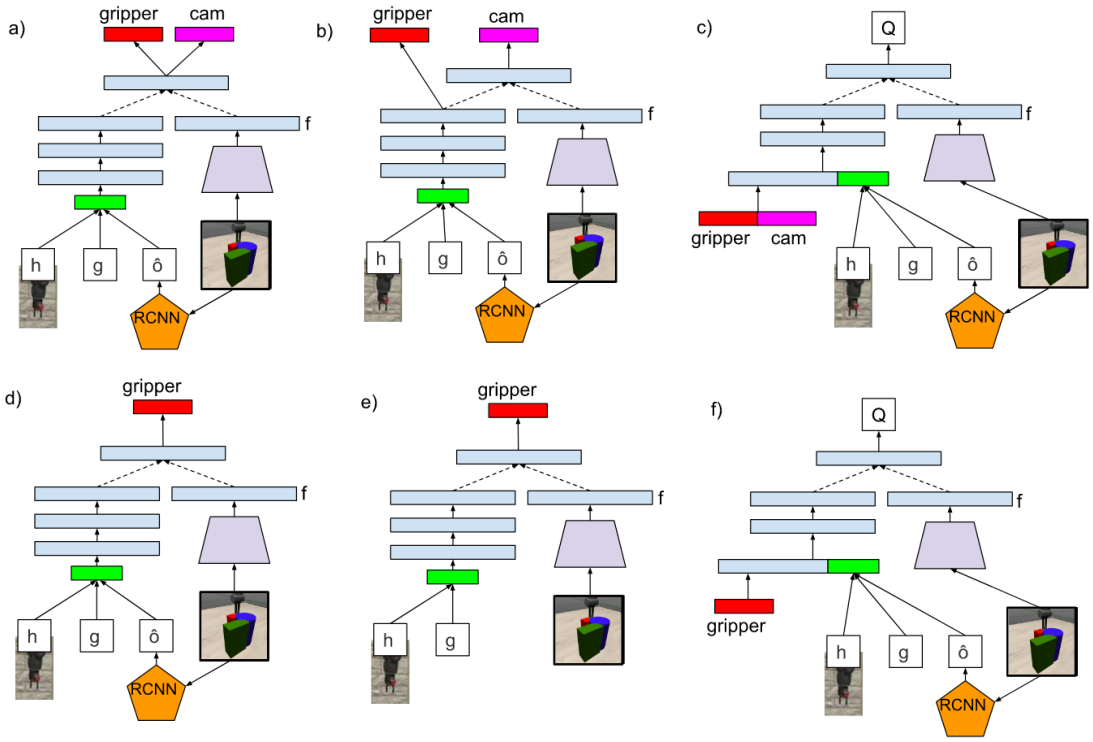

Figure 3.6: Different proposed architectures for the actor (a), b), d), e)) and critic (c), f)) networks. Most of them (except e) utilize a separately trained RCNN (Regions with CNN features) to detect the object of interest. One network (e)) skips the explicit object detection and relies only on the raw image data. The first row describes networks that include active vision actions (cam output). The only difference between them is that a) utilizes both the encoded raw image and the RCNN detections for the object manipulation while b) uses the raw image only for the active vision part. Figure from [CAF18].

4 Approach

In the following Chapter, the approach used in this thesis is described. A neural network is trained to control the head movement of a humanoid Wolfgang-OP robot based on the information in its world model. The head movement control is done in such a way that it maximizes the confidence of its world model. A higher internal world model confidence indicates a smaller divergence between the internal state which was based on partial observations and the world state. Due to the closed-loop nature of the problem, as shown in Figure 4.1, an interactive environment is needed for the training. Reinforcement learning is used to train the neural network policy because the interaction with the environment is not differentiable. The environment should also use little resources during training, due to the low sample efficiency of reinforcement learning. This is the reason why a new high-level simulation environment was developed. It simulates the RoboCup game dynamics, possible observations, and the internal world state of the robot without the need for a physically accurate simulation or the execution of eight robot software stacks. A detailed description can be found in Section 4.1.

Several different observation and action spaces are proposed to solve the nontrivial problem of this partially observable Markov decision process. First of all one might think of recurrent policies such as an LSTM [HS97] to learn more complex patterns that can discover currently unobserved objects in a given situation. But using recurrent policies such as the LSTM is sadly currently not possible in the used training framework. It is therefore investigated how similar effects can be achieved by adding a history to the observations, showing a demonstration pattern, or predicting the parameters of a pattern generator. All of these approaches will be evaluated and compared in Section 5.2.

4.1 High-Level Simulation Environment

Simulating a complete RoboCup game with accurate physics inside the official Webots simulation (see Section 2.5) is slow and expensive. The calculation for a 4 vs. 4 game runs with a real-time factor of around 0.02. But we just need high-level data to simulate the camera observations in the game. The actions that control the camera can also be added to this high-level data without much need for an accurate physics simulation. We, therefore, take the high-level data describing robot and ball poses, select one robot as our agent and add a controllable virtual camera to this agent. The virtual camera can

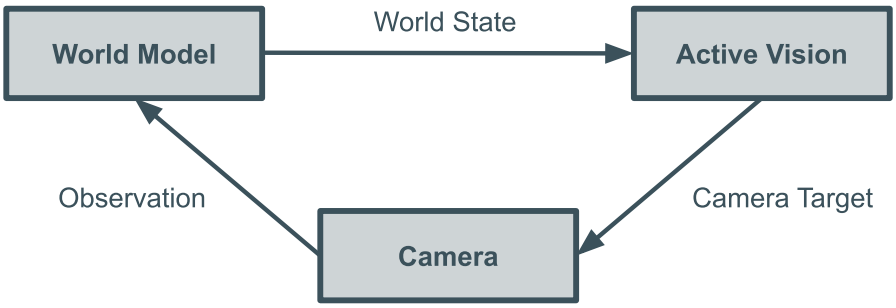

Figure 4.1: This Figure shows the closed loop interaction between the camera, which takes partial observations, the world model which models world dynamics based on them and the active vision component which uses the internal world state to predict a new camera target which closes the loop.

determine which objects are currently in the field of view of the agent. This information is used to build a simulated world model and reward the agent for observations.

Because we acquire the high-level data in an open-loop fashion, there is an abstraction happening. This abstraction decouples the agent's high-level movement from the agent's visual observations. Normally, the robot would move differently if other observations are made, so this is a bias. But it also makes the learning easier as the robot does not need perfect observations to get in certain situations so it can be viewed as some sort of initialization. The rest of the system runs in a closed-loop manner where the actions of the agent influence the state of the environment, which again affects the future observations of the agent. Using high-level data and not simulating the whole robot also prohibits overfitting of the network on certain strategies or positioning of the Hamburg Bit-Bots software stack which would be used if the games were simulated from the ground up in the Webots environment. Code from a few other teams could be used but many teams use proprietary code which would be hard to acquire for reinforcement learning on our machines. So the purpose of the high-level simulation environment is to take the high-level game data and add camera observations which turn the whole process into a POMDP because only a small portion of the field can be observed at once.

Figure 4.2 shows the data flow in the high-level simulator and its interface to the reinforcement learning framework. A visualization of the simulated environment can be seen in Figure 4.3. A 2D top-down view is used for the visualization, but all the projections and movements are calculated in three dimensions.

4.1.1 Data Acquisition

To gather the high-level data needed for the high-level simulation environment game logs from the Webots simulator are used. These logs are normally used for visualization

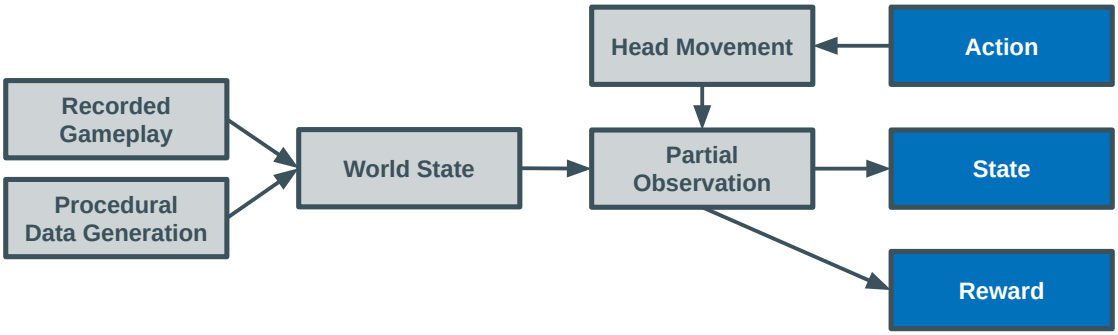

Figure 4.2: This Figure shows the dataflow inside the high-level simulation environment. The high-level data consisting of recorded gameplay or procedurally generated data enters on the left forming a high-level world state. The action of the agent is then applied to the neck joints and a partial observation is calculated with respect to the world state. This partial observation is then passed back to the network as its new internal state. The reward is also calculated based on the partial observation and returned to the reinforcement learning framework. The dark blue boxes on the right highlight the OpenAI Gym Interface.

purposes, where the game can be viewed in the browser after being calculated by a server. The browser rendering includes a 3D viewport with a freely movable camera. So the logs must include geometrical information as well as trajectories for the objects in the scene. Looking at the logs reveals an HTML, x3d, and JSON file as well as some textures. The HTML file includes some JavaScript for the browser visualization as well as some visual and textual elements. Most of it is not needed for the extraction of the movement information, but it can serve as a starting point to understand the way the logs are structured and processed for the visualization The x3d file includes object and mesh information and is based on XML [8]. We can use the file to query all the robots that are present in the game, including information regarding the team (red or blue) and the player number. In addition to that, each robot is associated with a unique object ID which is used later on to query the movement of the object. We can also query the ball including its object ID. This information can be used while parsing the JSON file. It includes position and rotation updates for each simulation step. The object ID from earlier can be used to search for all movements of a given object. Afterward, some interpolation or subsampling is applied to match the update rate to the rate of the high-level simulation environment.

The data extraction code developed for this thesis has been split into its own repository[1] and is available on PyPi[2] as a library for the analysis of RoboCup games or more general

[1] `https://github.com/bit-bots/webots-web-log-interface`
[2] `https://pypi.org/project/webots-web-log-interface`

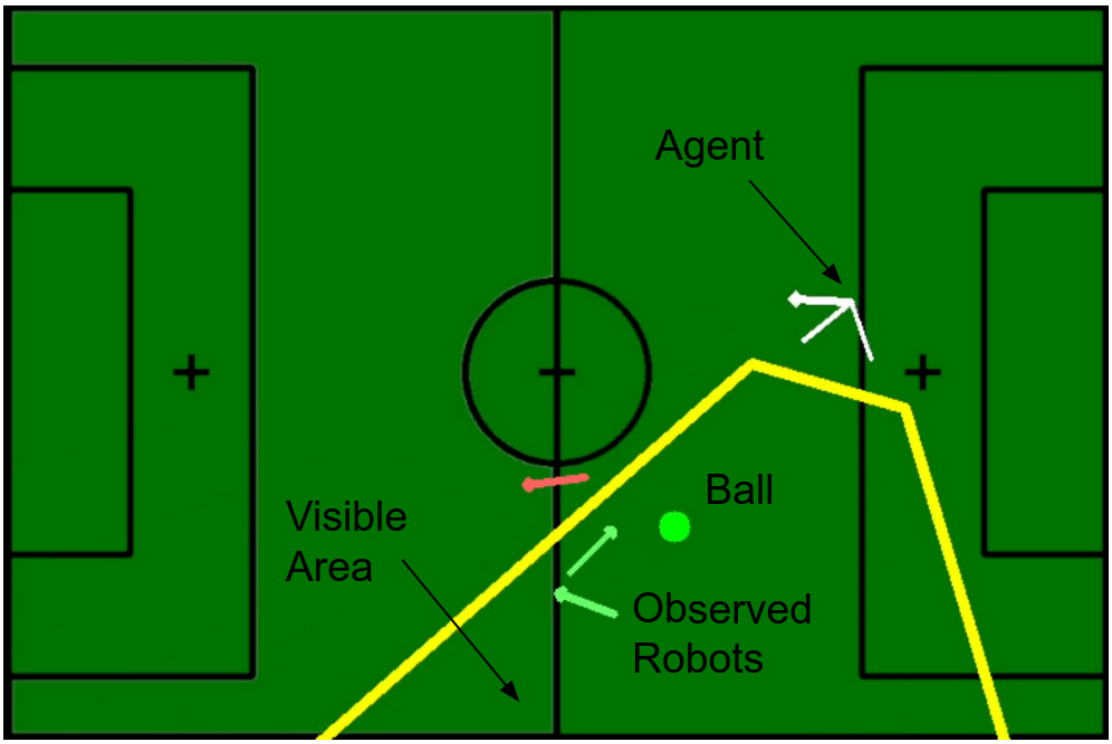

Figure 4.3: This Figure shows a top-down view visualization of the high-level simulation environment. The white arrow marks the pose of our robot. The V-shaped marker indicates the camera orientation and field of view. The yellow polygon surrounds the visible area of the field. The other robots are shown as green and red arrows, with their color indicating if they were observed or not respectively. The same color-coding applies also to the ball which is shown as a small circle. In the background, the layout of the RoboCup field is shown.

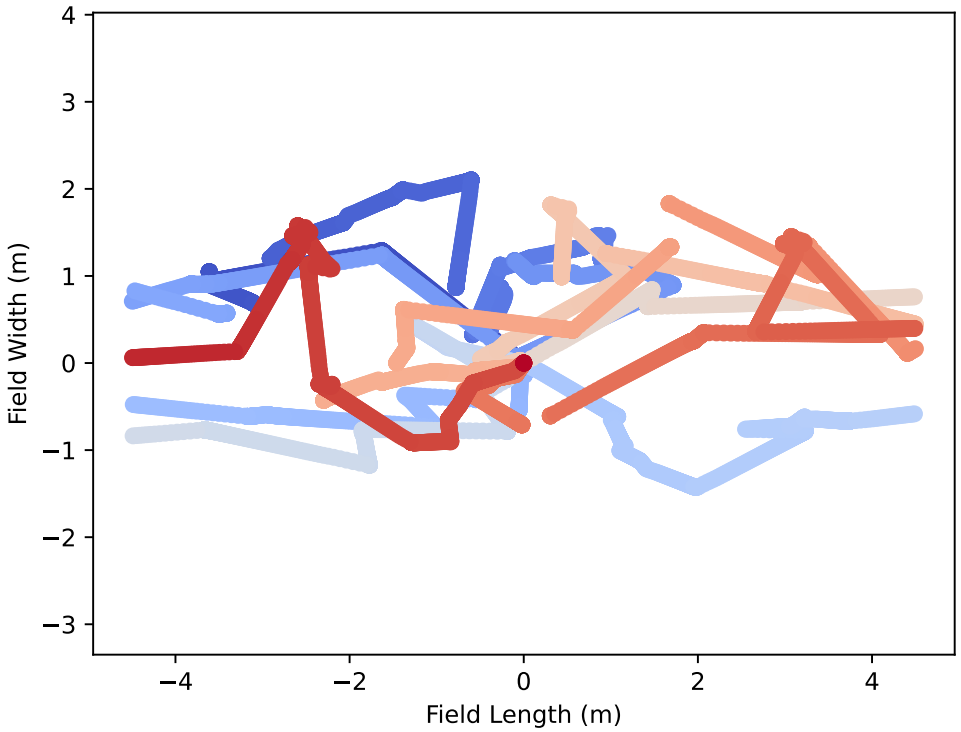

Figure 4.4: Trajectory of the ball in meters during a game at the RoboCup 2021. The games progress is encoded in the path color going from blue to red in 24.7 minutes. The data has been extracted from the game visualization logs.

Webots simulations. An example plot of all ball movement in a given game can be seen in Figure 4.4.

4.1.2 Data Generation

In addition to the gameplay recordings, high-level movements could also be simulated using randomly generated movements. This can be used if no data is available or the data needs to be supplemented to improve generalization. The random generation also has a faster loading time and was therefore used during the development, debugging and initial test training runs. It simulates first-order dynamics for the different classes. Currently, a data generation for ball and robot movement is implemented.

The ball generator includes an exponential velocity dampening to model ball friction. In addition to that, it is randomly accelerated in a random direction. The values of this

acceleration are chosen by sampling a normal distribution. The different hyperparameters of the generator including the kick probability, the mean kick strength and kick strength variance are chosen to mimic values expected in a real RoboCup game. These parameters are not investigated further as we mainly rely on the recorded data. If the generator would be used to train the final policy further effort would be spent to quantify these parameters based on real-world observations.

The robot pose generator works similarly to the ball generator. For position generation, it uses randomly generated walk directions and speeds that are resampled with a certain probability in each timestamp. The orientation is sampled independently from the position generator by a weighted random decision between applying a random angular velocity in the z-axis, resetting the angular velocity, or continuation of the current motion.

4.1.3 Domain Randomization

To help with generalization and sim-to-real transfer some simple domain randomization is applied. All observations including ball and robot observations are augmented using a small amount of Gaussian noise. The same is true for the orientation and position of the observing robot. In the future, the modeling could be improved. This could be done for example by scaling the variance of the Gaussian for a given object with the distance between the observing robot and the said object. For now, only the static Gaussian is implemented to keep the number of hyperparameters down.

4.1.4 Observation Estimation

The generated high-level information needs to be combined with the actions of the agent. At this point, it is assumed that the actions are provided as joint positions for each of the neck joints in the Wolfgang-OP robot. Other action spaces are described in Section 4.2.4 but all of them can be transformed to joint positions after some processing, so they are chosen as the input for the observation estimation.

In the first step, the pose of the agent's robot is queried. Then a transform is applied to get from the robot's *base_footprint* frame [4] to the neck. Here the neck joints are modeled with a combination of static and dynamic transforms. The dynamic transforms for the neck pan and tilt joints are based on the actions clipped to match the constraints of the neck joints. After that two static transforms are applied to get to the camera and *camera_optical_frame* respectively. All transforms are based on the URDF [11] description of the Wolfgang-OP robot.

We now know a chain of transforms from the field frame to the *camera_optical_frame*. This enables us to transform a given point into the *camera_optical_frame* coordinate

system. After this is done we can use the equation 2.10 to check whether or not the given point is currently visible on our image plane. It should be noted, that we also need the intrinsic camera calibration in this step.

The previous steps can be applied for all robots and ball positions, as well as static landmarks used for localization to build a set of all visible objects. This set can be used later on to update the robot's simulated world model as well as the calculation of the rewards or visualization purposes.

4.1.5 World Model Approximation

The world model used by the Hamburg Bit-Bots is a continuously evolving topic. In the past, a particle filter-based approach was proposed for multi-object tracking [Fie19; FBZ19]. Currently, it mainly consists of a Kalman filter [Kal+60] for the ball position and some nearest neighbor matching for observed robots. Future work is planned to improve upon the robot filter. Due to the complexity of the world model implementation as well as the possibility of future changes to the specific approach a simple approximation is done.

We define the approximated world model as the set W. This set contains a tuple for each object o of interest. The tuples themselves contain the two-dimensional position p_o of the object in the field plane as well as a confidence value c_o. We define Δt as the time since the last filter step and λ as the steepness of the linear decay. This results in the following update rule for c_o if o is currently unobserved.

$$c_o = max(c_o - \lambda \Delta t, 0) \tag{4.1}$$

The value of c_o is reset to 1 if o was observed in the given time step. A random smaller value close to 1 could be sampled to harden the policy against inputs from other world model methods, but it has been dropped for now to reduce the number of hyperparameters.

4.1.6 Software Architecture

The learning environment implements an OpenAI Gym [Bro+16] interface on the highest level. This interface is used to communicate with the Stable Baselines 3 [Raf+21] reinforcement learning framework. The environment class *SoccerWorldEnv* itself defines the action as well as the observation spaces for compliance to the Gym interface. It also instantiates the simulation and handles the reward calculation. A new *SoccerWorldSim* simulation is instantiated after each reset call. The *SoccerWorldSim* handles most of

the simulation behavior and structure. It instantiates the robot and ball generators or players. In addition to that, a *WebotsGameLogParser* is created to load a random Webots game replay if a class uses the position or orientation player. The position and orientation generators or players are combined into an instance of the *Robot* or *Ball* class, which also handles the necessary world model calculations. A reference to a random robot is also given to the instance of the *Camera* class. Finally, buffers for the action and observation history are created. Each time the *SoccerWorldSim* is stepped it preprocesses the actions and sends them to the *Camera* class. In addition to that, the world model is stepped, the new object positions are queried and the observable objects are calculated. In the end, the partial observation is constructed and returned to the Gym environment which proxies the step calls.

4.2 Training

After describing the simulation environment we now focus our attention on the training. The training setup itself is described in Section 4.2.1. In addition to that, the architecture of the neural network is shown in Section 4.2.2. The corresponding action and observation spaces are described in Section 4.2.3 and Section 4.2.4 respectively. In the end the reward calculation is shown in Section 4.2.5.

4.2.1 Training Process

The training is done using the *Stable Baselines 3* [Raf+21] reinforcement learning framework. In addition to the core libraries a fork of the *Stable Baselines 3 zoo* repository[3] was used as it provides scripts to train and run policies, handle hyperparameters, or plot results. It enables training with a multitude of different algorithms, including PPO which was used in this thesis, without implementing any further code.

The training was done using multiple workers on a single machine in parallel. Due to the relatively small neural network size (see Section 4.2.2) simulating the environment was the constraining factor and the training was mostly CPU bound. Eight parallel workers were used to collect the rollout buffer.

All settings regarding the setup of the environment were managed using a set of YAML [9] files containing the parameters for the different experiments. The parameters in the YAML files range from the number of robots, over the field size to things like the weights for different parts of the reward or the used action and observation spaces. All of the files for the different experiments are available in the repository of this thesis.[4] Most parameters like field size or the number of robots are constant across all experiments.

[3] `https://github.com/Flova/rl-baselines3-zoo`
[4] `https://github.com/Flova/BA/tree/main/active_soccer_vision/config`

Changes are explicitly mentioned in the description of the experiments in Section 5. The episode length is fixed to a number of steps, by default 4000. This is equivalent to 400 seconds of game time. No early termination of an episode is applied. The 400 second window is randomly sliced out of a game log if a recording is used. Attention is paid to the fact that the last 400 seconds are no valid starting point because they would result in a reduced episode length. The frequency at which the active vision control loop is executed is set to 10 Hz. It can be further investigated how this rate affects the efficiency of the reinforcement learning and the reactivity of the deployed policy, but for now, this near real-time value is chosen manually as a default.

4.2.2 Network Architecture

The model for both the policy and value function consists of a Multi-Layer Perceptron (MLP) neural network. While it is theoretically possible to share the first encoder layers of the network between the policy and value function separate networks are used in this thesis. By default, each network consists of two hidden layers with 64 neurons each. The ReLU function is used as an activation function. Both networks can be seen in Figure 4.5. As described in Section 2.2.1 the policy not only predicts the action values but also a variance which is used to sample the actions during training. This enables further exploration in the training process and is not used during inference.

In addition to the one-dimensional vector observation which is processed directly by the MLP, observations can also come in the form of two-dimensional feature maps that highlight e.g. the amount of coverage on certain field areas. These feature maps are too large to be flattened and input into the MLP directly. We, therefore, need to learn an embedding that abstracts the information from the feature maps into a low-dimensional vector. To do this a convolutional neural network (CNN) can be added to extract features and break the overall size of them down. This has already been done in reinforcement learning e.g. to extract visual information from Atari games [Mni+15]. Adding this CNN encoder is also directly supported by Stable Baselines 3.

4.2.3 Observation Space

The observation space for the partial observation of both the policy as well as the value function can include a multitude of features. In the following Section, a set of possible observations is described. Different combinations of whom form the basis for later experiments.

- **Base footprint pose**: The base footprint pose described the position and orientation of the robot on the field plane. It includes the two-dimensional position on the field as well as the orientation in the field plane. Normally this can be

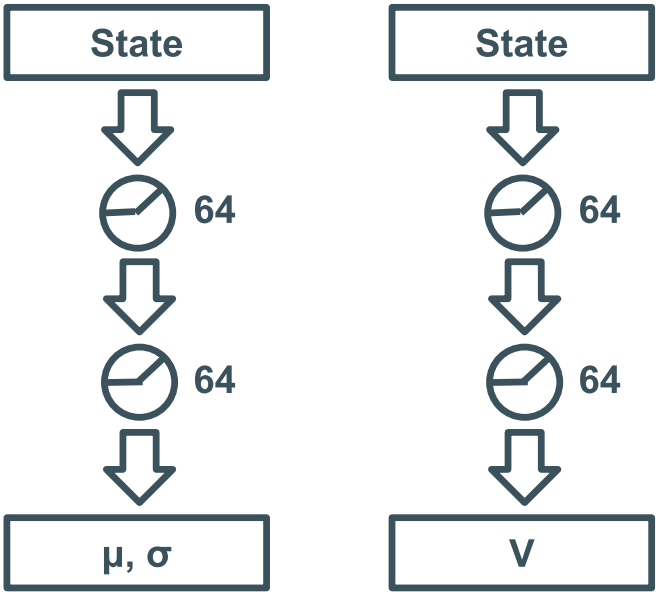

Figure 4.5: This Figure shows the policy and value function networks on the left and right respectively. Both get the same partial observation as an input, which is passed through two hidden layers with 64 neurons each. In the end, the value function outputs the value of the state directly, while the policy predicts both the mean and variance of a multi-dimensional Gaussian distribution that is sampled during training.

represented as a simple angle, but to improve the gradient stability it is preferable to add it as a two-dimensional heading unit vector. Bypassing the angle through the sine and the cosine function the space becomes continuous without the angle wrapping around at 2π [Zho+19]. This results in a four-dimensional vector for the pose.

- **Neck joint positions**: The neck joint positions just describe the position of the head pan and tilt joints. This two-dimensional vector can be used to query the current state of the agent and see if its sensor is near the joint limits.

- **Neck joint position history**: Here a history of n vectors of the neck joint position state is included. By default n is set to 1.

- **Phase**: In case a pattern demonstration is used the policy needs to know at which point it currently is in the repeating pattern. So a sine wave that is in phase with the pattern is provided.

- **Action history**: Is the same way as the neck joint position history includes the last n joint positions, the action history includes the last n actions with $n = 1$ as the default.

- **Ball position and confidence**: This observation includes the two-dimensional position of the ball in the field plane at the last time it was observed as well as its confidence estimated by the world model approximation described in Section 4.1.5.

- **Robot positions and confidences**: Similarly to the ball position and confidence, this observation includes the two dimensional positions of the robots in the field plane the last time they were observed as well as their confidence estimated by the world model approximation described in Section 4.1.5. As there are multiple robots in a game the size of the vector is equivalent to $3n$ where 3 is the number of values needed for each robot and n is the number of robots. Because only the world model is stateful and the MLP is stateless during inference, the robots are shuffled before being assigned a position in the vector. This ensures equal usage of all slots in the vector and should increase learning efficiency and reduce biases.

- **Viewed field regions**: This feature map highlights the number of observations done at each point on the field as a Motion History Image [Aha+12]. This could enable the agent to cover regions that were observed less often in the past more efficiently while searching e.g. for the ball. The dimensions of the feature map are equivalent to the field dimensions. The resolution of all feature maps can be defined in the training configuration file

All of the value ranges from the different observations are manually normalized to be in the range between zero and one.

4.2.4 Action Space

As PPO is able to handle a continuous action space all of the proposed actions are continuous. Four different action spaces are proposed.

- **Joint positions**: When the joint positions are used, two values from the policy output are mapped to the position of the neck joints. The mapping includes a mapping of the output interval of the network to the interval between the maximum joint angles.

- **Joint velocities**: When the joint velocities are used, two values from the policy output are interpreted as a multiplier for the maximum velocity v_{max}. In each sim step, the resulting velocity is multiplied by the sim step size and applied to the current neck joint positions.

- **Pattern**: In this case, the network predicts parameters for a pattern generator instead of controlling the neck joints directly. The pattern generator is essentially a sine function that generates a scanning head motion. The policy can change the amplitude of the pattern as well as add a joint position offset. The actions would be equivalent to the joint position actions if the policy would set the amplitude to zero for both neck joints. So it extends the control of the joint positions with a way to scan a given area of interest in a continuous way without being requiring a history or a recurrent policy. The frequency of the tilt joint sine function is set to half the frequency of the pan joint sine function. This is needed so the robot looks to the right and to the left both when looking far away and to his feet.

- **Absolute**: In contrast to the other actions Cartesian coordinates are used to define a goal position for the camera to look at. So the policy essentially predicts a position of the field that should be in the center camera. The calculation of the necessary joint positions is done outside of the neural network by the high-level simulation environment itself. The closest neck joint positions are used if the current goal cannot be reached. Accepting the actions in the same coordinate system as e.g. the observations of the ball might simplify the problem as not transformation into the image space is learned directly. But the policy still needs some implicit mapping of the image space the projection of the visible area around the center point is still strongly dependent on the camera orientation.

4.2.5 Rewards

The reward function mainly rewards a higher world model confidence. In addition to the confidence of the approximated world model, a more direct reward can be given by rewarding the visibility of objects in a discrete way instead. The set R_{wm} which contains all world model related rewards is formed by the following elements:

- Ball confidence
- Discrete ball visibility
- Mean confidence of all robots
- Fraction of visible robots

The elements of R_{wm} are summed up in a weighted sum forming the first component of the reward function. The second component is added if a demonstration pattern is used. It consists of the mean squared error R_{demo} between the demonstration and the actual joint positions. Both of them are normalized to a value between one and zero so the mean squared error is also in this range. Because we want to penalize the said error we invert the sign and add a bias called R_{base} to keep the value positive. This results in the following equation.

$$r = \sum_{p \in R_{wm}} w_p \cdot p + R_{\text{base}} - R_{\text{demo}} \tag{4.2}$$

4.3 Deployment

After the training and evaluation, a policy is extracted from the PyTorch model and exported to the interchangeable ONNX [3] format. This file is copied onto the robot where an ONNX runtime is used to run the network on the CPU. Deployment on the NCS2 VPU could be possible, but due to the low update rate and small network size, it is considered reasonable to run the network on the CPU. Regarding the ROS software layout on the robot, it is integrated inside the robot's head behavior node (see Figure 2.6 for reference). The node consists of a Dynamic Stack Decider (DSD) [PB21] that is able to choose from different head modes for different situations. Many of these head modes will be obsolete if the active vision mode is used, but for flexibility, the active vision is integrated as another head mode alongside the existing ones. The head behavior node also handles the task of collecting all the data (e.g. world model information), normalizing it, and feeding it to the network if it is needed. In the end, head joint positions are sent to a specific topic for further processing in the rest of the software stack.

5 Experiments

This chapter presents the different experiments performed in this thesis. We start with the comparison against two other approaches in Section 5.1. The effects of different action and observation space configurations are presented in Section 5.2. Following that Section 5.3 investigates the viability of an additional CNN encoder for feature maps. Finally, Section 5.4 compares different network sizes for the policy and value function.

5.1 Baseline Comparison

Multiple entropy-based approaches were described in the related work in Chapter 3. The one described mainly in Section 3.1.1 and briefly in Section 3.1.4 has been integrated into the evaluation environment used for this thesis. The approach is compared to the reinforcement learning-based approach presented in this thesis. In addition to that, both approaches are compared against a simple scanning pattern used by the Hamburg Bit-Bots.

5.1.1 Setup

The entropy-based approach is provided with the partial observation given to the base models in Section 5.2.1 as it does not need any demonstration or input history. It is compared to the best (as determined in Section 5.2) learning-based policy, as well as, the static keyframe pattern used by the Hamburg Bit-Bots.

The entropy-based approach minimizes the entropy by sampling a number of different joint positions and calculating the estimated number of visible objects for each class based on the sampled joint positions and the current world model state. The number of objects for each class is combined into a ranking score by summing up the fraction of visible objects for all classes and adding a movement penalty. The penalty is based on the euclidean distance in the joint space between the current joint positions and the sampled ones. It is mainly used to remove jitter in the case of an otherwise equal score at multiple different positions. The new neck joint position action is selected by selecting the joint position sample with the highest-ranking score (equivalent to the lowest entropy). The sampling of the continuous action space is done using Gaussian

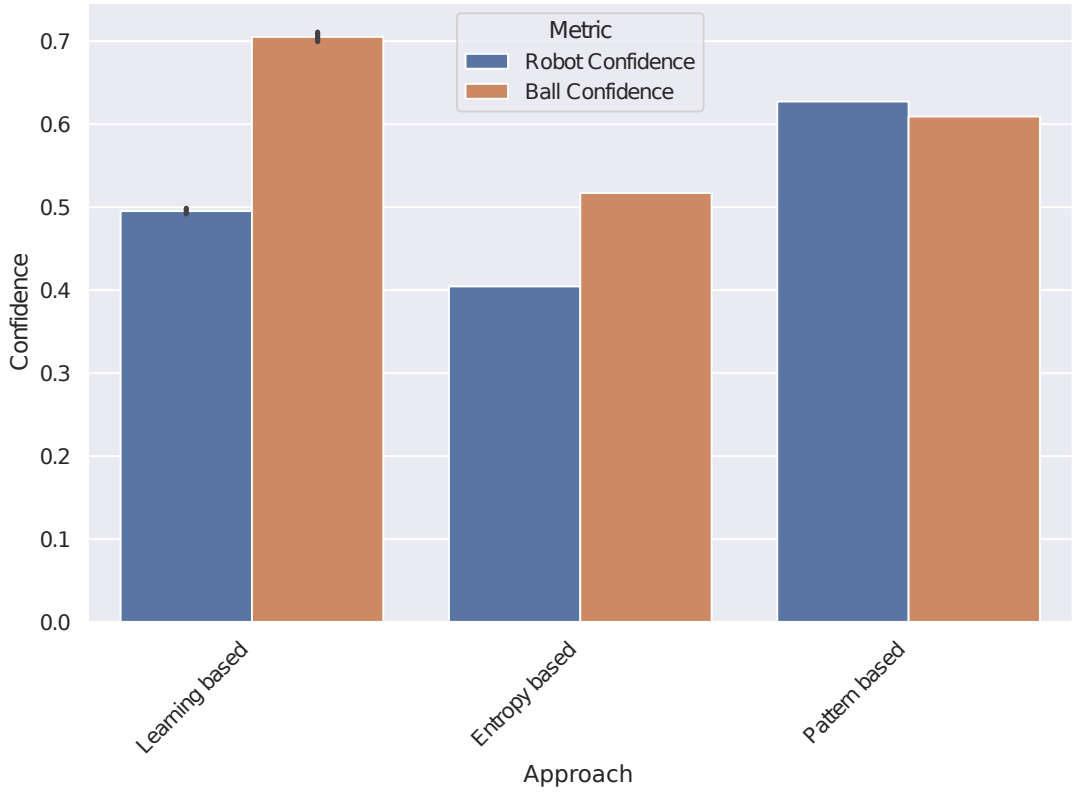

Figure 5.1: Performance of the learning based approach (left) compared to the entropy based (middle) and pattern based (right) one. As a metric the world model confidence for the robot (blue) and ball (orange) is used.

distribution and a Monte Carlo exploration around the current neck point position. The sampling has two parameters, where σ describes the variance of the Gaussian used in the sampling and *n_samples* which controls the number of samples in each step. The default values for σ and *n_samples* are 0.1 and 20 respectively.

The static keyframe pattern moves the head sequentially to a list of predefined keyframes. The keyframes are positioned in such a way that the head follows n (default $n = 3$) horizontal scan lines, reaching the horizontal joint limit at the end of each line.

5.1.2 Results

The evaluation results can be seen in Figure 5.1. The entropy-based approach is outperformed by the learning-based one, as well as, the static pattern-based one in both the ball and robot class. This seems to be the case because it assumes a normal MDP and

not a POMDP. So it trusts the world model in regards to the world belief and assumes at it has an accurate representation of all currently unobserved objects. This assumption is reasonable if it is used to just optimize for static landmarks whose positions are known regardless of the field of view if the localization is reasonably accurate. For dynamic objects like the ball or robots, a POMDP needs to be assumed. This issue also exists for the learning-based approaches that have no demonstration or pattern action space. They only converge to a local minimum taking only recently visible objects into account. In contrast to that, the demonstration or pattern actions allow the policy to explore the environment in a controlled way if objects are missing. The static pattern scans as much of the environment as possible. This comes at the cost of viewing certain areas with a low observation probability in the same way as ones with a high probability, but this is good to cover all objects without missing any. In the future, one could also add similar functionally to the entropy-based approaches by triggering a partial or complete scan of the environment if some conditions are triggered. The learning-based approaches seem to focus more on the ball class, even though the overall reward for the ball and robot classes are equal. This could be due to the ball being only a single instance with a higher per instance reward, which is easier to learn than a multi-instance class with a lower per instance reward.

5.2 Observation and Action Spaces

Different observations and actions have been proposed in Section 4.2.3 and Section 4.2.4. In the following experiment, they are combined into reasonable combinations that are trained, evaluated, and compared against each other to find the best configuration.

5.2.1 Setup

The different configurations and their mapping to the observations and actions can be seen in Table 5.1 and Table 5.2 respectively. All agents trained in the following trails have the *Neck joint positions* as well as the *Base footprint* and the *Ball state* which includes the Ball position, as well as, the world model confidence, in their observation space. All of the configurations also target the robot class so the *Robot states* are also included. This configuration is assumed to provide the minimum information needed to solve the problem. We call it the *Base* configuration. For all actions that control the neck joints directly, meaning the *Velocity* and *Position* action spaces, two additional observation configurations were evaluated. The first one includes a history of both the last action and the last neck joint positions. This could enable better scanning patterns because a history is provided to the otherwise not recurrent policy. We call this one the *History* observation configuration. The second one extends the *Base* one with a sinusoidal phase observation. These configurations are used in combination with the sinusoidal demonstration in the reward function. The phase provides the policy with information regarding the current state of the demonstration pattern.

All of the configurations were trained for 5 million steps using the hyperparameters in Table 5.3 resulting in a training time of about 6.5 hours on an AMD Ryzen 9 3900X. All policies are trained 3 times with seeds 0, 1, and 2 respectively.

After the training, all policies are run in the environment in a deterministic way on the same game from the RoboCup 2021. The confidence of the simulated world model is logged for the robot and ball classes. The mean confidence during the game for each class is chosen as our metric.

Name	Neck joints	Base foot-print	Ball state	Robot states	Phase	Neck joints his-tory	Action his-tory
Velocity History	✓	✓	✓	✓		✓	✓
Velocity Demonstration	✓	✓	✓	✓	✓		
Velocity Base	✓	✓	✓	✓			
Position History	✓	✓	✓	✓		✓	✓
Position Demonstration	✓	✓	✓	✓	✓		
Position Base	✓	✓	✓	✓			
Absolute Base	✓	✓	✓	✓			
Pattern Base	✓	✓	✓	✓			

Table 5.1: Parts of the observation space used for each of the configurations. For a detailed description of each part of the observation space see Section 4.2.3

Name	Velocity	Position	Absolute	Pattern
Velocity History	✓			
Velocity Demonstration	✓			
Velocity Base	✓			
Position History		✓		
Position Demonstration		✓		
Position Base		✓		
Absolute Base			✓	
Pattern Base				✓

Table 5.2: Action spaces used for each of the configurations. For a detailed description of each of the action spaces see Section 4.2.4.

Name	Value
Number of Environments	8
Total Number of Steps	$5 \cdot 10^6$
Number of Steps per Environment in the Rollout	4096
Number of Epochs during Optimization	20
Batch Size	32
Learning Rate	$2.5 \cdot 10^{-4}$
ϵ	0.2
γ	0.99
λ	0.95
Entropy Coefficient	0.001

Table 5.3: This Table shows the hyperparameters used for all experiments.

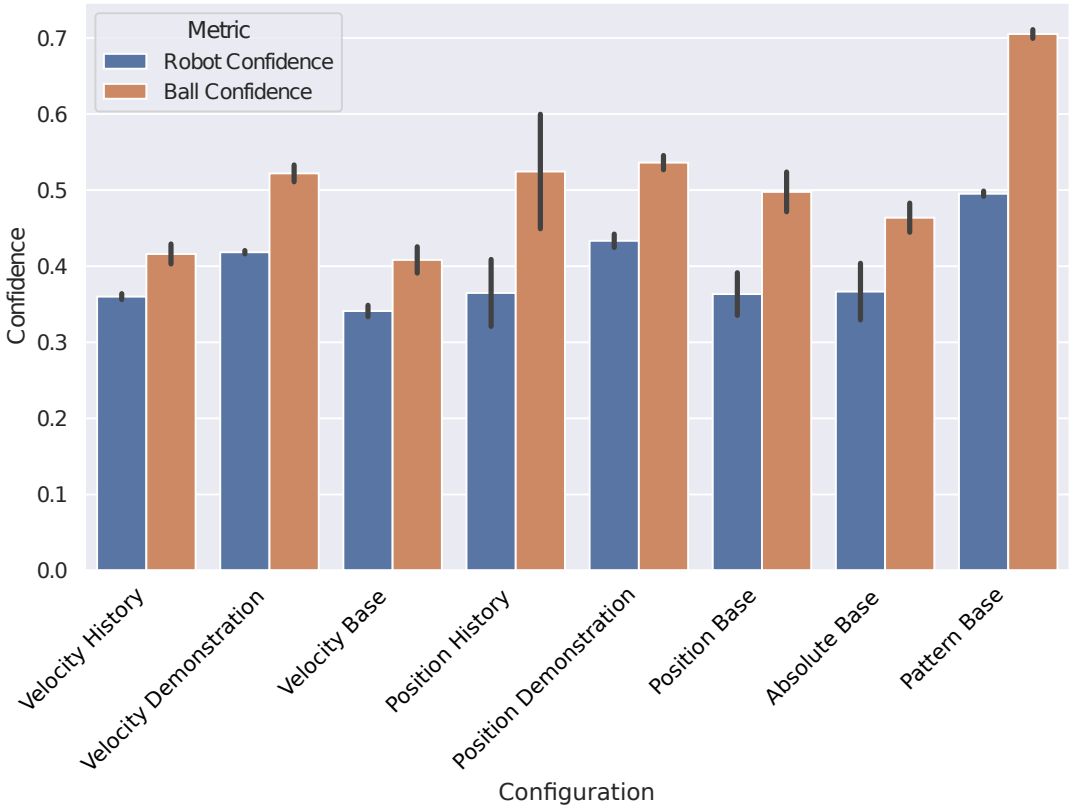

Figure 5.2: Class-wise confidence for each configuration. The orange bars show the mean confidence for the ball class while the blue ones show the mean confidence for the robot class. The evaluation was run with a policy from each of the training seeds, the standard deviation between the results of the different trainings is shown by the black error bars.

5.2.2 Results

Running the experiment as described in the previous Section 5.2.1 resulted in the mean confidences for the ball and robot classes shown in Figure 5.2. It can be seen that the pattern generator-based action in combination with the default observations performed substantially better in the ball class than most other combinations. Its mean confidence for the robot class is lower but still the best among the other ones. The lower mean confidence in the robot class compared to the ball class can be seen across all configurations and indicates that the robot class is more difficult, possibly because it consists of multiple individual instances that are often not all observable at once. When we compare the first-order control of the velocity actions to the position ones it can be seen that they are consistently outperformed by the latter. Both the history and the pattern demonstration show improvements over the base observations, with the pattern

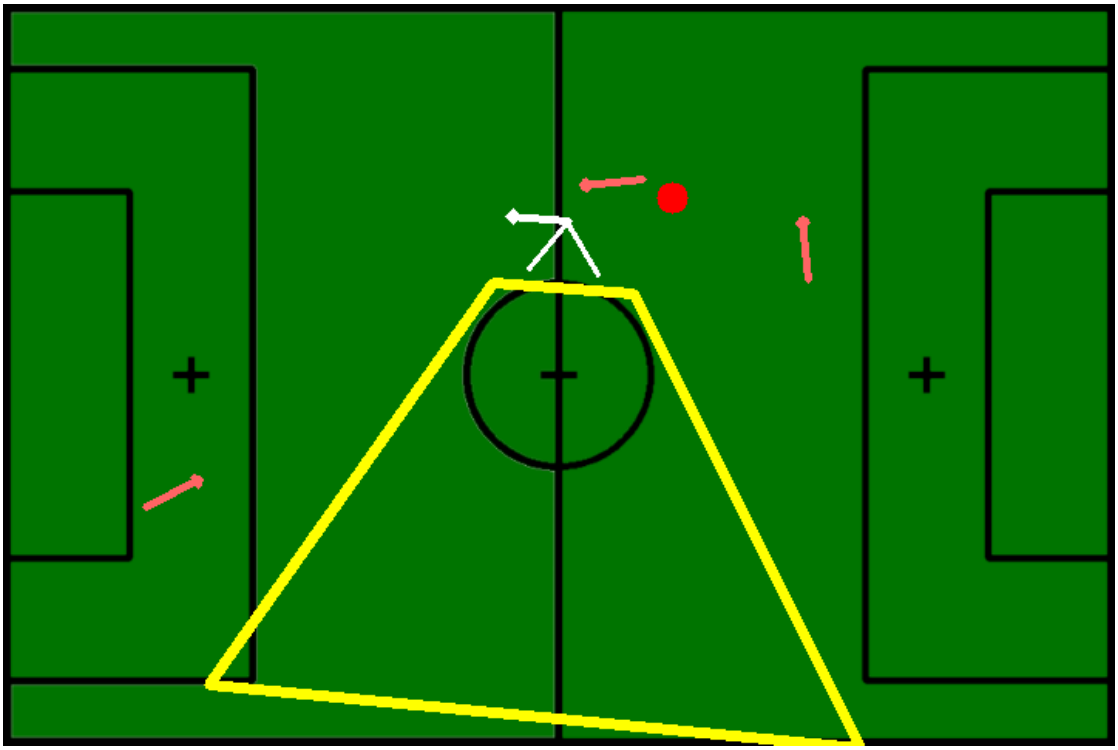

Figure 5.3: Clipping can be observed if the absolute position goal is not reachable in the given neck joint constraints. Policies trained with absolute actions consistently failed to avoid the constraints. They did not focus on a secondary target (for example the robot on the left side of the field) if the primary goal was not reachable.

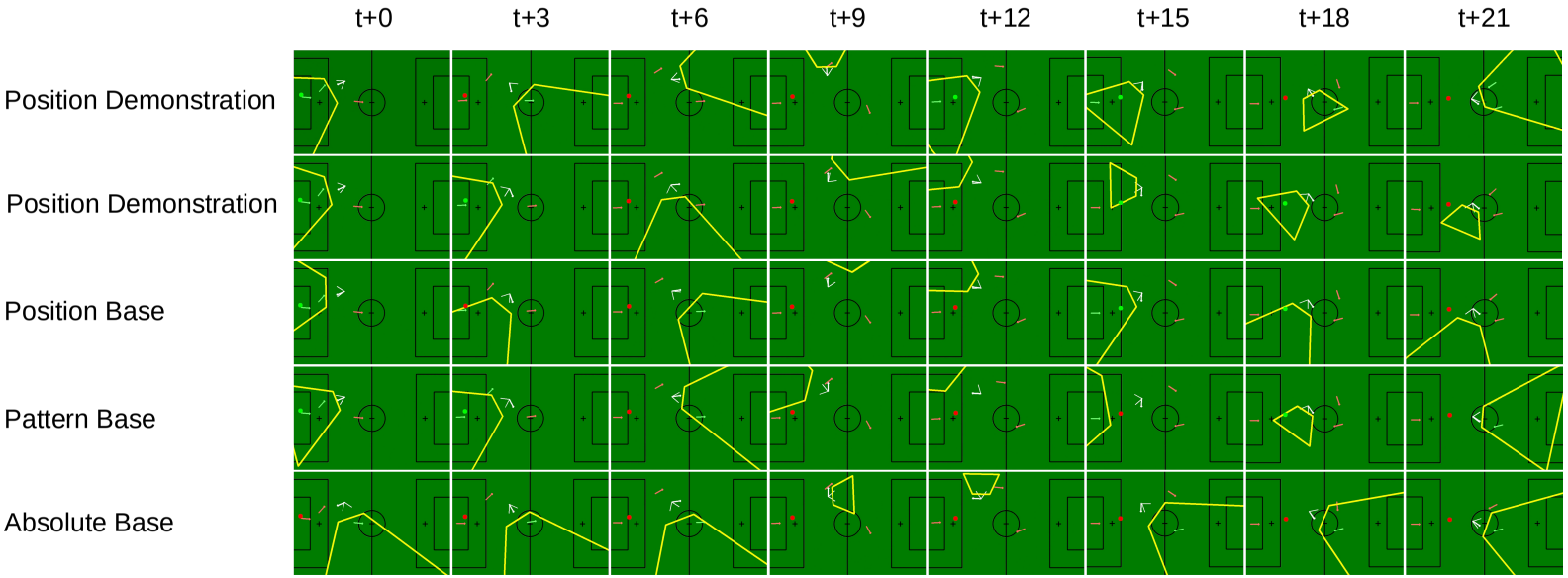

Figure 5.4: Image sequences showing the state of the environment as well as the executed observations for the *Position Demonstration*, *Position History*, *Position Base*, *Pattern Base*, and *Absolute Base* configurations over during a 21 seconds long game scenario. The velocity action space configurations were left out due to being very similar to image sequences of the position action configurations.

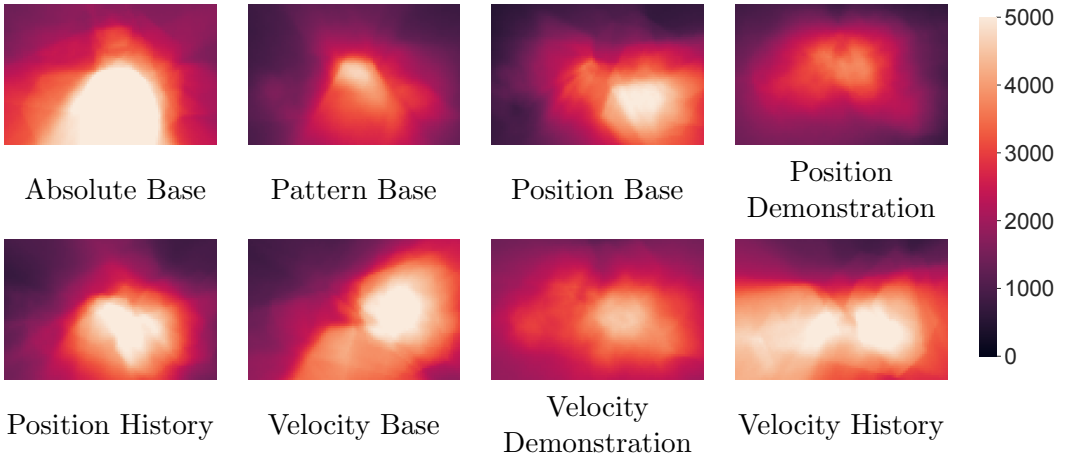

Figure 5.5: This figure shows a heatmap for each of the different configurations. The color of a given point on the field notes the number of steps in which it was observed.

demonstration also outperforming the history observations. The worst performance was observed with absolute actions. A qualitative analysis shows that it fails to avoid the joint angle constraints and clips while trying to reach unrealistic absolute goals. This behavior can be seen in Figure 5.3 as well as Figure 5.4. In addition to that, it shows, like all base observation models, no signs of active exploration of the partial observable environment during inference. This can also be seen in the image sequence of Figure 5.4. An increased amount of exploration of the pattern demonstration configurations and therefore a more even field coverage can also be seen in Figure 5.5. The figure also shows a higher amount of coverage around the center point, which is a very active area on the field.

Figure 5.6: An example Motion History Image feature map before being processed by the CNN. The recently visible regions can be clearly seen in the bright part of the feature map, while unseen regions are black.

5.3 CNN Encoder for Feature Maps

5.3.1 Setup

In this experiment, the performance of the CNN encoders for feature maps is evaluated. For this purpose a Motion History Image (an example can be seen in Figure 5.6) of the recently covered field regions was encoded using the CNN, then flattened, and concatenated with the other observations. Different configurations from the experiments in Section 5.2.1 were used as a starting point. The selected configurations were *Position Base*, *Position Demonstration*, and *Pattern Base*. Their only modification was the concatenation of the flattened and encoded features from the Motion History Image to the observations. The number of time steps during the training was raised from 5 to 10 million steps, in case the increased complexity requires a longer training time. The rest of the training setup was identical.

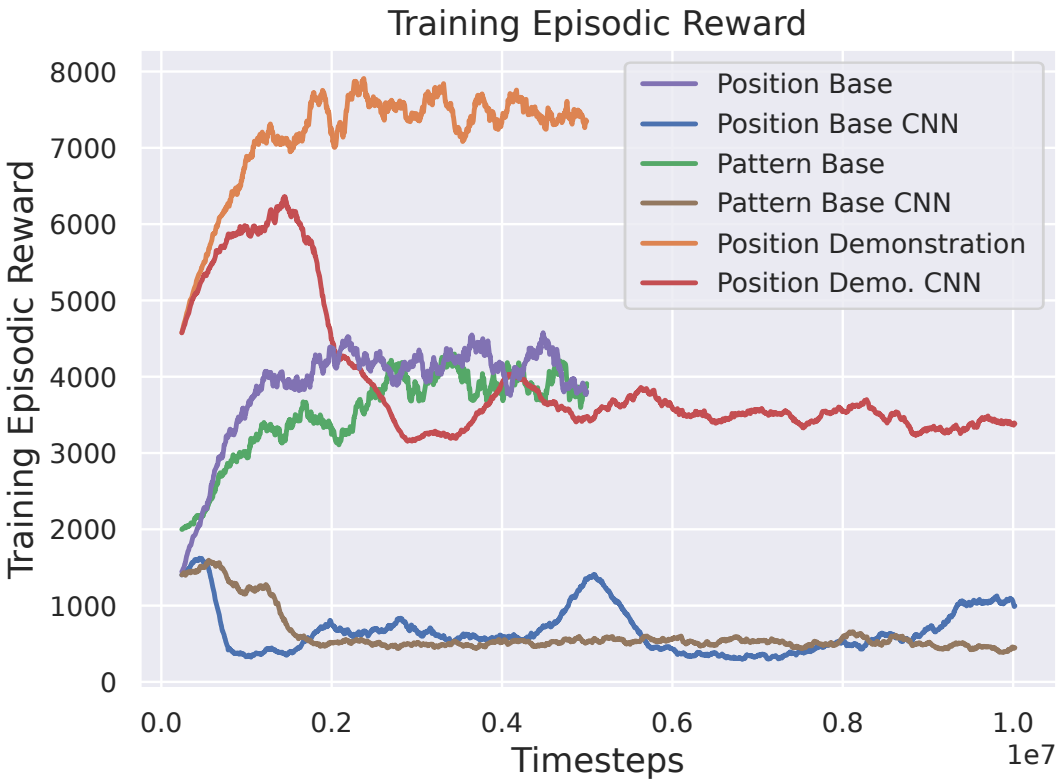

Figure 5.7: This Figure shows the training reward for the evaluated policies both with and without the CNN feature extractor.

5.3.2 Results

As seen in Figure 5.7 all CNN policies failed to converge. They start up at a similar level compared to their counterparts and then even drop in performance compared to this starting point. The addition of the CNN seemed to have a strong negative impact on all evaluated policies. It needs to be added that both the *Position Demonstration* and *Position Demonstration_cnn* configurations include the demonstration reward, so they can easily be compared with each other, but a direct comparison between them and the other configurations is meaningless. For a demonstration independent evaluation see Section 5.2. The reduced performance could be related to a bad hyperparameter, increased gradient noise by many additional observation input dimensions with a small practical benefit, or an implementation error.

5.4 Network Architectures

The size of the neural network used in both the value function and the policy might impact performance. The idea is that a larger network might approximate more complex functions and therefore enable the handling of more complex situations.

5.4.1 Setup

A further evaluation of different hidden layer sizes for the MLPs used in both the policy and the value function (described in Section 4.2.2) was conducted. The setup is based on the setup in Section 5.2.1. In addition to the standard network with a 64 neuron hidden layer size, a second network with a hidden layer size of 255 was proposed. Both of them were trained using the position action and a demonstration pattern.

5.4.2 Results

In contrast to the expectation of a larger network being able to approximate a more complex function, the larger network performed worse as seen in Figure 5.8. Notably, both the ball and robot class performed equally compared to Section 5.2 where the ball class had a consistently higher mean confidence. One explanation for the worse performance might be a suboptimal hyperparameter for the large network as both networks have been trained with the same hyperparameters. It could also be the case that the larger network size allowed the network to memorize the training games and overfit on them.

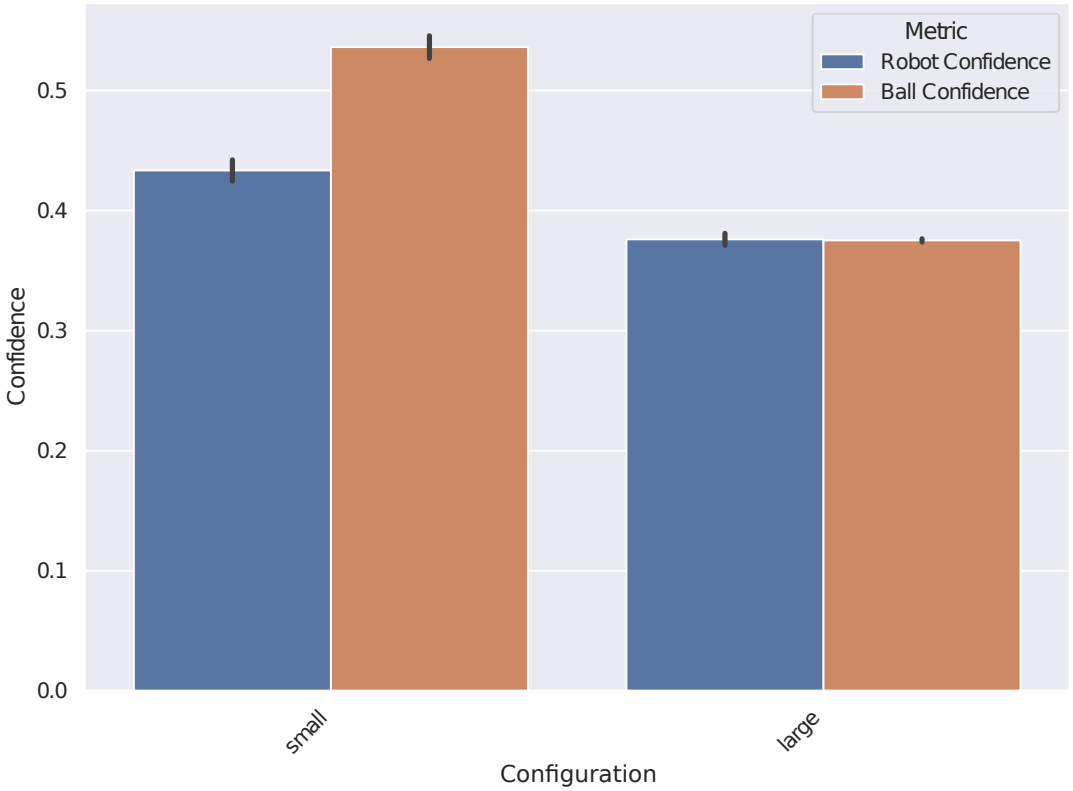

Figure 5.8: Performance of the small network (two hidden layers with 64 neurons) compared to the large network (two hidden layers with 255 neurons each).

6 Conclusion

Coming back to our research question, we wanted to test how a reinforcement learning-based active vision approach performs in the RoboCup soccer domain and how its performance can be improved.

The evaluation in Chapter 5 shows that the best configurations of the reinforcement learning-based approach are able to outperform the entropy-based baseline significantly. Prior to this, different configurations were evaluated to maximize the performance of the reinforcement learning-based approach. The comparison of the different action and observation configurations showed a clear advantage for configurations that utilize a pattern demonstration during training or output the parameters of a pattern generator. Only those configurations were able to efficiently handle the POMDP without getting stuck on local minima as their design encouraged exploration of the partially observable environment. The *Base* configuration, as well as the *History* configuration, were not able to explore their environment sufficiently which resulted in a local optimization based on randomly observed objects. The pattern generator-based policy, as well as the demonstration-based one, outperformed the entropy-based baseline algorithm, which by design had issues with the partially observable environment. The static head pattern currently used by the Hamburg Bit-Bots was only outperformed for the ball class. Overall the pattern generator and joint position action spaces outperformed the joint velocity and Cartesian position ones. Regarding the size of the policy and value function neural network, different configurations were trained and evaluated. Surprisingly, the larger model performed worse than the smaller one, which could be attributed to mismatching hyperparameters or overfitting on the high-level data due to the increased memorization capabilities of a larger network. To enable the incorporation of feature maps into the observation space the usage of a CNN encoder was evaluated. This was not successful as the whole training failed to converge when the CNN encoder was used.

To get these results a novel reinforcement learning approach for active vision in the RoboCup soccer domain has been proposed in Section 4. It is trained to maximize the world model confidence for different object classes like enemy robots or the ball. To train it in a closed-loop fashion a high-level simulator that simulates high-level partial observations based on game recordings was proposed. The high-level game recordings were taken from different RoboCup games in 2021. Then the simulator was used in combination with the PPO reinforcement learning algorithm to train a multitude of different policies with different observation and action spaces. Multiple experiments (see Section 5) were performed. They included a comparison of different action and

observation spaces (see Section 5.2 and Section 5.3), as well as different model sizes (see Section 5.4). In addition to that, the performance was compared to an existing entropy-based approach, as well as a static approach were evaluated in Section 5.1.

6.1 Future Work

In the future, the benefits of the proposed approach on the real robot could be evaluated. An important point would be to test it not only in an internal test game but also when playing against other teams. Additionally, a simulation baseline for other teams could be acquired by parsing, not only the robot's position but also the forward kinematics up to the camera in the Webots log files. Combined with some extra information regarding the team's field of view (not included in the logs) this could be used to evaluate how well the approaches of other teams perform. Also, the impact of false positives and other errors in the internal world state need to be evaluated as an idealized world model, object detection, and self-localization are assumed during the training.

In addition to that, a recurrent policy could be integrated if the framework support for this is added to Stable Baselines 3. If this is done LSTM based policies could be evaluated. This could enable more complex policies that not only take the current world state into account but also have an internal state which can be utilized to perform more complex scanning motions. Such motions could differ significantly from the sinusoidal patterns used by the pattern generator actions or the pattern demonstration.

A relatively simple addition would be the integration of static objects and landmarks used for self-localization in addition to the evaluated robot and ball classes. Features of interest would include the line intersections on the field or the goalposts. Both of which were left out for now as this thesis mainly focused on classes that require a POMDP and static landmarks only need an MDP. But the observation of such landmarks could affect the real-world performance of a self-localization significantly and a good self-localization is the basis for many higher-level tasks.

Another interesting topic would be the integration of support for robots with multiple cameras, as the Wolfgang-OP robot used in this thesis only has one (see Section 2.7), so this case was not covered for now. Other reinforcement learning algorithms could also be investigated in the future, even though PPO matches the current requirements for a model-free algorithm that also has a continuous action and observation space.

Bibliography

Literature

[Aha+12] Md Atiqur Rahman Ahad et al. "Motion history image: its variants and applications". In: *Machine Vision and Applications* 23.2 (2012), pp. 255–281.

[Baj88] Ruzena Bajcsy. "Active perception". In: *Proceedings of the IEEE* 76.8 (1988), pp. 966–1005.

[Bar+19] Daniel Barry et al. "XYOLO: A model for real-time object detection in humanoid soccer on low-end hardware". In: *2019 International Conference on Image and Vision Computing New Zealand (IVCNZ)*. IEEE. 2019, pp. 1–6.

[Bes+19] Marc Bestmann et al. "Hamburg bit-bots and wf wolves team description for RoboCup 2019 humanoid KidSize". In: *RoboCup Symposium*. 2019.

[Bes+21] Marc Bestmann et al. "Wolfgang-OP: A robust humanoid robot platform for research and competitions". In: *2020 IEEE-RAS 20th International Conference on Humanoid Robots (Humanoids)*. IEEE. 2021, pp. 90–97.

[BGZ19] Marc Bestmann, Jasper Güldenstein, and Jianwei Zhang. "High-frequency multi bus servo and sensor communication using the Dynamixel protocol". In: *Robot World Cup*. Springer. 2019, pp. 16–29.

[BHW17] Marc Bestmann, Norman Hendrich, and Florens Wasserfall. "ROS for humanoid soccer robots". In: *The 12th Workshop on Humanoid Soccer Robots at 17th IEEE-RAS International Conference on Humanoid Robots*. 2017, p. 1.

[Bro+16] Greg Brockman et al. "Openai gym". In: *arXiv preprint arXiv:1606.01540* (2016).

[Bur+02] Hans-Dieter Burkhard et al. "The road to RoboCup 2050". In: *IEEE Robotics & Automation Magazine* 9.2 (2002), pp. 31–38.

[BWL20] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. "Yolov4: Optimal speed and accuracy of object detection". In: *arXiv preprint arXiv:2004.10934* (2020).

[CAF18] Ricson Cheng, Arpit Agarwal, and Katerina Fragkiadaki. “Reinforcement learning of active vision for manipulating objects under occlusions”. In: *Proceedings of The 2nd Conference on Robot Learning*. Ed. by Aude Billard et al. Vol. 87. Proceedings of Machine Learning Research. PMLR, 2018, pp. 422–431. URL: https://proceedings.mlr.press/v87/cheng18a.html.

[Che+19] Zexi Chen et al. “Champion team paper: Dynamic passing-shooting algorithm of the RoboCup Soccer SSL 2019 Champion”. In: *Robot World Cup*. Springer. 2019, pp. 479–490.

[CLK11] Shengyong Chen, Youfu Li, and Ngai Ming Kwok. “Active vision in robotic systems: A survey of recent developments”. In: *The International Journal of Robotics Research* 30.11 (2011), pp. 1343–1377.

[FBZ19] Niklas Fiedler, Marc Bestmann, and Jianwei Zhang. “Position estimation on image-based heat map input using particle filters in cartesian space”. In: *2019 IEEE International Conference on Industrial Cyber Physical Systems (ICPS)*. IEEE. 2019, pp. 269–274.

[Fie+19] Niklas Fiedler et al. “An open source vision pipeline approach for robocup humanoid soccer”. In: *Robot World Cup*. Springer. 2019, pp. 376–386.

[Fie19] Niklas Fiedler. “Distributed multi object tracking with direct FCNN Inclusion in RoboCup humanoid soccer”. In: *Bachelor Thesis at the University of Hamburg* (2019).

[Gül19] Jasper Güldenstein. “Comparison of measurement systems for kinematic calibration of a humanoid robot”. In: *Bachelor Thesis at the University of Hamburg* (2019).

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[Kal+60] Rudolph Emil Kalman et al. “A new approach to linear filtering and prediction problems [J]”. In: *Journal of basic Engineering* 82.1 (1960), pp. 35–45.

[Kit+97] Hiroaki Kitano et al. “Robocup: The robot world cup initiative”. In: *Proceedings of the first international conference on Autonomous agents*. 1997, pp. 340–347.

[KTR21] Soheil Khatibi., Meisam Teimouri., and Mahdi Rezaei. “Real-time active vision for a humanoid soccer robot using deep reinforcement learning”. In: *Proceedings of the 13th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART,* INSTICC. SciTePress, 2021, pp. 742–751. ISBN: 978-989-758-484-8. DOI: 10.5220/0010237307420751.

[KW14] Diederik P. Kingma and Max Welling. “Auto-Encoding Variational Bayes”. In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. 2014. arXiv: http://arxiv.org/abs/1312.6114v10 [stat.ML].

[Mah+19] Hamed Mahmoudi et al. “Mrl team description paper for humanoid kidsize league of robocup 2019”. In: *Mechatronics Research Lab, Department of Computer and Electrical Engineering, Qazvin Islamic Azad University, Qazvin, Iran* (2019).

[Mat15] Matias Mattamala. “A dynamic and efficient active vision system for humanoid soccer robots”. In: *Robot Soccer World Cup*. Springer. 2015, pp. 316–327.

[Mic04] Olivier Michel. “Cyberbotics Ltd. Webots™: professional mobile robot simulation”. In: *International Journal of Advanced Robotic Systems* 1.1 (2004), p. 5.

[Mni+14] Volodymyr Mnih et al. “Recurrent models of visual attention”. In: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. Ed. by Zoubin Ghahramani et al. 2014, pp. 2204–2212. URL: https://proceedings.neurips.cc/paper/2014/hash/09c6c3783b4a70054da74f2538ed47c6-Abstract.html.

[Mni+15] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *nature* 518.7540 (2015), pp. 529–533.

[PB21] Martin Poppinga and Marc Bestmann. “DSD-Dynamic Stack Decider”. In: *International Journal of Social Robotics* (2021), pp. 1–11.

[Qui+09] Morgan Quigley et al. “ROS: an open-source Robot Operating System”. In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, p. 5.

[Raf+21] 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.

[RBH07] Constantin A Rothkopf, Dana H Ballard, and Mary M Hayhoe. “Task and context determine where you look”. In: *Journal of vision* 7.14 (2007), pp. 16–16.

[Rol15] Martin Rolfs. “Attention in active vision: A perspective on perceptual continuity across saccades”. In: *Perception* 44.8-9 (2015), pp. 900–919.

[Rui+07] Javier Ruiz-del-Solar et al. “UChile Kiltros 2007 team description paper”. In: *RoboCup 2007 Symposium*. 2007, pp. 9–10.

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[SBB18] Daniel Speck, Marc Bestmann, and Pablo Barros. “Towards real-time ball localization using cnns”. In: *Robot World Cup*. Springer. 2018, pp. 337–348.

[Sch+16] John Schulman et al. “High-dimensional continuous control using generalized advantage estimation”. In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2016. URL: http://arxiv.org/abs/1506.02438.

[Sch+17] John Schulman et al. “Proximal Policy Optimization Algorithms.” In: *CoRR* abs/1707.06347 (2017). URL: http://dblp.uni-trier.de/db/journals/corr/corr1707.html#SchulmanWDRK17.

[SLR10] Andreas Seekircher, Tim Laue, and Thomas Röfer. “Entropy-based active vision for a humanoid soccer robot”. In: *Robot Soccer World Cup*. Springer. 2010, pp. 1–12.

[TET12] Emanuel Todorov, Tom Erez, and Yuval Tassa. “Mujoco: A physics engine for model-based control”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 5026–5033.

[Ver+16] Tom Verstraten et al. “Series and parallel elastic actuation: Impact of natural dynamics on power and energy consumption”. In: *Mechanism and Machine Theory* 102 (2016), pp. 232–246.

[Vil+14] Claudio O Vilao et al. “A single camera vision system for a humanoid robot”. In: *2014 Joint Conference on Robotics: SBR-LARS Robotics Symposium and Robocontrol*. IEEE. 2014, pp. 181–186.

[Zha00] Zhengyou Zhang. “A flexible new technique for camera calibration”. In: *IEEE Transactions on pattern analysis and machine intelligence* 22.11 (2000), pp. 1330–1334.

[Zho+19] Yi Zhou et al. “On the continuity of rotation representations in neural networks”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 5745–5753.

Online

[1] *bitbots_head_behavior*. URL: https://github.com/bit-bots/bitbots_behavior/tree/master/bitbots_head_behavior (visited on 01/14/2022).

[2] *DYNAMIXEL All-in-one Smart Actuator*. URL: https://www.robotis.us/dynamixel/ (visited on 01/14/2022).

[3] *ONNX - Open Neural Network Exchange*. URL: https://onnx.ai/ (visited on 01/11/2022).

[4] *REP 120 – Coordinate Frames for Humanoid Robots (ROS.org)*. URL: https://www.ros.org/reps/rep-0120.html#base-footprint (visited on 01/14/2022).

[5] *RoboCup Federation official website*. URL: http://www.robocup.org/ (visited on 01/11/2022).

[6] *RoboCup Soccer Humanoid League laws of the game 2019/2020*. URL: https://humanoid.robocup.org/wp-content/uploads/RCHL-2020-Rules-Dec23.pdf (visited on 12/03/2021).

[7] *Virtual RoboCup Soccer Humanoid League laws of the game 2020/2021*. URL: https://cdn.robocup.org/hl/wp/2021/05/V-HL21_Rules_v3.pdf (visited on 12/03/2021).

[8] *What is X3D? — Web3D*. URL: https://www.web3d.org/x3d/what-x3d (visited on 01/11/2022).

[9] Oren Ben-Kiki, Clark Evans, and Ingy döt Net. *YAML Ain't Markup Language (YAML™) version 1.2*. URL: https://yaml.org/spec/1.2.2/ (visited on 01/11/2022).

[10] Mega juice. *Reinforcement learning diagram*. URL: https://commons.wikimedia.org/wiki/File:Reinforcement_learning_diagram.svg (visited on 12/03/2021).

[11] Ioan Sucan and Jackie Kay. *URDF - ROS Wiki*. URL: https://wiki.ros.org/urdf (visited on 01/11/2022).