Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

MASTERTHESIS

Footstep Planning using Reinforcement Learning for Humanoid Robots

vorgelegt von

Jasper Güldenstein

MIN-Fakultät

Fachbereich Informatik

Studiengang: Master Informatik

Matrikelnummer: 6808089

Abgabedatum: 17.08.2022

Erstgutachter: Prof. Dr. Jianwei Zhang

Zweitgutachter: Dr. Philipp Allgeuer

Betreuer: M.Sc. Marc Bestmann

Acknowledgments

I would like to thank my advisors for their guidance and ideas during the development of this thesis, as well as for their feedback on this document. Furthermore, I would like to thank all members of the Hamburg Bit-Bots as they developed much of the underlying software this thesis uses. Especially, I would like to thank Florian Vahl for insightful discussions and advice, as well as his work on the baseline approach, and Timon Engelke for maintaining the infrastructure used in this thesis and for helpful hints for debugging some programming issues.

Abstract

Navigation is a crucial component of any mobile robot. For humanoid robots, this task is more challenging than for wheeled robots. Their capabilities can not be modeled by the velocity and acceleration limits used for wheeled navigation, as the discrete nature of taking steps interferes with the assumption that acceleration is possible at any time. Footstep planning can provide a viable alternative. It enables the robot to step onto or over obstacles and navigate more precisely as it considers individual footstep placement. However, modeling the capabilities of where a robot can place its feet while remaining stable given a specific walking engine and the robot's dynamic state is challenging. This thesis proposes using a model-free reinforcement learning approach to train a policy that outputs the footstep poses for a walking engine. This allows learning the capabilities of the robot and the walking engine to minimize the time required for navigation.

The policy is trained and evaluated in simulation. However, the employed walking engine has been used successfully on several real robots, which could ease the sim-to-real transfer. The evaluation shows a significant performance increase compared to velocity-based navigation while still being robust to noisy measurements.

Zusammenfassung

Navigation ist eine entscheidende Komponente jedes mobilen Roboters. Bei humanoiden Robotern ist diese Aufgabe komplexer als bei Robotern auf Rädern, denn ihre Fähigkeiten der Fortbewegung können nicht ausreichend durch die Geschwindigkeits- und Beschleunigungsgrenzen modelliert werden, die für die Navigation von Robotern auf Rädern verwendet werden. Bei humanoiden Robotern ist eine Beschleunigung nicht jederzeit möglich, weil die diskrete Natur der Schritte dies beeinträchtigt.

Schrittplanung kann eine brauchbare Alternative darstellen. Sie ermöglicht es dem Roboter auf oder über Hindernisse zu treten und genauer zu navigieren, da die Platzierung der einzelnen Schritte berücksichtigt wird. Eine Herausforderung besteht darin, die Menge der Fußschritte zu modellieren, die für einen Roboter und einen Laufalgorithmus stabil sind, da diese abhängig vom jeweiligen Zustand, z. B. von der Geschwindigkeit des Roboters sind.

Diese Arbeit beschreibt einen modellfreien Reinforcement-Learning-Ansatz. Dieser wird verwendet, um ein neuronales Netz zu trainieren, das die Schrittpositionen für einen Laufalgorithmus ausgibt. Der Ansatz erlaubt das Lernen der Fähigkeiten des Roboters und der Menge der stabilen Schritte des Laufalgorithmus', um die für die Navigation benötigte Zeit zu minimieren.

Für das Training des neuronalen Netzes und für die Evaluation wird Simulation verwendet. Der genutzte Laufalgorithmus wurde jedoch bereits erfolgreich auf mehreren realen Robotern eingesetzt, was den Transfer von der Simulation zur Realität erleichtern könnte. Die Auswertung zeigt eine signifikante Steigerung der Leistung im Vergleich zur geschwindigkeitsbasierten Navigation. Gleichzeitig erweist sich der Algorithmus als robust gegenüber verrauschten Messungen.

Contents

1. Introduction

Humanoid robots have shown remarkable advances in the last decades. Incorporating them into the human-made environment instead of modifying the environment to suit the robot's needs is one of the key motivations driving development.

Generally, robots are built to assist humans in performing repetitive, tedious, or hazardous tasks. The structure of a humanoid robot allows it to perform much more tasks than a robotic arm or wheeled robot could but also brings with it many challenges. Performing these tasks is possible for several reasons: Firstly, the robot can change the location of its workspace by navigating, allowing it to perform tasks in different locations or tasks which require switching of the workspace (e.g., taking dishes from a table and placing it in a dishwasher). While wheeled robots equipped with a robotic arm may also perform many of these tasks, their navigation is limited by obstacles such as stairs or uneven terrain. Secondly, a humanoid robot can change its whole body pose. This allows it to extend its workspace further, for example, by kneeling down to perform tasks on the ground or by holding onto a handrail to allow it to reach further while remaining stable. Lastly, as the humanoid robot has two arms, it can generally perform more complex tasks than would be possible with a single manipulator. The challenges a humanoid robot brings are significantly increased complexity in hard- and software and therefore cost and development effort.

One of the main challenges for mobile robots is navigation. They need to localize in their environment and navigate while avoiding obstacles. While the navigation problem for wheeled mobile robots is a well-established and successful research area, humanoid navigation has not been explored to this extent.

The navigation of a humanoid robot is significantly more complex. However, it is necessary to perform navigation on a humanoid robot as, without it, the humanoid would be easily replaceable by a pair of stationary robotic arms.

Mobile robots usually use velocity control for navigation. This approach can be applied to humanoids by calculating the next footstep pose based on a velocity command and the time the step is calculated to take. However, the approaches used in mobile robot navigation usually assume that an acceleration can be applied at any time. A humanoid robot is not capable of this since the base footprint, the usually employed navigational reference frame [1], depends on the pose of footsteps which is controlled by the discrete action of taking a step. Furthermore, stability criteria must be taken into account. This is a significant problem when the robot approaches the navigation goal as it may overstep it. A strategy to reduce this is severely limiting the commanded velocity when close to the goal. However, this increases the time required for navigation.

Footstep planning, i.e., commanding the poses of the robot's feet on the ground, on the other hand, allows precise footstep placement and overcomes this problem. This planning method can be performed in various ways, which will be described in Chapter 2.

In recent years significant progress has been made in reinforcement learning (RL). This method of machine learning is particularly suited if the best solution to a problem is not known in advance, but only a reward function, expressing how well an agent is doing, can be formulated. Deep RL, i.e., RL, which uses a neural network as the policy or value function, has recently become a viable solution to previously intractable problems.

RL can be applied to the problem of humanoid navigation as it is challenging to find an optimal sequence of commands to give to the robot to navigate from a starting position to a goal efficiently (i.e., minimizing time and falls). A reward function of how well a robot performs this task is easy to formulate as it could simply be the time required to reach the navigation goal and a penalty for falling. However, in practice, a more complex function is chosen to guide the RL process.

Controlling the motion of a humanoid robot can be achieved in a variety of different ways. In this thesis, a walking engine based on quintic splines [2] is chosen. An RL-trained policy generates footstep poses for controlling it. The policy receives information about the navigation goal and about obstacles in the environment. Falling is penalized and reaching the goal quickly is rewarded to encourage the policy during training to find stable but fast trajectories. The approach is trained and evaluated in a simulated environment based on the RoboCup Humanoid League Virtual Season, further described in the following section.

As RL is relatively sample inefficient, learning the complete navigation problem on a real robot may be infeasible. The approach in this thesis uses a walking engine that has been successfully implemented on real robots. Therefore, the hypothesis is that transferring it to the real world requires less effort.

1.1. RoboCup

This thesis is developed in the context of a RoboCup team, the Hamburg Bit-Bots at the University of Hamburg. The RoboCup competition was developed to promote the research on various types of robots through competition [3]. Successes in the field of artificial intelligence (e.g., beating humans in chess) motivated the founders of the RoboCup to create standard competitions for robotics. The goal of the initiative is to beat the world champion team of human soccer players by the middle of the century.

In the Humanoid Kid Size League, in which the Hamburg Bit-Bots participate, two teams of four robots compete in a soccer match autonomously against each other. Soccer was chosen as a competition since it requires many of the skills that humans use to fulfill tasks and because of its appeal and tangibility to the public. These skills include the detection and localization of objects, self-localization and navigation, team communication and strategy, omnidirectional walking, and many complex motion tasks such as standing up after falling and manipulating a ball.

The robots in the Humanoid League Kid Size can be between 0.4 m and 1.0 m tall and must comply with several rules to ensure their proportions are human-like. They play on a 6 m by 9 m field made from artificial turf.

Efficient navigation is one of the critical tasks as the robot that reaches the soccer ball first or gets quickly into a defending position has a crucial advantage over its opponent.

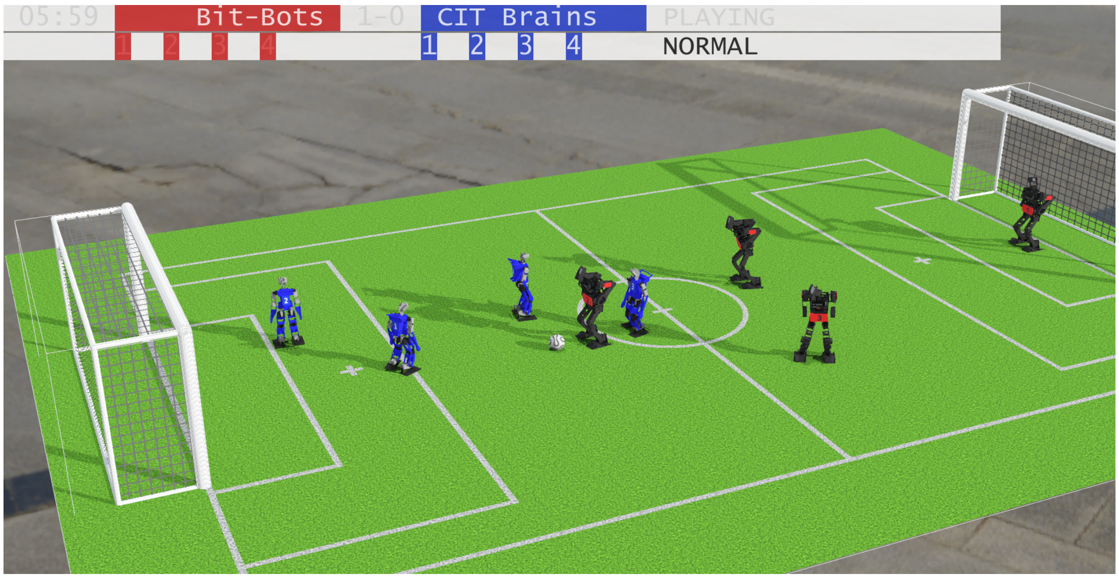


Figure 1.1.: Typical scene from the Humanoid League Virtual Season. Four robots per team compete in the simulated soccer competition based on the rules of the RoboCup Humanoid League. Only simulated sensor information and team communication can be used to perceive the environment. The robot platform playing in red is used in this thesis.

Navigating precisely is also favorable since it simplifies the task of kicking the ball accurately. However, non-planar navigation is not required since the field is flat. Obstacle avoidance is essential to efficiently navigate around the other robots on the field to prevent collisions, falls, and fouls.

The Hamburg Bit-Bots play with the Wolfgang-OP robot [4]. It is an 0.8 m tall robot with 20 degrees of freedom (DOF). Its hard- and software are fully open source. The software stack has proven its viability in the virtual competition held for the 2021 RoboCup and the Humanoid League Virtual Season 2022, achieving third and second place, respectively.

The software described in this thesis is based upon this software stack, specifically the walking engine and simulation models. Furthermore, a performance comparison to the current approach used by the Hamburg Bit-Bots for navigation will be made.

1.1.1. Humanoid League Virtual Season

A virtual competition using the Webots simulator [5] was developed for the RoboCup 2021. This competition was continued throughout 2021 and 2022 as the Humanoid League Virtual Season (HLVS) [6]. It features biweekly matches between participating teams which allows to test and adapt strategies or evaluate new approaches. The HLVS is planned to continue for the coming years with adaptations to the simulation to increase realism (e.g., domain randomization). Figure 1.1 shows a typical scene from the competition.

1.2. Research Questions

To guide the development and evaluation of this approach, two research questions are considered:

Can an RL-trained policy plan more efficient footstep plans than velocity planning or classical footstep planning? The hypothesis for this question is that an RL-trained policy may exploit the capabilities of the walking engine more efficiently than a velocity or classical footstep planner would and therefore be able to reach the given goal pose more quickly.

Are the footstep plans generated by the policy more stable (the robot falls less often) than velocity planning or classical footstep planning? Velocity or classical footstep planning usually does not consider the robot's state other than simple acceleration limits. An RL policy could learn the limits of a walking engine more accurately and reduce the number of falls.

1.3. Structural Outline

This thesis is structured as follows: Chapter 2 describes state-of-the-art approaches to navigation with a focus on legged robots and learning-based techniques. The fundamentals the approach relies upon are outlined in Chapter 3. The baseline against which the approach will be compared is presented in Chapter 4. In Chapter 5 the approach itself is described. Chapter 6 proposes metrics and evaluates the approach compared to the baseline. The results of the evaluations and limitations of the approach are discussed in Chapter 7, and a conclusion is drawn in Chapter 8. Possible future work is outlined in Chapter 9.

2. Related Work

Many approaches attempting to solve the navigation problem on legged robots have been proposed. This chapter gives an overview of several approaches related to the one presented in this thesis. Therefore, an emphasis is put on methods that either use footstep planning or machine learning.

This chapter is structured as follows: Classical methods (i.e., methods not based on machine learning) are described in Section 2.1. Learning-based footstep planning approaches, where the output of the learned policy is a single footstep of footstep plan, are presented in Section 2.2. Section 2.3 describes more general learning-based techniques for navigation of legged robots.

2.1. Classical Footstep Planning

Classical footstep planning is defined here as solving the footstep planning task by using algorithmic methods not based on machine learning. Perrin [7] classifies the approaches into several categories: Firstly, the problem can be formulated using finite transition sets. These will be described in 2.1.1. Secondly, tiered strategies which divide the planning phase into a high and a low-level phase are briefly described in 2.1.2. Optimization techniques solve the footstep planning problem by defining the footstep pose as an optimization parameter of the walking controller are described in 2.1.3.

2.1.1. Finite Transition Sets

Approaches based on finite transition sets discretize the footsteps a walking engine can perform. From this, a transition set (or graph) can be constructed (where nodes respond to possible states and transitions to taking a single footstep). Graph traversal algorithms search for a path from the starting state to the goal state.

Chestnutt et al. [8] propose an approach that traverses the transition set of footsteps using A*. They managed to plan and execute a path through an environment with both static and predictable dynamic obstacles.

Replanning is often required as the walking engine does not perform the planned footsteps exactly as the model predicts or the environment changes. Garimort et al. [9] propose using D* Lite instead of A* as a search algorithm. It plans from goal to start state and allows reusing a previously calculated plan. Hornung et al. [10] expand upon the evaluation of search algorithms by exploring Anytime Repairing A* (ARA*) and randomized A* (R*).

All presented approaches use graph traversal algorithms which require heuristics to guide the search. All heuristics include the cost of taking a step combined with a distance metric.

The cost of taking a step includes a cost based on the distance to the obstacles in the environment. The heuristic used in Chestnutt et al. [8] uses the remaining distance to the goal pose calculated by a standard planner for a wheeled mobile robot. Garimort et al. [9] evaluate three heuristics: Euclidean distance + expected number of steps, Euclidean and angular distance + expected number of steps, and a similar approach to Chestnutt et al. [8] combined with the expected number of steps. All these heuristics lead to finding the optimal (in respect to the defined possible successor footstep set) path for the given environments. Hornung et al. [10] also evaluate Euclidean distance and a standard approach for wheeled robots as a heuristic. Using the Euclidean distance heuristic lead to significantly more states of the graph being explored, which leads to a long planning time.

Chestnutt et al. [11] expand upon the approach of using a discrete set of footsteps by adapting the set of possible steps based on the position of obstacles. This allows the planner to perform better in cluttered environments or with narrow passages.

While the first approach discussed in this chapter bases its transition set on the previously commanded actions to the walking engine, the dynamics of the system are only indirectly taken into account. Schleich et al. [12] propose a method to inform the footstep planner about the possible footstep placements. These constraints are calculated using the state of the center of mass (i.e., its position and linear velocity). In contrast to this, the approach proposed in this thesis learns these constraints model free.

2.1.2. Tiered Strategies

The idea of a tiered planner is to separate the planning into two or more stages of planning which are progressively more complex but utilize the solution from the previous planner to guide the search. The earlier stages of a tiered planner use more simplified approximations of the robot and can be as simple as a bounding box for representing the robot's dimensions, such as described by Yoshida et al. [13].

Chestnutt et al. [14] describes a three-tiered planner for bipedal navigation. Firstly, a global search disregarding motion constraints is performed. Then a mobile robot planner uses a subgoal from the global plan to act as a heuristic for a low-level planner which searches the finite transition set of footsteps.

Another tiered approach is described by Gutmann et al. [15]. Two bounding cylinders are used for approximating whether a state is collision-free. The resulting states are then searched for a path.

The problem of collision checking is solved by pre-processing with a volume approximation by Perrin et al. [16]. This reduces the online processing time and, therefore, considers more transitions. The search is also performed using a Rapidly-exploring Random Tree (RRT) [17] as opposed to previously mentioned approaches.

More recently, a hierarchical structure that refines the level of detail of the approximation of the planning algorithm has been proposed by Wermelinger et al. [18]. This allows a global plan to be calculated more efficiently by increasing the level of detail only when required (i.e., no path can be found or the path is invalid due to model simplifications).

Styler et al. [19] use seven levels of model detail for navigation with a wheeled mobile robot. This approach is expanded upon for a quadruped robot by Brandao et al. [20]. Further, they expand the approach by including the computation-time budget for each transition in the state model. Their main contribution is the introduction of using evolutionary optimization to learn the state transition costs and the computation-time budget for each transition.

2.1.3. Optimization Based Footstep Planning

Instead of separating the problem of finding a feasible footstep plan and executing it, the approaches described in this section solve the whole motion planning as an optimization problem. Many approaches using optimization as a method of generating a walking gait use a predefined footstep plan [21], here only the ones that generate their own footstep poses are considered.

Adapting footsteps to increase the stability of a walking gait is introduced by Diedam et al. [22]. However, this approach still requires a footstep plan and only modifies the footstep poses in this plan.

The approach presented by Herdt et al. [21] includes the footstep position as an optimization variable and only requires the desired mean velocity as an input. However, since regions need to be convex for efficient optimization, the yaw of the robot and its feet is not optimized.

An approach that overcomes this challenge is presented by Kuindersma et al. [23]. They efficiently decompose the feasible terrain into multiple convex regions. This allows the approach to incorporate the yaw of the footstep pose.

Bledt et al. [24] show how Regularized Predictive Control can be used to generate a gait for a quadruped robot. The footstep positions are part of the optimization problem and are chosen by the optimizer to maximize stability while the robot is tracking a given velocity command.

2.2. Learning-Based Footstep Planning

Learning-based footstep planning is here defined as a machine learning algorithm that has as its final or intermediate output the poses of one or multiple footsteps.

An approach using RL for the full navigation stack is described by Peng et al. [25]. The presented approach splits the problem into a low-level controller (walking engine) and a high-level controller (footstep planner). A limitation to this approach is that it works in physics-based rendering as opposed to a full physics simulation and that the navigation obstacle representation requires ground truth knowledge of the environment.

Tsounis et al. [26] also splits the problem into two steps but for a quadruped robot. Here, a high-level planner, named gait controller, performs the task of footstep planning by generating a phase plan consisting of state vectors describing the robot's planned configuration. The low-level controller then tries to execute this phase plan. However, this approach is model-based and developed for a quadruped.

The approach presented by Gangapurwala et al. [27] uses RL to generate a footstep plan which is then executed by a motion controller (walking engine) based on optimal control. Their work focuses on finding the correct poses for the feet given a velocity goal rather than finding a path through an environment. Another promising approach explored in this paper is the usage of an actuator neural network to model real-world behavior more closely, which allowed for sim-to-real transfer.

Yu et al. [28] present an approach that processes depth images to generate the desired CoM state and swing feet targets. A low-level controller executes these. However, the environments the robot was trained and evaluated in do not require navigation in a plane (i.e., two-dimensional position and angle), but rather only forward movement with stepping over gaps or onto stairs.

Missura and Behnke [29] present a model-based approach to adjusting footstep placements to maintain balance while keeping the length of the steps close to a reference. Their approach learns online in very few examples due to it being model-based and an approximation of the policy's gradient being provided by the model.

2.3. Learning-Based Navigation

Several approaches for solving the humanoid navigation problem are proposed and evaluated by Hofer and Rouxel [30]. Their best-performing method is based on RL with a regression forest (also referred to as random forest) as a policy and also as a value function. The action space is defined as the acceleration of the walking speed in x, y, and θ direction. The state space is defined as the distance and orientation to the target, the target orientation, and the current velocities in x, y, and θ direction. Furthermore, they evaluate several models to account for the inaccuracy of placed footsteps in the real world.

Visual navigation for a humanoid robot in the RoboCup context is presented by Lobos et al. [31]. The action space of the policy is the velocity commanded to the walking engine. However, the policy does not take measurements of the motion sensors of the robot into account and therefore has to be conservative with its commanded velocities to guarantee stability.

A similar approach is presented by Brandenburger et al. [32]. Visual information is directly translated into command velocities for the motion engine (more specifically, the change in command velocity). As opposed to the previously presented approach, the information of the motion sensors is included in the state representation. They manage to transfer their policy onto a real robot and navigate through simple environments.

Li et al. [33] present a hierarchical approach using a low-level controller which is able to perform several motion primitives (moving forward, turning left and right, and moving backward) and a high-level controller which selects one of these primitives to reach a given goal. The high-level controller is model-based and learns the transition model of the primitives. They demonstrate their approach on a hexapod robot in the real world.

Another approach using hierarchical RL for legged locomotion is described by Jain et al. [34]. They apply this technique to a path-following task for a quadruped robot. They train the high-level and the low-level policy in conjunction.

3. Fundamentals

This chapter describes the fundamental concepts and algorithms on which the work in this thesis is built. Firstly, an overview of RL is presented in Section 3.1. Section 3.2 describes the employed robot platform, the Wolfgang-OP. Section 3.3 presents the walking engine used on the robot and in this thesis. Motivated by the modeling of noise for the approach, Section 3.4 describes the localization approach used by the Hamburg Bit-Bots RoboCup team. Section 3.5 gives an introduction to the fundamentals of rigid body simulation.

3.1. Reinforcement Learning

This Section is based on Reinforcement Learning: An Introduction [35] unless otherwise cited. In RL the problem to be solved is usually formulated as a Markov decision process (MDP). An MDP is a probabilistic model of how a system behaves depending on the actions of an agent. Each state of the system has a possibly infinite set of transitions leading to successor states. The probability of each transition is determined not only by the system but also by the action the agent takes in this state.

In RL a problem is usually formulated as: An agent performs an action in an environment. The environment provides an observation and a numerical reward signal. The agent's goal is to maximize the sum of rewards over a finite number of steps (an episode). Figure 3.1 visually represents the interaction between the agent and the environment.

Contrary to supervised learning, the policy the agent uses to decide its action receives no example solution. Instead, it must find suitable strategies through exploration to maximize the reward. This has the advantage that RL can be employed in situations where the strategy required to solve a problem is unknown. However, RL is usually less sample efficient, meaning

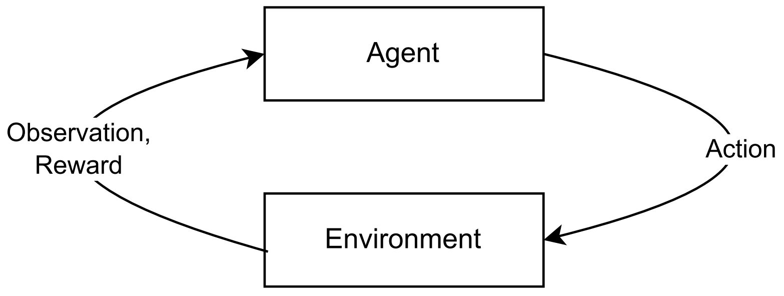


Figure 3.1.: Visual representation of the interaction between the agent and the environment in RL modeled as an MDP.

more training samples are required since the strategy needs to be extracted from the samples of the interactions between the agent and the environment.

The goal of the RL is to maximize the reward signal over an episode, i.e., a fixed or variable number of steps. To achieve this, the agent performs exploration (i.e., trying out random actions) during the training phase. The experiences collected through interaction with the environment can be used to approximate a value function $V^\pi(s)$.

$$V^\pi(s) = E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\}$$

The value function estimates the discounted rewards the agent will obtain in the successor states by applying its policy. π is the agent's policy, s is the state of the environment, γ is the discount factor in $(0, 1)$ applied to ensure the value function is bounded. r_t describes the reward an agent receives at timestep t.

The collected experiences may also be used to approximate an action-value function $Q^\pi(s, a)$ which maps a state-action pair to an estimated future reward.

A classic example of a problem that RL can solve is the cart-pole problem [36] pictured in Figure 3.2. The goal of the agent is to keep the pole upright. It takes a discrete action of moving the cart either to the left or the right. It receives the pole's angle and angular velocity as well as its own position and velocity as an observation. While the pole is upright, the agent receives a reward of $+1$ in each timestep. When the pole's angle deviates more than $12°$ from its upright position, the cart is more than $2.4m$ away from its starting position, or a maximum episode length is reached, the episode is terminated. The system fulfills the Markov property, i.e., the probability distribution of the next state of the system only depends on the present state of the system and the agent's action.

The mapping of observation to action (i.e., the policy) is a mathematical function. This function can be represented in many ways depending on the action and state space. If both are finite, a simple two-dimensional table can approximate the action-value function with a row for each state and a column for each action. A policy would then choose the maximum value in a row. However, interesting problems are often continuous-valued and therefore not finite. Neural networks have been proposed to represent the value or action-value and policy function. Methods using these are classified as deep RL. Adapting the parameters of neural networks in order to be good function estimators must be done efficiently as there are too many parameters for a brute force approach.

Many algorithms have been proposed to solve this problem. They are commonly classified into on-policy and off-policy algorithms. On-policy algorithms use the target policy to collect samples, whereas off-policy algorithms use a behavior policy to collect samples for training. As the value or action-value function does not depend on the policy in off-policy learning, algorithms can reuse collected samples. However, in on-policy learning, the collected samples are more relevant to the actions an agent actually selects.

The algorithm used in this thesis is proximal policy optimization (PPO) which the following Section describes. It is an on-policy algorithm.

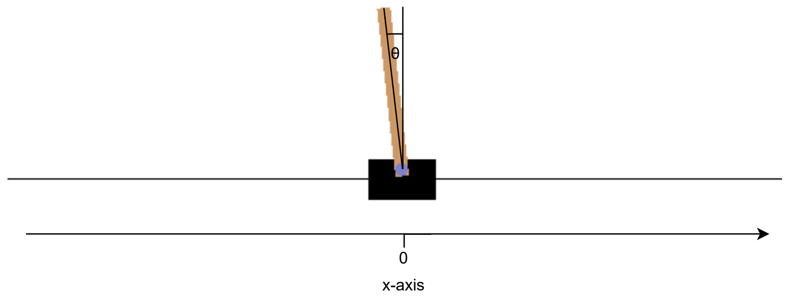


Figure 3.2.: Cart-pole environment, a classic example problem for RL. The agent can move the cart either left or right and observe its position x and velocity $\dot{x}$ as well as the poles angle θ and angular velocity $\dot{\theta}$. Annotated rendering of the OpenAI gym implementation [37].

3.1.1. Proximal Policy Optimization

Proximal policy optimization (PPO) [38] is a policy gradient method, meaning it estimates the gradient of the policy towards a state which increases the reward. It operates on a batch of collected experiences (i.e., samples of interactions between the agent and the environment) to stochastically approximate this gradient. Like its predecessor trust region policy optimization (TRPO) [39], it uses a clipping function to limit the change in the policy's parameters. Clipping is applied, as excessively changing the policy based on a stochastic estimate of the gradient can cause performance deterioration. As the policy is used to collect further samples, degrading its performance to a state where no valuable experiences can be collected is detrimental. Furthermore, the experiments performed by Schulman et al. [38] show that the algorithm performs reasonably well without tuned hyperparameters and in continuous control tasks.

3.2. Wolfgang-OP

The Wolfgang-OP [4] is the open source and open hardware platform developed and used by the Hamburg Bit-Bots RoboCup team. Figure 3.3 shows the 20 DOF humanoid robot, its kinematic model, and its simulation model. Its hardware is based upon the Nimbro-OP [40] but has been heavily modified.

Section 3.2.1 will briefly describe the robot platform's hardware. The software stack developed for the robot by the Hamburg Bit-Bots is outlined in Section 3.2.2.

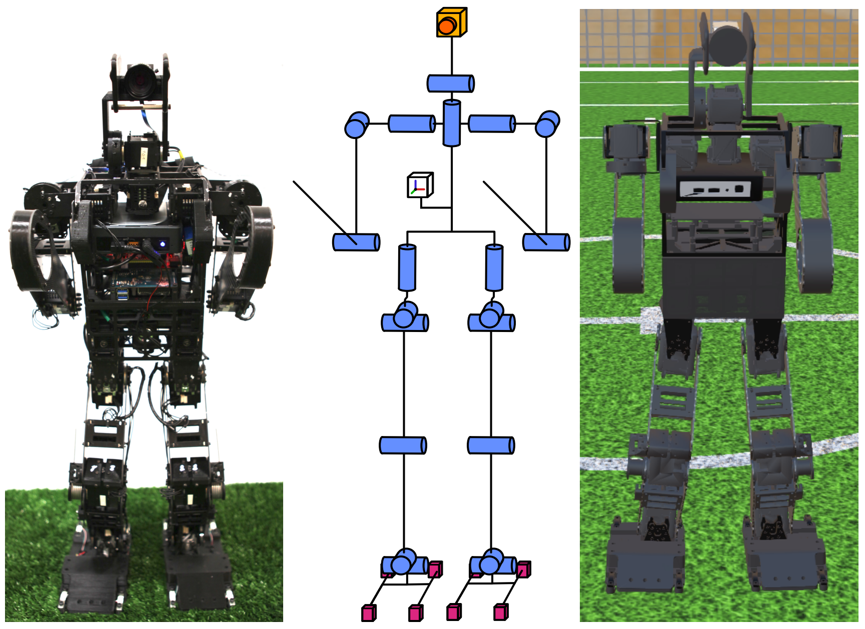


Figure 3.3.: Wolfgang-OP in reality (left), kinematic model and sensor placement (center), and its simulated model in Webots [5] (right). The left and center images were taken from [4].

3.2.1. Hardware

The Wolfgang-OP robot features 20-DoF kinematic model typical in small-size humanoid robots. It measures $0.8\,\mathrm{m}$ in height and weighs $7.5\,\mathrm{kg}$. Each leg is equipped with six actuators allowing it to control the position and orientation of its feet fully independently. Three servo motors each control the arms, which are primarily used for standing-up motions after the robot falls. The head is equipped with two actuators to allow the robot to control the pan and tilt of the camera mounted on the head.

An inertial measurement unit (IMU) is mounted to the torso. The iteration after the version presented by Bestmann et al. [4] also features an IMU in the head. Foot pressure sensors allow the robot to detect contact forces with the ground. While the servo motors do not feature a torque sensor, they measure the consumed current, which can be assumed to be proportional to the applied torque.

Robustness is critical in the RoboCup competition in which the robot is used as falls and collisions are frequent. Compliant elements at the shoulder pitch and the head tilt motors prevent damage to the gearboxes when the robot falls. As the knee experiences the highest load during typical operation [4], a torsion spring is used to reduce the maximum torque it has to exert.

3.2.2. Software

The software stack the Hamburg Bit-Bots RoboCup team has developed allows the robot to perform the tasks required for autonomously playing in the soccer competition. A state-of-the-art vision pipeline [41] detects objects and segments the captured images into the playing field and field lines. These features are transformed into Cartesian space using inverse projection mapping. Particle filter localization described in Section 3.4 localizes the robot in the playing field. A behavior approach based on the Dynamic Stack Decider architecture [42] uses the robot's perceptions and information communicated with other robots to decide which action to take. The pathfinding and execution it controls are further described in Chapter 4 as they are the baseline against which the approach presented in this thesis is evaluated. Three motion generation engines control the robot's actuators. The *Quintic Walk* described in Section 3.3 generates a walking pattern. *DynUp* [43] allows the robot to efficiently recover from falls. The *Dynamic Kick* [44] allows kicking the soccer ball using a dynamically generated spline based on the desired direction of the kicking motion. An abstraction layer, the Humanoid Control Module [45] performs basic reflexes and multiplexes between the motion engines. The described architecture is pictured in Figure 3.4.

ROS2 is used as a middleware to allow for modular design, interchangeable components, and the use of existing visualization tools. The software stack is available fully open source [46].

3.3. Quintic Walk

Bipedal legged locomotion has several advantages and drawbacks compared to wheeled locomotion. A legged robot must have multiple actuators to move its limbs, leading to increased cost, weight, and complexity compared to wheeled robots. While simple velocity control can be used for a wheeled robot's motors, defining the movements of a legged robot's actuators which leads to stable and efficient locomotion, is more complex. Furthermore, bipedal walking is inherently unstable and the robot's movements must be adapted to remain balanced. However, legged locomotion is more flexible than wheeled locomotion as it allows stepping over obstacles, onto stairs, and through uneven terrain.

The method for controlling the robot's gait employed in this thesis is the *Quintic Walk* engine [2] which is based on the IKWalk engine [47]. Cartesian quintic splines of the flying foot and the torso relative to the support foot define the walking motion. Six splines are used each as they need to define both 3D position and rotation.

In the current implementation, two methods of controlling the walking engine are possible. Firstly, a command velocity can be passed to the walking engine from which it calculates the goal pose of the flying foot (the last point in its spline) using the frequency of the walking engine. This method is used for the baseline described in Chapter 4. Secondly, the walking engine can directly receive the footstep poses that it executes. The approach proposed in this thesis uses the second method of control.

The splines are constrained by several parameters of the algorithm (e. g. the frequency with which the robot steps, the height of the flying foot, and the pitch of the torso). These parameters were optimized by Bestmann et al. [2] using different optimization algorithms

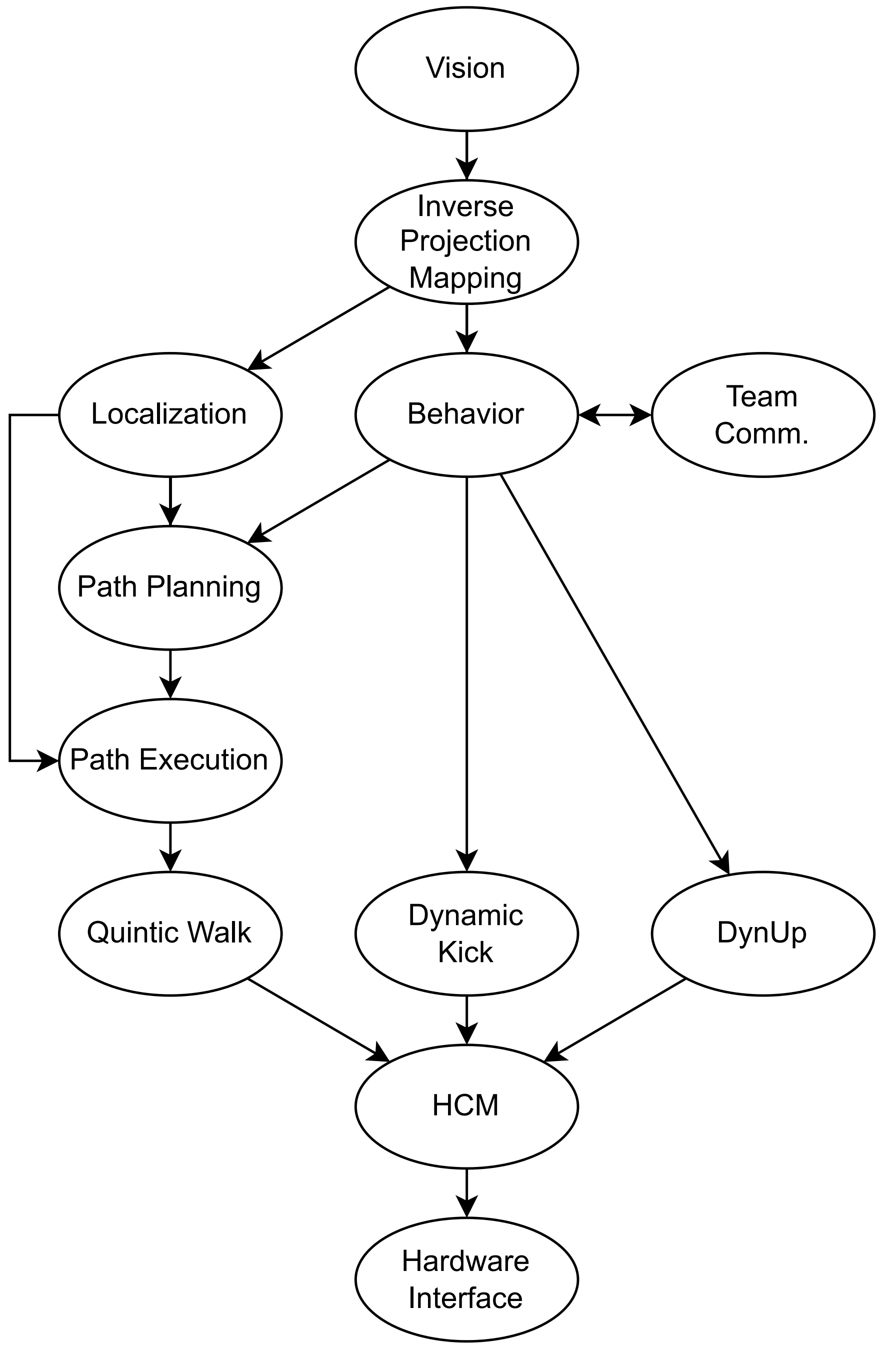


Figure 3.4.: Simplified software architecture of the Hamburg Bit-Bots RoboCup team.

implemented in the Optuna library [48].

The walking engine also provides stabilization techniques to react to perturbations in the walking cycle. Firstly, a PID controller can modify the torso's pitch and tilt based on the orientation inferred from an IMU. Secondly, when sensing that the robot's flying foot has made contact with the ground through foot pressure sensors or the actuator's torque measurements, a reset in the phase of the walking cycle can be triggered.

These stabilization techniques are disabled for the approach presented in this thesis as they may interfere with the policy. They were also not used in the optimization process of the walking parameters.

3.4. Localization

Localization is the process of finding the robot's pose given some information about the environment. A straightforward approach to solving this problem is using odometry, i.e., tracking the actions the robot has taken and integrating them over time (sometimes also referred to as dead reckoning). While this is usually accurate for small movements, the error in the odometry accumulates over time which usually leads to an unusable localization estimate.

The localization used by the Hamburg Bit-Bots on the Wolfgang-OP is based on particle filters. It was introduced in [49] and has been significantly changed since then. It is available fully open source [50].

The working principle of the localization algorithm is to combine the odometry with a measurement to update a set of particles with x, y, and θ coordinates from which an estimate of the robot's position is extracted. The odometry is generated by calculating the forward kinematics when both feet of the robot are touching the ground (i.e., double support phase) between the robot's feet. During the swing phase, the forward kinematics between the support foot and the navigation frame (i.e., the `base_footprint` according to REP120 [1]) is used to interpolate the odometry. The measurements are the lines on the RoboCup field detected by the vision system. Inverse projection mapping is applied to calculate their coordinates in Cartesian space. Figure 3.5 shows a visualization of the particles and measurements during localization.

Three steps are applied in the localization for each update. Firstly, the motion model, i.e., the odometry, is applied to the particles (e.g., if the robot moves forwards, all particles are moved forward in their respective direction). Secondly, the likelihood of each particle being the robot's pose is estimated by matching the line measurements with a map of the field. Lastly, the particles are resampled based on their likelihood. The localization estimate is the mean of the n most likely particles, where n is a tunable parameter of the algorithm.

An issue this algorithm faces is the delay in the measurement of the field lines introduced by the processing time of the vision algorithm and the inverse projection mapping. This can cause the estimate to lack behind the current state as the likelihood of the particle, and therefore the robot's localization estimate is calculated based on a delayed measurement. The localization is evaluated in Section 6.1, to create a noise model used in the proposed approach.

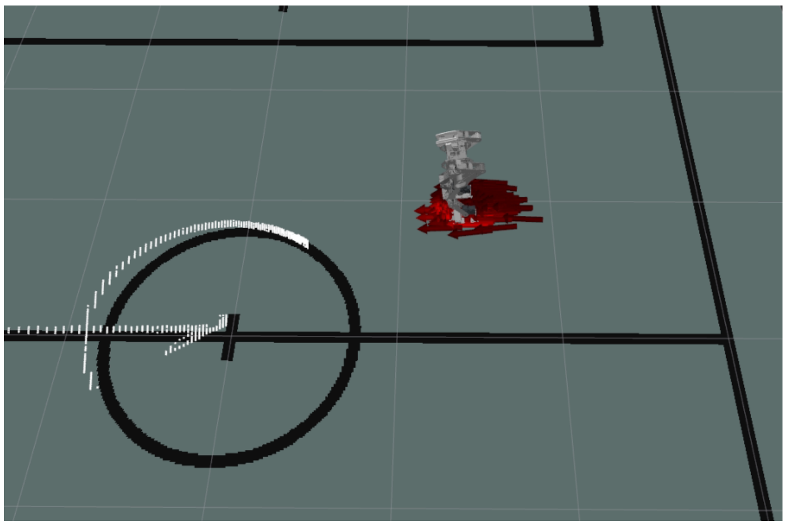


Figure 3.5.: Visualization of the Localization using Rviz [51]. The particles are shown as red arrows. Their brightness represents their likelihood of being the robot's pose. The field lines are depicted in black. The robot's measurements of the field lines are shown in white. Their resolution decreases with the distance from the robot as it is a mapping from image space to Cartesian space. Their accuracy also decreases due to the error of the robot's pitch angle measurement.

3.5. Simulation

Simulation is the modeling of a real-world process using computation. In robotics, this is usually achieved by modeling the robot and its environment as a rigid body system. In such a system, the mechanical parts are assumed to be rigid (i.e., not deformable) and may be connected by movable joints. The system's dynamics are calculated in discrete time steps.

The software stack of the Hamburg Bit-Bots team and this thesis use Webots [5] which employs a modified version of the Open Dynamics Engine (ODE) [52] to calculate the dynamics. An accurate model of the robot and the environment is required to allow accurate simulation of the physical behavior of the robot. This model includes the mass distribution of the robot's links, the torque that actuators apply, and the contact properties of surfaces (e.g., the artificial turf used in the RoboCup competition).

The simulation model of Wolfgang-OP described in Section 3.2 and pictured in Figure 3.3 was generated using its CAD model. Its link's mass distributions are therefore accurate to the tolerances of the manufacturing processes (i.e., CNC milling and 3D printing). However, the cables connecting the servo motors and several small parts such as screws are not modeled. Furthermore, the real robot's links may deform by wear or impact. This is also not modeled as it is not easily measurable.

The Wolfgang-OP robot model also includes a definition of the backlash each joint typically introduces to the kinematic chain on the real robot. It is modeled as a freely moving hinge joint between the output of the actuator and the subsequent link. Furthermore, the parameters of the actuator models (i.e., the torque output depending on velocity) are derived from the datasheets of the actual robot's motors.

The contact properties of the artificial turf used in RoboCup were studied for the virtual RoboCup 2021. The parameters have been used successfully in this competition and the Humanoid League Virtual Season (see Section 1.1).

Webots [5] and most simulators used in robotics simulate not only the dynamics of the system but also the robot's sensors (e.g., camera, IMU, or torque sensors). This allows for full simulation of the robotic system.

Simulation has many applications in robotics. Firstly, it is useful for safely (i.e., without harming the robot or its surroundings) testing and verifying new approaches. It is also usually less complicated to test in as a robotic system has many failure points that a simulation does not have. Setup and reset to an initial state can also be done automatically, eliminating the need for human intervention. Additionally, it is less costly as no wear on the robot occurs. Using modern processing capabilities allows for running multiple simulations in parallel.

The approach developed in this thesis is trained and evaluated in simulation as it lends itself to being used in machine learning applications due to the previously mentioned reasons. However, the simulation can not perfectly model the physical behavior of the robot as the robot's model is imperfect and the physics calculation is discretized. Transferring an approach from a simulated environment onto the real robot (sim-to-real transfer) can therefore require adaptation strategies. A strategy often employed is domain randomization, such as described by Muratore et al. [53] to bridge the reality gap. This changes the parameters of the simulation or the observation. The approach, therefore, has to be more robust to variable environments. Another approach is to bridge the reality gap using a transfer function between the simulated and real environment, such as the one described by Koos et al. [54].

4. Baseline

This chapter describes the functioning principle of the baseline this approach is evaluated against. It is based on the `navigation2` framework [55] with a custom local planner [50] and localization [49, 50], both implemented by the Hamburg Bit-Bots.

To perform the task of navigation, the framework manages a local and a global costmap that describe obstacles in the robot's navigation space. The costmaps are further described in Section 4.1. When a new navigation goal is requested, firstly, a global plan is generated based on the global costmap. Section 4.2 describes this further. The global plan is passed to a local planner described in Section 4.3 which generates a control command (e.g., a velocity or footstep command) for the robot. Figure 4.1 shows a visualization of the costmaps, local, and global plans. The walking engine described in Section 3.3 generates a suitable gait from the command. The framework allows performing recovery behaviors if the navigation does not succeed. Furthermore, the planning is recalculated at a configurable period.

The main advantage of the framework is its modularity and configuration possibility. Components (e.g., map, local or global planner) are interchangeable because of clearly defined interfaces. The behavior of the navigation system is easily reconfigurable using behavior trees [56].

4.1. Local and Global Costmap

The local and global costmap are used to keep track of obstacles in the robot's navigation space. Generally, the global costmap can include fixed obstacles (e.g., walls) as well as temporary obstacles (e.g., other robots). The local costmap can be a rolling window of the global costmap to facilitate the task of the local planner by reducing the size of the planning area. It may also include more precise or additional information compared to the global costmap. The `navigation2` framework allows for multiple types of costmaps for different robot and sensor setups. The Hamburg Bit-Bots use the *Obstacle Layer* which generates a map based on data provided by the robot's perception system or through communication with other robots. Furthermore, an *Inflation Layer* is applied to account for the robot's footprint and add a cost for navigating close to obstacles.

In the case of the Hamburg Bit-Bots, both costmaps are configured identically except for their size. However, this is not required by the framework.

4.2. Global Planner

The global planner finds a path through a given map from the starting pose of the robot to a given goal. The map, which is represented as a grid, is transformed into a visibility

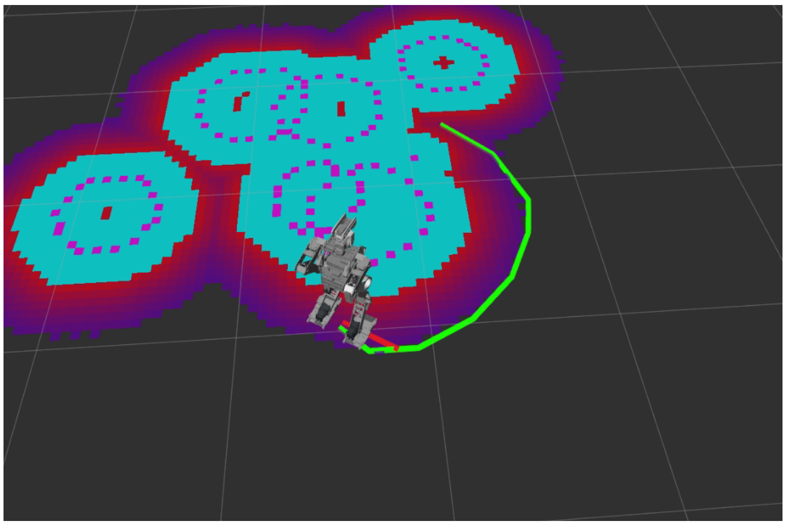


Figure 4.1.: Visualization of navigation pipeline usin RViz [51]. The global plan is pictured as the green line. The red line shows the local plan. The local and global costmap are identical and therefore overlapping. The obstacles in the costmap are pictured in magenta. The inflated costmap is pictured in cyan. Around the inflated obstacles an exponentially decaying potential field is displayed in red to purple.

graph. In the configuration used by the Hamburg Bit-Bots Lazy Theta* [57] is applied on the visibility graph of the global costmap to produce a shorter and smoother path than classical A* would. The generated path is passed to the local planner and published for visualization. The implementation is available as part of the `navigation2` framework.

4.3. Local Planner

The local planner generates a set of commands for the motion system of the robot to execute (e.g., goal velocity or footsteps). The Hamburg Bit-Bots and, therefore, this baseline employ a local planner approach similar to a carrot planner [50]. This planner generates a velocity command based on the position of a point on the path several steps ahead of the robot's current pose. If the distance to the final goal position is above a configurable threshold, the planner attempts to rotate such that the robot is moving forwards along the path. This is employed as the robot can move forwards more quickly than laterally. However, lateral movement is still employed to stay close to the global path. When the threshold is crossed, the planner generates velocity goals such that the robot turns to the goal's orientation and moves in a holonomic fashion (i.e., using forward, lateral, and angular movement) towards the goal.

Since rapid changes in velocity can cause instability, the planner applies a smoothing function on the walking command. The generated velocity is proportional to the distance of the

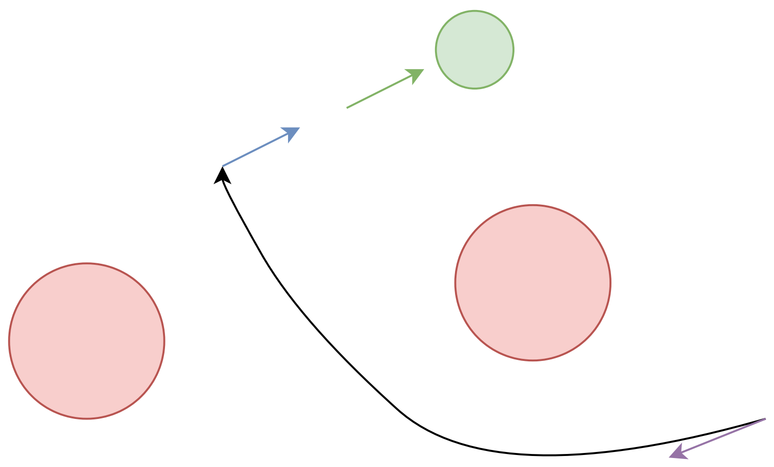


Figure 4.2.: Navigation strategy to reduce collisions with the soccer ball in the RoboCup context employed by the Hamburg Bit-Bots. The ball is represented as the green circle. Red circles show the obstacles, such as other robots. The starting pose of the robot is the purple arrow. The blue arrow shows the preliminary navigation goal, while the green arrow shows the final navigation goal. The path the robot calculates is visualized as the curved black arrow.

selected point in the global plan. This allows for slowing down when the robot is close to the goal pose to reach it accurately.

A downside of the planner is that it does not execute the global plan exactly as it does not take into consideration all points of the plan but rather a point several steps ahead. This can lead to collisions when the robot is close to obstacles.

For this thesis, the parameters of the local planner were tuned to find an optimum of stability and reaching the goal position efficiently. As they highly depend on the walking parameters, they had to be adapted from the version used in the Humanoid League Virtual Season (see Section 1.1.1) as better walking parameters were produced by Bestmann et al. [2].

4.4. Navigation Strategy

Approaching the goal position accurately is important in the RoboCup context. The robot usually navigates to a pose close to the ball to kick it towards the opponent's goal. Touching and moving the ball during navigation is not desirable as it changes the goal pose and requires navigating to a different pose. The strategy to reduce collision with the ball implemented in the Hamburg Bit-Bots navigation stack is firstly navigating to a pose behind the final goal pose with the same orientation. After this pose is reached, the final goal pose is approached. This reduces the chance of accidentally colliding with the ball as the final approach is only a forward movement. This strategy is visualized in Figure 4.2.

5. Approach

This chapter describes the developed approach. This thesis's goal was to train and evaluate an RL policy that can solve the footstep planning problem (i.e., which footsteps a robot should take to reach a navigation goal). The target platform of the approach is the Wolfgang-OP [4] (see Section 3.2). The novelty of this approach is in the combination of a walking engine with model-free RL to solve the navigation problem on a humanoid robot using footstep planning.

`stable-baselines3` [58] was used for the implementation of the PPO [38] algorithm. Furthermore, `rl-baselines3-zoo` [59], a collection of training and evaluation scripts, was used to train the models.

Section 5.1 presents the simulated environments in which the agent acts. The observation produced by the environment is presented in Section 5.2. Section 5.3 describes the policy which decides the action the agent takes based on this observation. The transformation from this action to a footstep command for the walking engine is presented in Section 5.4. Section 5.5 details the developed reward function the policy is optimized for. The performed hyperparameter optimization is described in Section 5.6.

5.1. Environment

Two environments were implemented for this thesis. Firstly, a 2D geometric simulation of the footstep capabilities of the walking engine. Secondly, a 3D environment in the Webots simulator [5] (see Section 3.5) using the simulation model of the Wolfgang-OP [4] (see Section 3.2) with the *Quintic Walk* engine [2] (see Section 3.3). Both environments share as much behavior as possible to allow for future transfer learning between them as well as applying policies trained in one environment to the other.

At the beginning of an episode the robot is initialized to the zero position. The goal position is sampled from a zero mean two dimensional normal distribution with standard deviations $\sigma_x = \sigma_y = 1\,\text{m}$. The goal orientation is also sampled from a zero mean normal distribution with $\sigma_\theta = 1.5\,\text{rad}$. These parameters were selected to ensure enough samples are interesting navigation goals for a holonomic robot that has different maximum velocities in different directions. For example, if a navigation goal has the same orientation as the starting orientation but is to the side of the robot, depending on its distance, it may either be favorable for the robot to rotate, walk forward, and rotate again, or to walk laterally, or to do a combination of both. Early experiments also showed that when few goals close to the robot were in the training samples, the agent was not able to navigate some manually selected goals.

Obstacles the agent must avoid can optionally be placed in the environment. Inspired by the RoboCup Humanoid League context, seven circular/cylindrical obstacles are generated representing other robots in the own or opponent team and the soccer ball. Their radius is uniformly sampled between 0.1 m and 0.3 m as these are typical robot radii in the Humanoid League. Three obstacles each are placed around the robot and the goal position. The deviation of the robot or goal position is sampled from a two-dimensional normal distribution with $\sigma_{obs} = 0.5$ m. The obstacle is resampled if the robot collides with it when initialized or cannot reach the goal because it is inside this obstacle. Lastly, an additional obstacle with the radius of the soccer ball used in the RoboCup Humanoid League is placed in front of the goal pose as the robot typically has to navigate to the ball to perform a kick.

Each step of the agent interacting with the environment is a fully walking cycle, a double step. This ensures that the policy always decides the agent's action in the same state of the walking cycle.

The robot reaches its navigation goal when it is closer than 0.05 m in Euclidean distance and 10° in angle distance away from the goal position. The episode is terminated if the agent reaches the goal, the number of steps reaches 100 (the maximum episode length), the robot collides with an obstacle (only in the 2D environment), or the robot falls (only in the 3D environment).

The environments are implemented using the OpenAI gym interface [37]. Common behaviors are defined in an abstract base class to limit code duplication between the environments.

5.1.1. 2D Environment

Figure 5.1 shows a rendering of the 2D environment. The rendering was created to qualitatively evaluate the performance of the policy during the development process. During training, it is disabled to increase performance. As the 2D environment only calculates the next position of the robot geometrically, a policy can be trained reasonably quickly. This is useful as many iterations were required to develop all aspects of the RL setup. It also allows for optimization of the hyperparameters of the training algorithm (see Section 5.6) as many policies can be trained and evaluated in parallel.

The distance to the closest obstacle is calculated to check whether the robot collides with an obstacle or not. If it is below a defined threshold, the robot is considered in collision, and the episode is terminated.

The environment sacrifices realism for speed of calculation. A scenario that models the real robot's behavior more closely is the 3D environment described in the following Section.

5.1.2. 3D Environment

The 3D environment employs Webots [5] for simulating the physics of the Wolfgang-OP robot. A rendering of the environment is depicted in Figure 5.2. The *Quintic Walk* engine (see Section 3.3) generates joint positions based on the footstep command from the policy. These are passed to the simulator, which translates them into torques by internal proportional controllers. Contact properties and other simulation parameters, such as the length of the simulation step, are taken from the Humanoid League Virtual Season (HLVS) environment (see Section 1.1.1) to ensure the approach is transferable to the competition.

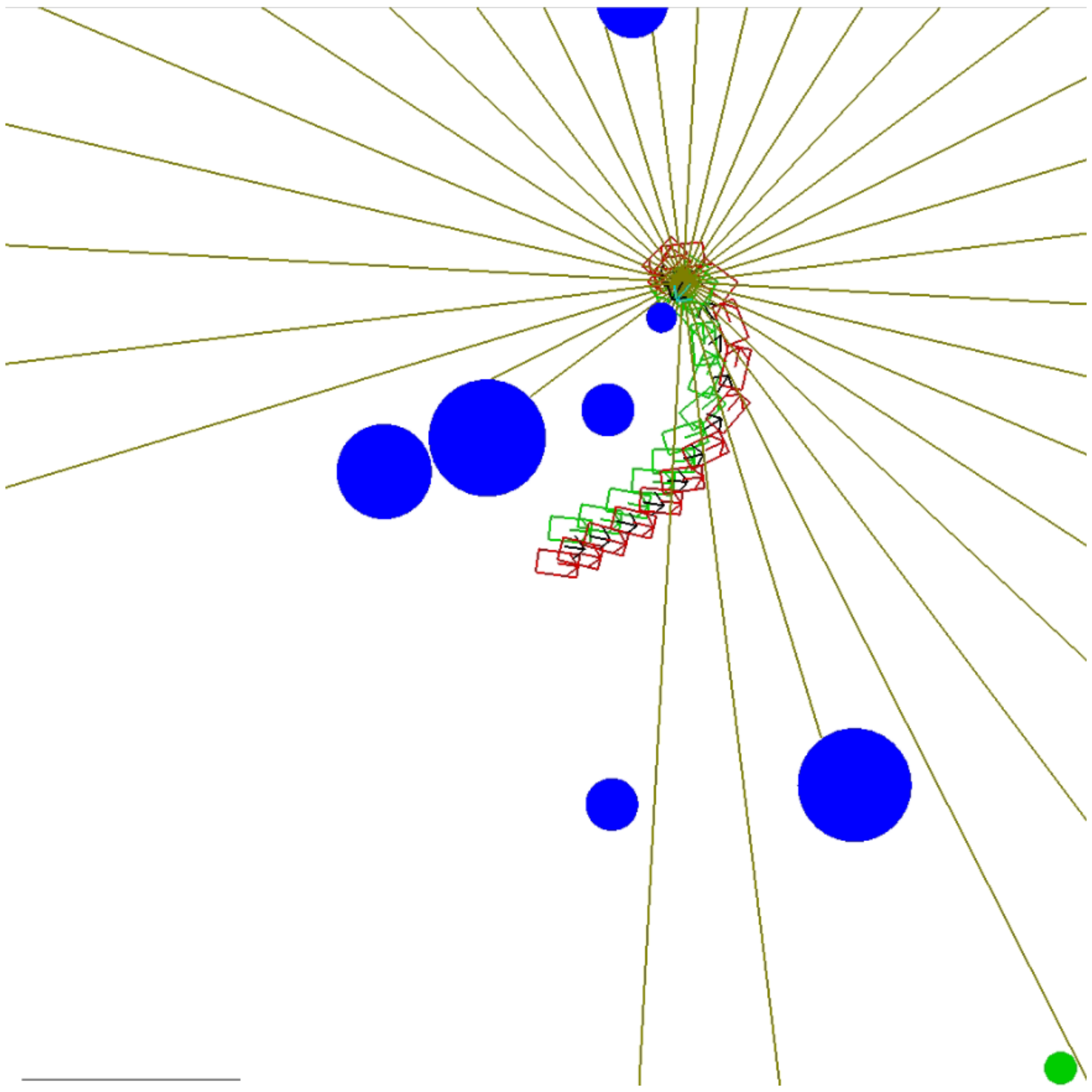


Figure 5.1.: Rendering of the 2D environment. The footstep path taken by the agent is colored green and red for the left and right foot, respectively. The black arrows indicate the mean pose of the footsteps, which is the navigational frame. The olive-colored lines are a rendering of the laserscan-like measurements of the obstacles colored in blue. The green circle in the bottom right corner indicates that the agent has reached to goal with sufficient precision. Otherwise, it would be red. The black line on the bottom right is used for reference as it has a length of $1\,\mathrm{m}$.

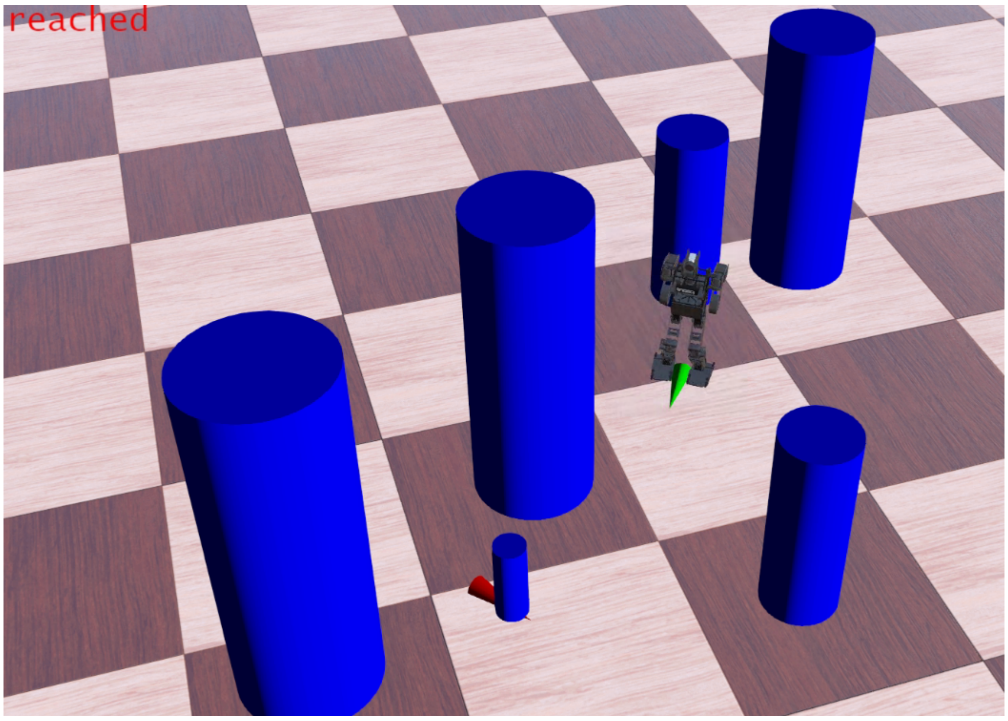


Figure 5.2.: Rendering of the 3D environment in the Webots [5] simulator. The robot's navigation pose is displayed using the green cone. The red cone shows the current goal pose. Blue cylinder obstacles are placed as described in 5.1. The color of the text "reached" in the top left corner indicates whether the robot has reached the goal with sufficient precision.

As opposed to the 2D environment, collisions between the robot and the obstacles are precisely checked during simulation. However, they do not lead to the termination of the episode as they already penalize the agent by destabilizing it. The robot's pose is defined as its base link in the center of the bottom of the torso. It is a six-dimensional vector in Cartesian space. However, the navigation approach operates in the 2D ground plane. The 6D vector is transformed into the navigation representation (x, y, and θ) by projecting it onto this plane.

Fused angles [60] are used to evaluate whether the robot fell during walking. If the robot crosses a threshold in the fused roll or fused pitch, it is considered as falling, and the episode is terminated. The threshold was chosen such that it does not occur during walking and can not be recovered from by the walking engine.

5.2. Observation

The observation is the input to the policy. It is either a 17- or 53-dimensional vector. Since a real robot has a noisy estimate of its localization, this noise is applied to the robot's

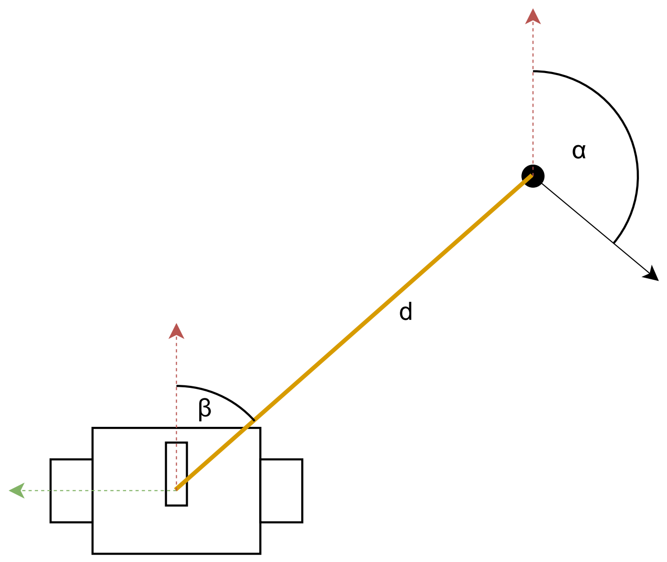


Figure 5.3.: Drawing of some of the parameters of the environment observed by the robot. d is the distance between the robot's center to the goal position. The distance the robot would need to turn to align with the goal's orientation is expressed as α. The angle β describes how much the robot would need to turn to face the goal position.

position and angle measurement before the observation is calculated. An evaluation of the current localization approach used on the robot platform was performed and is described in Section 6.1 to obtain a reasonable noise model. The observation is composed of the following measurements depicted in Figure 5.3:

Distance to Goal The relative distance d to the goal is normalized using a hyperbolic tangent function.

Relative Goal Orientation The relative goal direction α is the angle the robot would need to turn to have the same orientation as the goal orientation. It is represented as the sin and cos function of the angle in the observation. Neural networks have difficulty with the discontinuity and ambiguity of radians or degree values [61] which is not present in this representation.

Angle to Goal The angle to goal β is the angle the robot would need to turn to face the goal position. It is represented as the sin and cos function of this angle like the relative goal orientation.

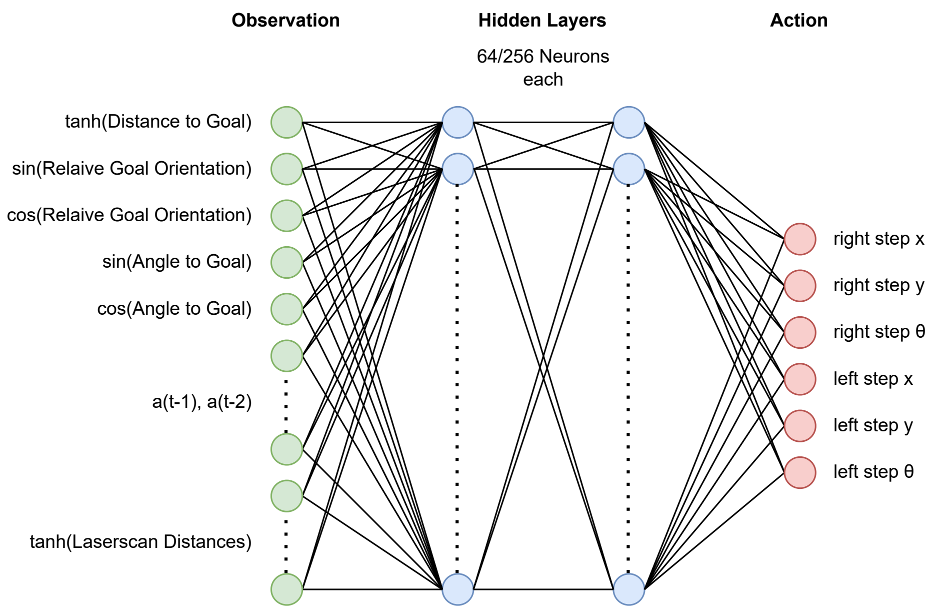


Figure 5.4.: Structure of the neural network used as a policy in the approach. The inputs are the observation described in Section 5.2. Two fully connected hidden layers with 64 or 256 neurons each calculate the output with either *ReLU* or *tanh* activation functions. The output is the action described in Section 5.4.

Step History The two previously selected actions of the policy are part of the observation to allow the policy to know the velocity of the robot.

Obstacle Distance Measurement The observation includes a measurement similar to a laserscan when obstacles are present in the environment. 36 rays are cast from the center of the robot in increments of $10°$. They measure the distance to the first intersection with an obstacle. The measurements are normalized using a hyperbolic tangent function. Figure 5.1 shows these measurements.

5.3. Policy and Value Function

The policy and value function are multilayer perceptrons (MLP). They are composed of an input layer with the dimension of the observation and an output layer with the dimension depending on the size of the action for the policy, and a single output for the value function. These layers are connected by two fully connected hidden layers. The hidden layers have 64 or 256 neurons in the policies trained in this thesis. A Rectified Linear Unit or hyperbolic tangent is used as the activation function of the neurons. While PPO [38] allows shared layers of the value and policy MLP, this is not used.

The policy's output is a parameter of a distribution. Several distributions can be used in the employed framework stable-baselines3 [58]. For a normal distribution, as presented in

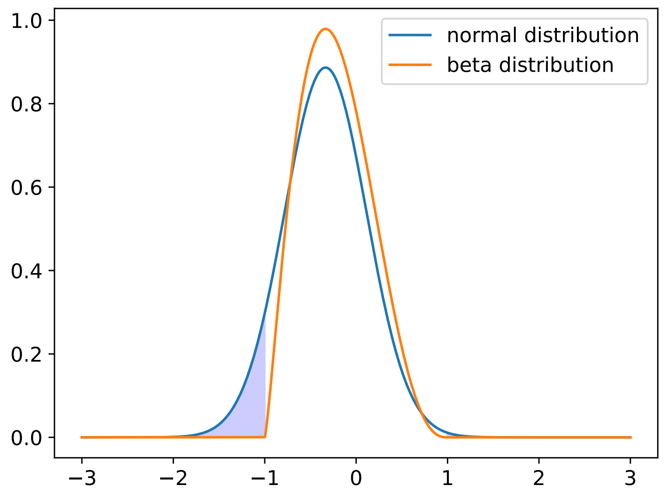


Figure 5.5.: Probability density function (PDF) of the normal and the beta distribution when used for an action space defined between -1 and 1. The beta distribution is scaled to fit this range. When using a normal distribution, clipping must occur in the blue-colored area to ensure the bounds of the action space are respected.

the original paper, the action is the mean of the distribution. The standard deviation is a hyperparameter of the training algorithm, which is decreased as training progresses. It faces the issue that the action space can be bounded, but sampling may result in values outside this range. A squashing function such as a hyperbolic tangent can be applied to the distribution to ensure boundedness.

The beta distribution can also be employed as it is already bounded. Here, the output of the network is an α and β parameter, which shapes the distribution [62]. Figure 5.5 shows the two distributions and the clipping that occurs in the normal distribution.

When the policy is evaluated, the mean produced by the policy or the mode of the beta distribution is applied as the action of the agent.

5.4. Action

The action the agent takes in each iteration of interaction with the environment is a double step. The output of the policy is six-dimensional, with three parameters for each footstep. Footstep poses are defined relative to the resting position of the foot as shown in Figure 5.6. The policy's output is scaled with the limits determined by the kinematic capabilities of the robot for the x- and y-direction. For the Wolfgang-OP, this results in a maximum step in of $0.156m$ and $0.103m$ in x- and y-direction, respectively.

The robot is kinematically capable of turning each footstep up to $90°$. However, footsteps with this rotation are almost never stable. An experiment was performed to determine a

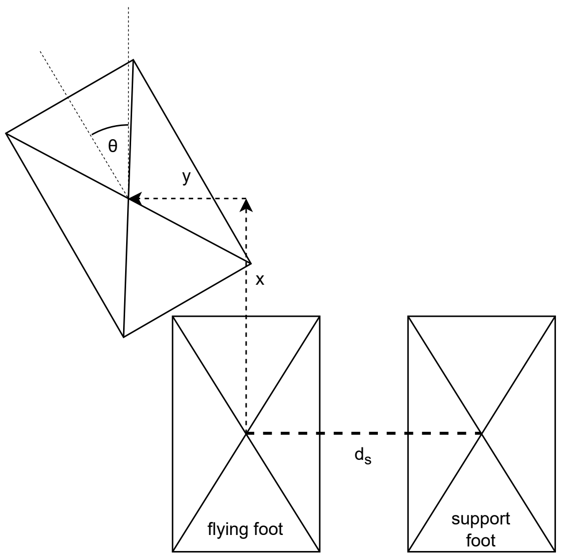


Figure 5.6.: Visualization of the coordinates describing a footstep position. The coordinates x, y, and θ are relative to the resting position of the flying foot. The resting position of the flying foot is defined as having the same x coordinate and angle θ as the support foot. The distance in y-direction d_s is the distance between the feet in the robot's standing posture.

reasonable range. The robot was placed in simulation, and only the rotational velocity was increased until it fell. The maximum determined rotation of the footstep was calculated to be $43.77° = 0.764\,\text{rad}$.

5.5. Reward Function

A reward function was carefully designed to guide the exploration of the policy. Most terms in the reward function are continuous, which allows a gradient to be extracted from them. This is favorable as PPO is a gradient-based method. The reward function is expressed as:

$$r(t) = s_d \cdot r_d(t) + s_a \cdot r_a(t) + s_o \cdot r_o(t) + s_{obs} \cdot r_{obs}(t) + s_c \cdot r_c(t)$$

The individual components of the reward function are explained in the following paragraphs. The scalars (e.g., s_d) are used to scale the importance of the component rewards. They were experimentally selected. For the scenario with obstacles $s_d = 0.25$, $s_a = 0.25$, $s_o = 0.15$, $s_{obs} = 0.2$, and $s_c = 0.15$ and without $s_d = 0.3$, $s_a = 0.3$, $s_o = 0.2$, $s_{obs} = 0.0$, and $s_c = 0.2$ was chosen. The sum of scalars is 1 in both scenarios. All rewards terms are furthermore in $[0, 1]$ leading to a maximum achievable reward of 1 per step.

If the episode is terminated because the agent failed to reach the goal, no additional reward is given. When the robot succeeds, the maximum achievable reward for the remaining episode

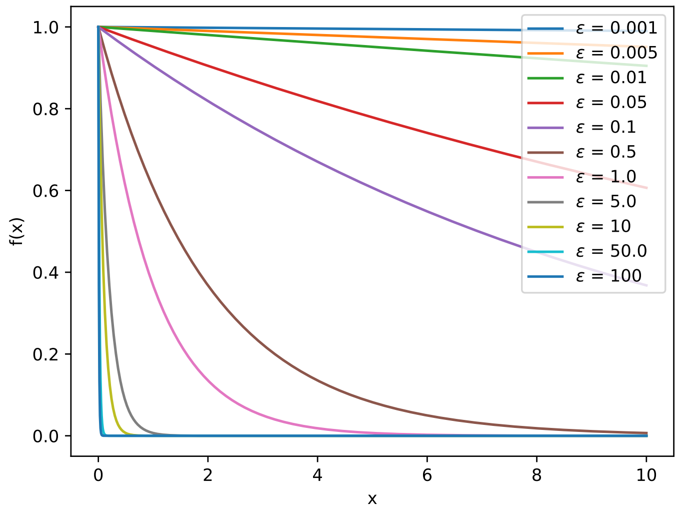


Figure 5.7.: $f(x) = exp(-\varepsilon x)$ plotted for different values of ε. The parameter ε must be chosen carefully to fit the expected range of occurring values for x. If ε is chosen too high, the no gradient will be extractable from the samples collected in the environment. If chosen too low, the agent will always receive a large reward in the expected range of values. Therefore, the reward would be less expressive of the agent's performance.

is given. As the maximum achievable reward per step is 1, it is the number of steps remaining in the episode.

All component of the reward function use the function $exp(-\varepsilon x)$ to normalize to the interval $(0, 1]$. The factor ε allows to adjust the function to different input spaces such as Euclidean or angular distance. Figure 5.7 shows the function for different choices of ε. It is inspired by the Gaussian radial basis function but features a steeper gradient near $x = 0$ to encourage the agent to precisely reach its navigation goal.

Distance Reward The distance reward guides the policy towards the goal by increasing when the robot is closer to the goal. It is expressed as:

$$r_d(t) = exp(-0.5 \cdot d)$$

The variable d is the Euclidean distance between the robot and the goal. The scalar ε was chosen such that a clear gradient is present for all expected values, which are determined by the goal pose selection described in Section 5.1. Figure 5.8 shows the function.

Angle Reward The angle reward ensures that the agent can gradually learn that it needs to rotate to match the goal's orientation. However, the agent's orientation only becomes

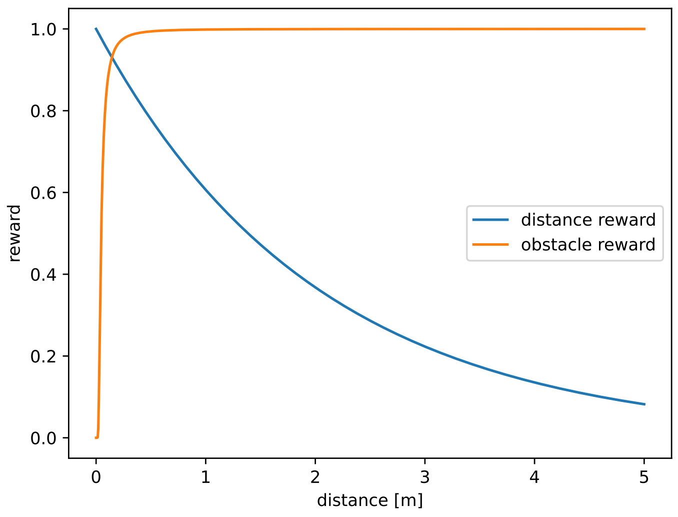


Figure 5.8.: Distance and obstacle reward as a function of the distance to the goal position and the minimum distance to an obstacle, respectively.

important as it comes close to the goal position. Therefore, it is scaled by a function of distance. It is expressed as:

$$r_a(t) = exp(-5 \cdot d) \cdot exp(-1 \cdot |\alpha|)$$

The variable d is the Euclidean distance between the robot and the goal. α is the angular distance between the robot's and the goal's orientation. The function is plotted in Figure 5.9.

Orientation Reward The orientation reward guides the agent to look toward the goal. This is generally favorable as the camera, the robot's primary sensor for perceiving its environment, is facing forwards, allowing the robot to observe what is ahead in its path (e.g., obstacles). The following function describes the orientation reward:

$$r_o(t) = \begin{cases} exp\,(-1 \cdot |\beta|) & d \geq 0.1 \\ 1 & d < 0.1\,\mathrm{m} \end{cases}$$

With d being the euclidean distance to the goal position and β the angle the robot must turn to face the goal position. If the robot is closer to the goal than the threshold 0.1 m, $+1$ reward is given to the agent. This is required as when the robot is close to the goal position, it may oscillate around it, which rapidly changes β and possibly drastically decreases the reward. Experiments during development have shown that omitting this clause creates a local minimum in the reward function, which decreases training performance.

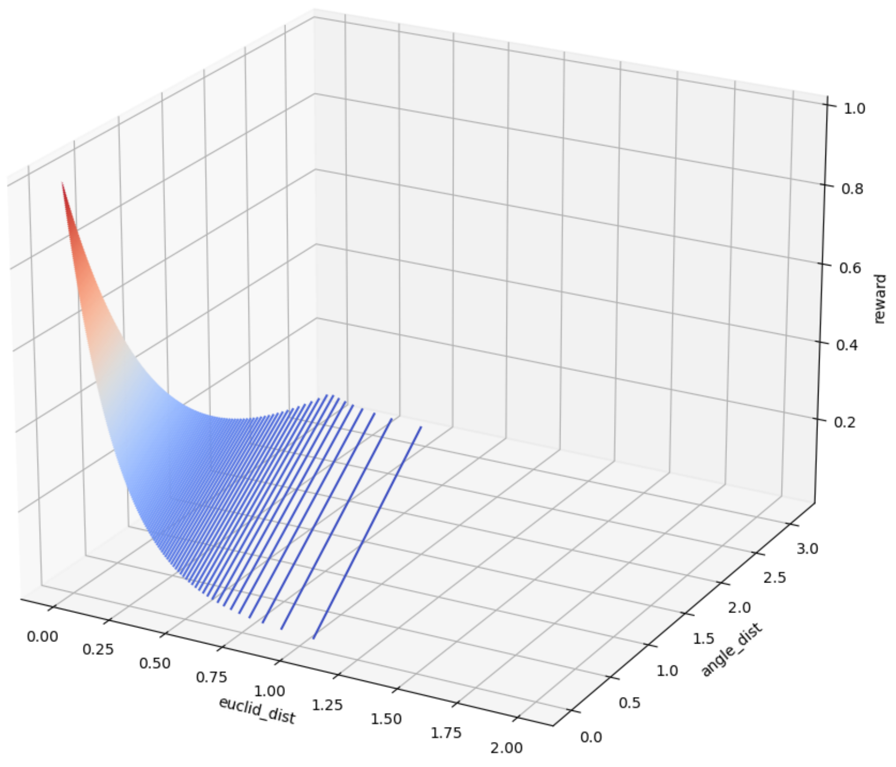


Figure 5.9.: Angle reward function depending on the euclidean distance between robot and goal and angle distance between robot orientation and goal orientation. When the euclidean distance is greater, the agent is negligibly penalized for not being oriented like the goal pose. It can only achieve a significantly larger reward by changing its orientation once it comes closer to the goal.

Obstacle Reward The obstacle reward rewards the agent for keeping distance from obstacles to prevent collisions. Figure 5.8 shows the function:

$$r_{obs}(t) = exp\left(-0.0015 \cdot \left(\frac{1}{d_{obs}}\right)^2\right)$$

where d_{obs} is the distance between the robot and the closest obstacle. ε was chosen relatively small to allow the agent not to be too strongly penalized for being relatively close to obstacles, as this may be required for efficient navigation. The obstacle representing the ball is not taken into account when calculating d_{obs} as it would create a local minimum before reaching the goal pose, which policy gradient methods generally do not handle well.

Continuity Reward Big changes in footstep length and, therefore, the robot's velocity is usually unstable. The following reward component is introduced to encourage the agent to accelerate continuously.

$$r_c(t) = mean\left(exp\left(-3 \cdot \|a(t) - a(t-1)\|\right)\right)$$

the difference between the current action $a(t)$ and the previous action $a(t-1)$ is normalized. The mean value is taken to account for each component when it is converted to a scalar reward value.

5.6. Hyper Parameter Optimization

Hyperparameters are the parameters used by the training algorithm to change the policy. Using multiple training episodes while varying these can give insight into which set of parameters best suits the given problem. Manually performing this task is tedious and error-prone. Therefore, the automatic hyperparameter optimization script included in rl-baselines3-zoo [59] was employed. It uses the Optuna library [48] to perform a study of the hyperparameters automatically. A sampler provides a set of hyperparameters for each training of the policy. Several samplers are available in the library. While random search or grid search can produce good results given enough time, a Tree-Structured-Parzen-Estimator (TPE) [63] was employed to reduce computational requirements. Only the parameters of the 2D environment could be optimized, as many iterations of training are required. As the observation and action space is equivalent between the 2D and 3D environment, the same parameters found to perform well for the 2D environment were used in the 3D environment.

6. Evaluation

This Chapter presents the evaluations performed in this thesis. Firstly, Section 6.1 describes an evaluation of the localization algorithm described in Section 3.4, which was used to obtain reasonable estimates about the noise in pose estimate the robot experiences. Section 6.2 presents the experiments performed to compare the trained policy with the baseline approach and their results. Lastly, Section 6.3 describes experiments performed to evaluate transferring the policy trained in the 2D environment to the 3D environment.

6.1. Localization Evaluation

This section describes the evaluation of the localization pipeline of the Hamburg Bit-Bots [49, 50] (see Section 3.4). The experiment is based on the pose tracking experiment presented in Section 5.2 by Harfill [49]. In the original work, the robot did not use the walking engine or path planning to navigate from the start to the goal position. It was simply teleported in the simulator. In this work, the baseline presented in Section 4 is used to walk from the start to the goal pose. Additionally, only field line measurements are used to update the localization's particles as they have shown to be the most robust measurements in a real-world scenario. The head behavior of the Hamburg Bit-Bots [64] is used during the experiment as in the original. It follows a scanning pattern to increase the number of observed lines. The experiment is, as in the original, repeated ten times.

Figure 6.1 shows the data from the reproduced pose tracking experiment of the robot entering the field at the sideline and walking towards the striker position (i.e., at the center circle, facing the opponent's goal). This experiment is referenced as experiment 0.

The x-deviation, i.e., the deviation in the forward direction, is well fitted with a bimodal Gaussian distribution. This fits the observation that the localization estimate lags behind the ground truth pose of the robot visualized in Figure 6.1. When the robot is moving forwards, the estimates are behind the robot, therefore in negative x-direction. However, when the robot is almost stationary (at the beginning and end of navigation), the change in position that occurs during the processing delay is negligible.

The y-deviation, i.e., the deviation in the lateral direction, is less clearly fitted by a bimodal distribution. However, this can be explained by the relatively small lateral velocity the path planner commands to the walking engine. The angle deviation is clearly well fitted by a single Gaussian distribution.

The delay of the pose estimate is well explained by the processing time of the vision pipeline detecting the lines in the image. Further delay is added by the inverse projection mapping to Cartesian space. The measurement of the kinematic chain at the time of recording the image on which the detections were made is used for this projection. However, the particle filter of

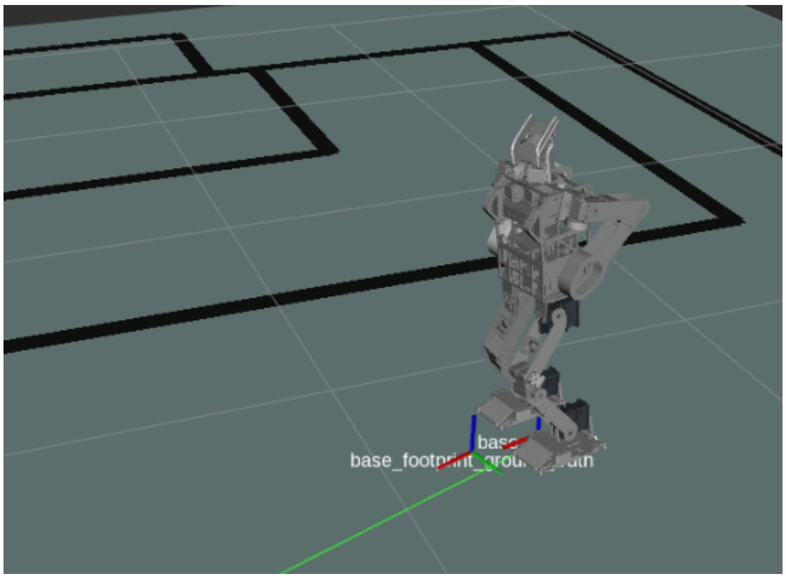
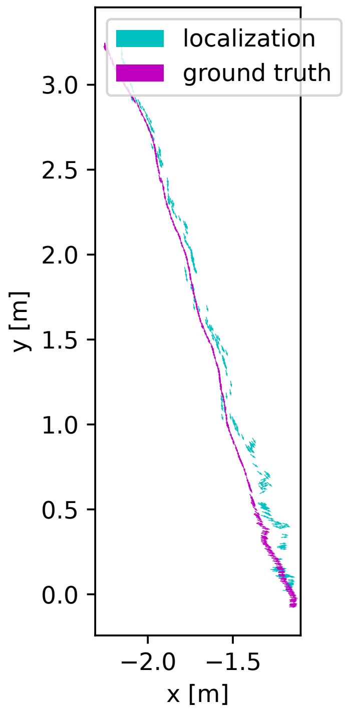


Figure 6.1.: Visualization using RViz [51] of the robots localization estimate and the ground truth extracted from the simulator. The robot is walking forwards. The ground truth measurement is displayed as a coordinate system in front of the robot. The robot is rendered relative to its localization estimate visible as the coordinate system between its feet.

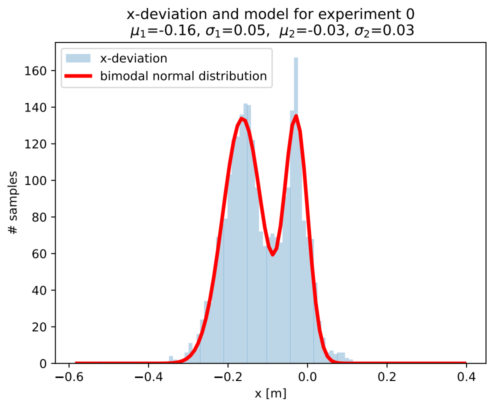
the localization updates its particles based on this delayed observation. A history of particles, which is extrapolated with the robot's odometry information to estimate its current position, could correct this.

For this thesis, it is assumed that this technique is applied and efficient or that the processing delay is reduced such that the distribution becomes unimodal and has zero mean. The standard deviation σ_x of the noise applied on the ground truth information from the simulator in x-direction is assumed to be the standard deviation of the robot in motion in Figure Figure 6.2b, $0.05\,\mathrm{m}$. In y-direction it is assumed to be $\sigma_y = 0.03\,\mathrm{m}$ as in Figure Figure 6.2c. The noise applied on the angular ground truth estimation is also taken from the experiment as $\sigma_\theta = 0.07\,\mathrm{rad}$.

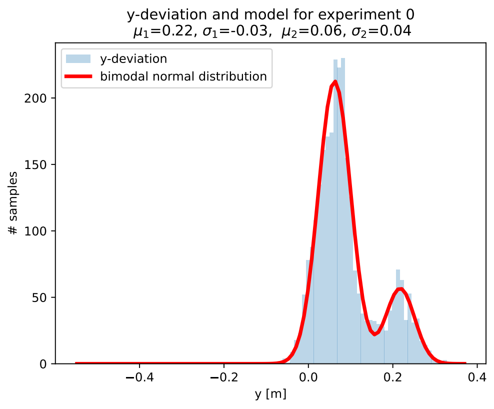
The other pose tracking experiments presented in the thesis by Harfill [49] were reproduced as well. The plots can be found in Appendix A. The bimodal nature of the distribution is not as evident in some of the other experiments as in experiment 0 since the robot performs walking in multiple directions along the same axis or simply rotates in place.



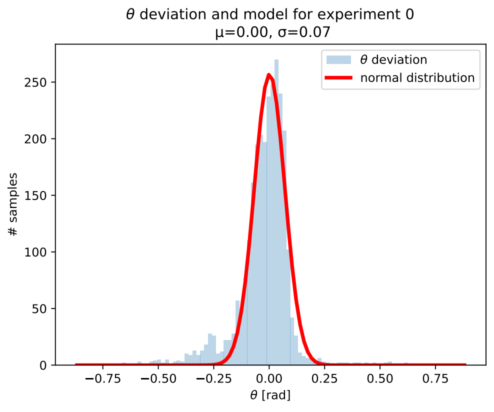
(a) Ground truth path taken from the simulator (magenta) and path measured by the localization approach (cyan) of an example run.



(b) Deviation between ground truth and localization pose in x-direction in the ground truth reference frame.



(c) Deviation between ground truth and localization pose in y-direction in the ground truth reference frame.



(d) Angle deviation between the ground truth and the localization pose.

Figure 6.2.: Results in deviation between the ground truth and the localization estimate.

6.2. Approach Evaluation

Two evaluation scenarios are proposed for comparing the baseline and the presented approaches. Firstly, the performance in obstacle-free navigation is compared. The goal of this scenario is to evaluate how well each approach can control the walking engine to reach a goal pose efficiently. This experiment is described in Section 6.2.1.

Secondly, performance in a navigation scenario with obstacles placed in the environment as described in Section 5.1 is compared. These experiments show how well the approaches handle navigating through a cluttered environment. The results are presented in Section 6.2.2.

Two experiments were performed for each of these scenarios. Firstly, ground truth information from the simulator is passed to the approaches. This demonstrates the performance in an ideal scenario. Secondly, Gaussian noise derived from the evaluation performed in Section 6.1 is added to the robot's position estimate. This demonstrates how well the approaches are able to act in an uncertain environment.

In the 2D environment, four policies were trained, with and without noise added to the observation, in the environments with and without obstacles. Training in this environment took 30 minutes to two hours. When obstacles are in the environment, the laserscan-like measurements must be calculated, which takes a substantial part of the processing time. Furthermore, hyperparameters affect the time required for training (e.g., a smaller batch size increases the frequency of performing the optimization step, and the number of epochs causes the optimization step to process longer).

In the 3D environment, only two policies were trained for the final evaluation, in the environments with and without obstacles. No noise was applied during training to reduce the error in the stochastic gradient. This choice was made as the training of these policies took 36 hours. All policies were trained for 1.000.000 steps on an AMD Ryzen 9 5900X with an NVIDIA GeForce RTX 2080 Ti.

A set of 500 goal poses and respective obstacle locations and sizes was generated by the method described in Section 5.1. 500 was chosen as the number of poses due to the processing time required for performing the evaluation. While the 2D environment can be evaluated in several minutes, the 3D environment requires one or twelve hours without and with obstacles, respectively. The baseline processes for approximately thirty hours due to the overhead introduced by ROS2 in Python. However, these processes can be parallelized as the evaluation episodes are independent of each other.

Navigation is considered successful if the robot reaches the goal within 0.05 m in Euclidean distance and $10°$ in rotation. Furthermore, the robot must reach the goal in a maximum time of 100 s. Otherwise, the episode is considered a timeout.

6.2.1. Obstacle Free Navigation

The first scenario evaluates obstacle-free navigation. The challenge for the 2D approach is to find a geometric solution to the footstep planning problem without considering possible inaccuracies in the execution of the footsteps. It was used as a proof of concept to show that an RL policy could learn this task. In the experiment where noise is applied to the observation, it also serves to show that a policy can deal with uncertainty.

Approach	Success	Timeout	Fall	Success Time [s]	Success Steps
2D	1.0	0.0	-	2.27 $\pm$0.82	4.45 $\pm$1.62
3D	0.98	0.0	0.02	6.07 $\pm$2.13	11.92 $\pm$4.19
Baseline	0.96	0.032	0.008	11.13 $\pm$4.95	21.86 $\pm$9.72

(a) Without noise

Approach	Successes	Timeout	Fall	Success time [s]	Success steps
2D	1.0	0.0	-	2.92 $\pm$1.01	5.72 $\pm$1.98
3D	0.978	0.0	0.022	6.35 $\pm$2.23	12.46 $\pm$4.38
Baseline	0.91	0.078	0.012	11.43 $\pm$5.11	22.44 $\pm$10.03

(b) With noise

Table 6.1.: Results from the obstacle-free evaluation scenario. The columns "Success", "Timeout", and "Fall" show the rate at which the respective result occurred. The column "Success time" shows the mean time and its standard deviation for the successfully completed runs. The column "Success steps" presents the mean number and standard deviation of double steps for the successfully completed runs.

The challenge for the 3D approach and the baseline is to efficiently control the walking engine to reach the navigation goal quickly without causing the robot to fall. Path planning is not required for this task as the robot can approach the goal in a straight line. However, multiple strategies can be employed to reach a navigational goal as the walking engine is omnidirectional, and the maximum velocity in each direction is different.

Table 6.1 shows the quantitative results of the experiment with and without noise. The trained policy performed significantly better than the baseline, with more successful runs. It took on average 54.5% or 55.5% of the time or number of steps as the baseline policy in their respective successful runs for the experiment without or with noise. The timeouts in the baseline approach were caused by the robot oscillating around the goal pose without reaching it precisely enough to be considered successful.

While falling in the simulator has no consequence, it can present a significant problem in the real world. The policy trained in the 3D environment produced 2.5 or 1.83 times as many falls as the baseline for the experiment without and with noise, respectively. This not only leads to increased time to navigate to a goal as the robot has to stand up, but can also lead to damage to the robot and hardware failure. However, the Wolfang-OP is built to be robust to falls, as these frequently occur in the RoboCup competition.

The performance of the 3D policy only slightly decreased between the scenario without and with noise added to its observation. In comparison, the baseline experienced a timeout 2.44 times more often. This result shows that the policy is tolerant to a noisy observation which is required for deployment in the real world. As the number of falls only very slightly increased in the baseline and 3D policy, it is questionable if these values are statistically significant or random variations.

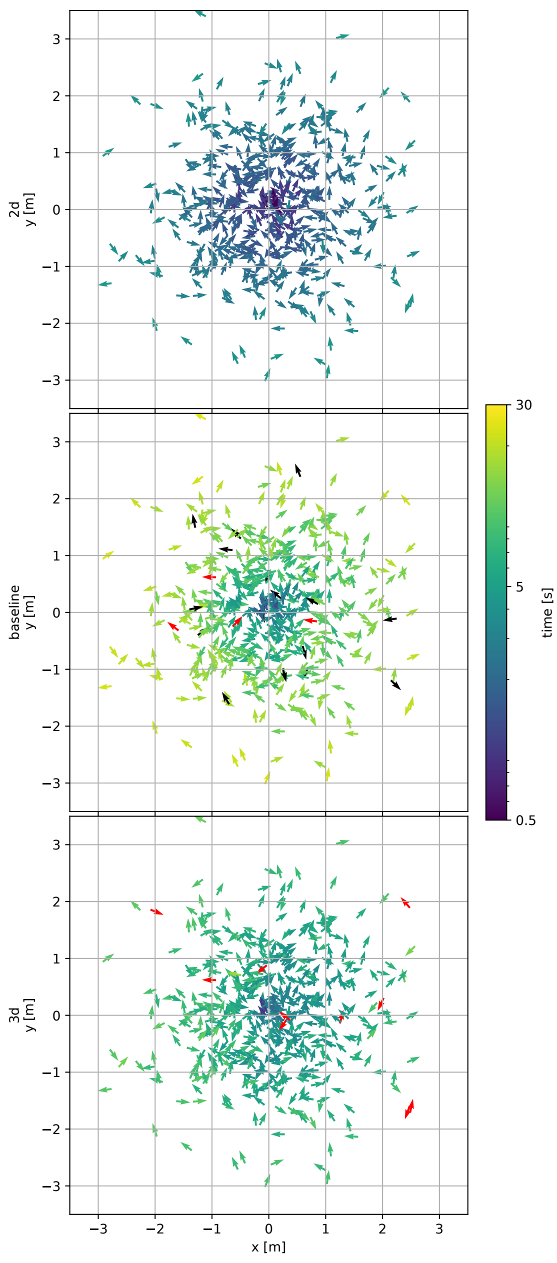
Figure 6.3 presents the results of the two experiments graphically, with each pose in the evaluation set colored according to the time it took the approaches to reach. No pattern was found on which navigation goals lead to falls or timeouts more frequently. As expected, the time to reach a goal correlates with the distance to it.

In the RoboCup context, this major increase in the robot's performance can be a decisive factor in overall performance. The robot would reach a pose to manipulate the soccer ball into the opponent's goal quicker and more often before its opponent. Furthermore, it could position itself for defense before the opponent is able to score a goal more frequently.

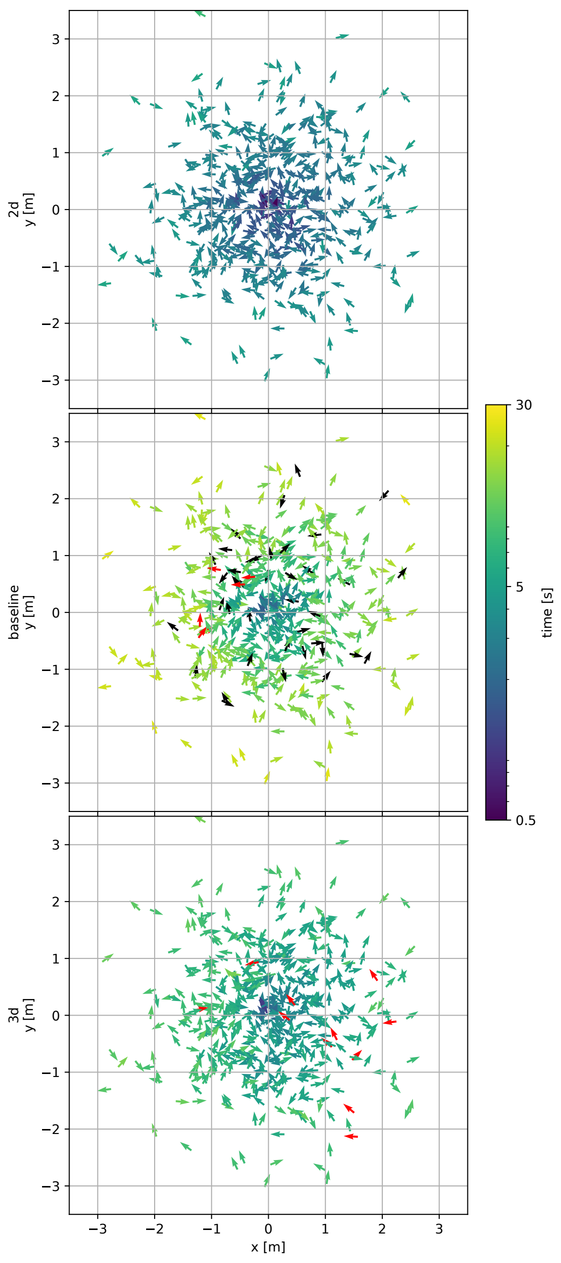
An exemplary path the different approaches took for the same navigation goal can be seen in Figure 6.4. Generally, it can be observed that the policies trained in the 2D and 3D environments take much larger steps than the baseline approach. The employed strategy to reach the goal pose differs between the 3D policy and the baseline approach. In the noise-free experiment, the baseline first tries to turn towards the goal pose. This does not occur in the experiment with noise. The disparity is caused by the distance of the navigation goal being close to the threshold employed to decide between the two strategies in the baseline approach (see Section 4.3). In the experiment with noise, the goal was initially measured as being slightly closer, and therefore the goal approach strategy was executed.

The policy trained in the 3D environment followed a strategy of first turning towards the goal pose and, after almost reaching it in position, rotating. This behavior is shaped by the reward function, which favors orienting towards the goal and only significantly penalizes the angular distance between the robot's orientation and the goal orientation when it comes close to the goal.

The policy trained in the 2D environment followed a strategy similar to the goal approach strategy of the baseline. However, a reason that this strategy is faster in the 2D environment than in the baseline is that there is no collision check between the robot's feet. Adding this collision check to increase the realism of the 2D environment is viable as it is computationally inexpensive.

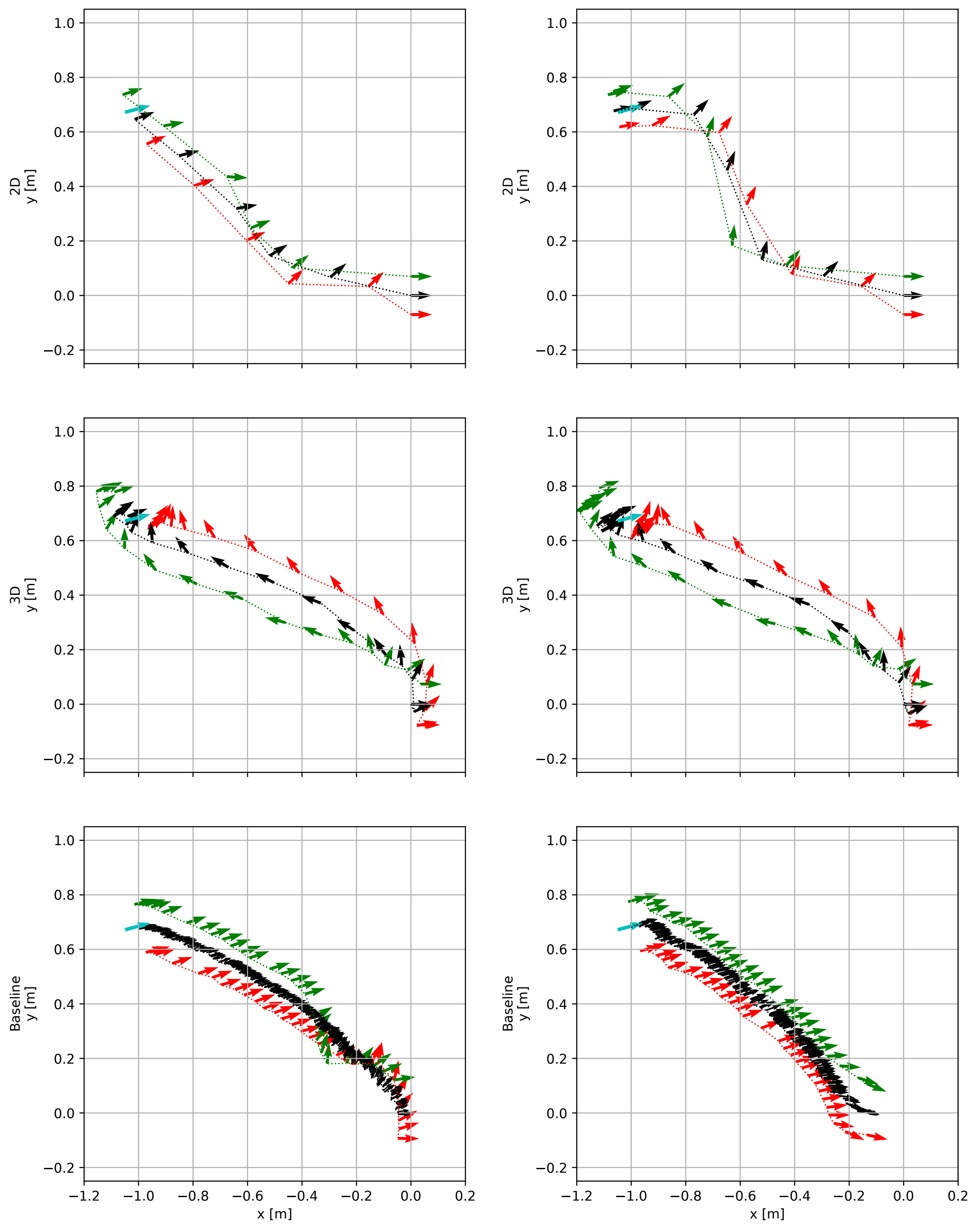


(a) No obstacles, without noise



(b) No obstacles, with noise

Figure 6.3.: Required time for reaching each pose in the evaluation set. Poses colored in red indicate that the robot fell. Poses colored in black show timeouts. The color scale is logarithmical.



(a) No obstacles without noise

(b) No obstacles with noise

Figure 6.4.: Example paths taken by the different approaches in the obstacle-free scenario with and without noise. Red and green arrows show the poses of the right and left foot, respectively. The black arrows are the navigational representation of the robot. In the baseline, they are sampled more frequently and therefore denser. The goal pose is colored cyan.

6.2.2. Navigation with Obstacle Avoidance

The scenario with obstacles is set up similar to the previous experiment. However, obstacles are placed in the environment as described in Section 5.1. The same set of poses and obstacles is used for all evaluations.

The goal of these experiments is to evaluate if the trained policies can outperform the baseline approach in an environment where it has to navigate around obstacles to reach a goal pose. In the RoboCup context, these obstacles are usually other robots. Avoiding collisions with them is crucial to prevent falls, fouls, and damage.

The quantitative results of the performed experiment are displayed in Table 6.2. They show that while the policies trained and evaluated in the 2D environment were able to reach the navigation goal in almost all of the cases with very few collisions or timeouts, the policy in the 3D environment did perform as well as could be expected from the previous scenario. However, as the observation and action in the 2D and 3D environments are similar, this hints at the approach being theoretically able to perform more successfully than the baseline. In practice, more training time or optimized hyperparameters may be required for the policy in the 3D environment with obstacles to be in a usable state. Additional possibilities for improving its performance are outlined in Chapter 9.

Unexpectedly, the policies trained in the 2D environment reached the goal more often with noisy observations than with ground truth information. This can be explained by the policy trained with the noisy observation having to be more conservative in its produced footstep poses to compensate for the uncertainty in observation. The increased time the successful runs took supports this hypothesis.

Also, unexpectedly, the baseline achieved more successful runs in the environment with than without noise. This is caused by the global planner. If the robot steps close to an obstacle, it may be inside an obstacle in the costmap. The global planner does not produce a path when the robot is inside an obstacle. If noise is applied, a global plan will be generated if the noise causes the measurement of the robot's position to be outside the obstacle in the costmap.

The baseline also performed significantly worse in these experiments than in the obstacle-free scenario. During the evaluation, frequent collisions with the obstacles were observed. While some of these collisions only caused a timeout as the robot was not able to pass the obstacles, others resulted in falls. This explains the observed increase in timeouts and falls. The suspected reason for this is the nature of the carrot planner. For reference, Figure 4.1 shows the local and global plans. The employed global planner, Theta* [57], operates on a visibility graph. The points in the path can therefore be spaced at uneven distances. However, the local planner uses a point n steps ahead of the robot's current position (i.e., the start of the global plan). This can lead to scenarios where the global plan correctly finds a path around an obstacle, the local planner, however, may command the robot to walk through the obstacle. Strategies to avoid this include using the A* planner of the navigation framework [55] which operates on an even grid, or ensuring a maximum distance to the point selected on the global plan by the local planner.

Figure 6.5 shows the required times for reaching each navigation goal of the evaluation set. The obstacles in the environment for each navigation goal can not be reasonably visualized in

Approach	Successes	Timeout	Fall	Collision	Success time [s]	Success steps
2D	0.968	0.014	-	0.018	4.43 ±2.45	8.70 ±4.81
3D	0.132	0.842	0.026	-	48.95 ±22.79	96.11 ±44.73
Baseline	0.666	0.206	0.128	-	26.59 ±23.04	52.21 ±45.23

(a) Without noise

Approach	Successes	Timeout	Fall	Collision	Success time [s]	Success steps
2D	0.98	0.014	/	0.006	6.54 ±4.01	12.85 ±7.87
3D	0.106	0.876	0.018	/	55.78 ±25.46	109.51 ±49.98
Baseline	0.694	0.178	0.128	/	24.99 ±20.65	49.07 ±40.54

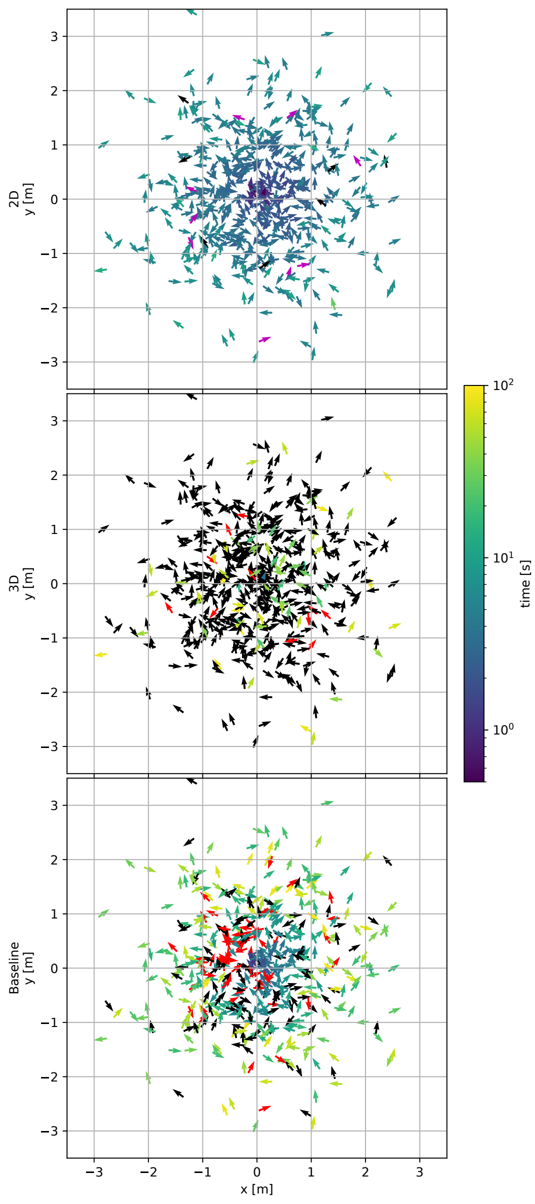
(b) With noise

Table 6.2.: Results from the evaluation scenario, which includes obstacles. The columns "Success", "Timeout", "Fall", and "Collision" show the rate at which the respective result occurred. The column "Success time" shows the mean time and its standard deviation for the successfully completed runs. The column "Success steps" presents the mean number and standard deviation of double steps for the successfully completed runs.

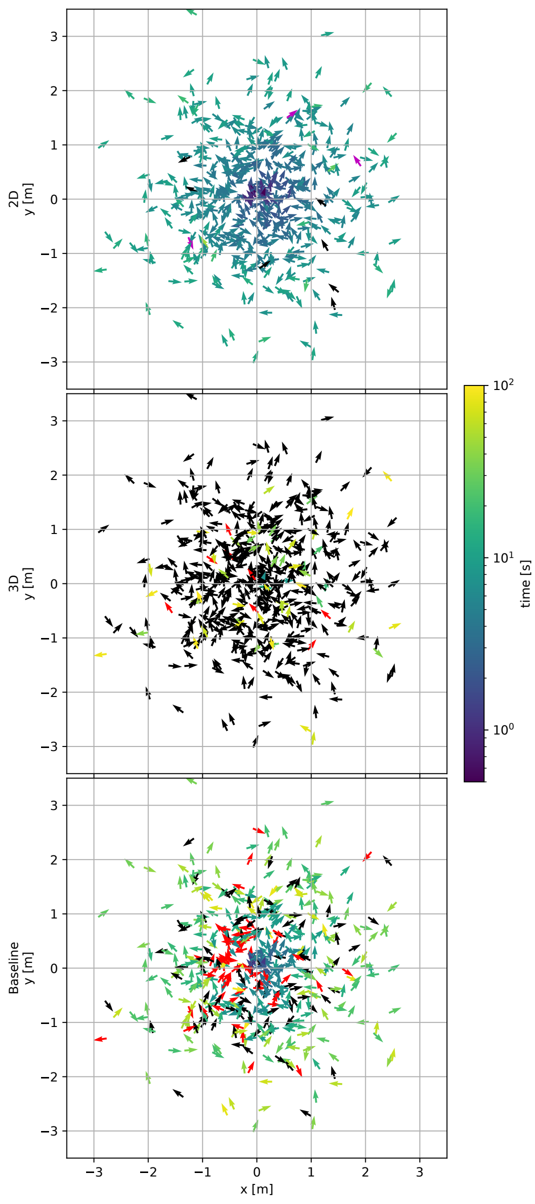
this plot. However, a correlation between the distance to a navigation goal and the required time can still be observed in the different experiments. This is expected as close navigation goals are less likely to be obstructed by an obstacle.

Figure 6.6 shows a run of the experiments where the policies trained in the 2D and 3D environment did not reach the navigation goal successfully. The 2D policies fell into a local minimum of the reward function. While the policy trained in the 3D environment managed to navigate around the obstacles, the time limit was reached before it could arrive at the goal pose. The baseline approach was successful in the experiments with and without noise added to the measurements. The 3D policy was much more conservative in footstep placement in this environment compared to the environment without obstacles. This may be caused by the increased difficulty of the environment. Taking larger steps is more likely to result in a collision with an obstacle which increases the chance of falling.

Figure 6.7 shows a run of the experiments where all approaches managed to reach the navigation goal. Again, policy trained in the 3D environment took very small footsteps. Furthermore, it struggled to reach the goal pose precisely enough while colliding with the obstacle that represents the ball. The policy in the 3D environment with noise produced a strong deviation from an optimal path. This shows that the policy did not adapt well to uncertainty, and it may be required to introduce noise during training. However, it degrades the performance of the training as the stochastic estimate of the gradient calculated in the optimization step can be a less accurate approximation of the true gradient. Furthermore, no penalty is currently given for collision in the 3D environment. This poses a problem as the approach is not encouraged to avoid collision with the ball obstacle as can be seen in the figure. A solution to this could be to terminate the episode when a collision with the ball obstacle, or possibly also with other obstacles, occurs.

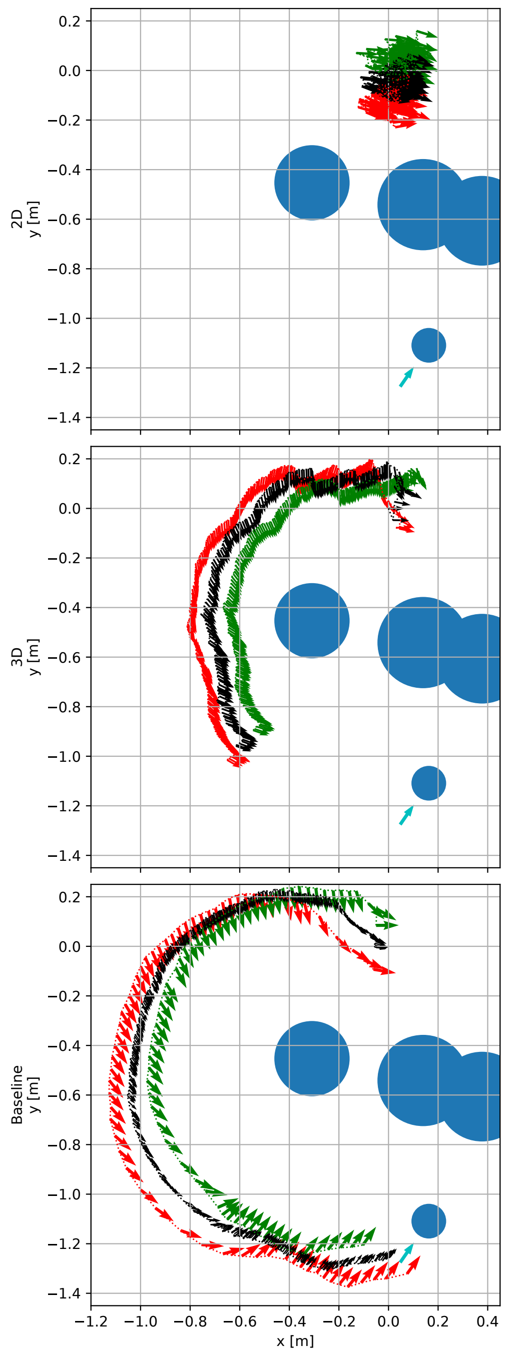


(a) Obstacles without noise

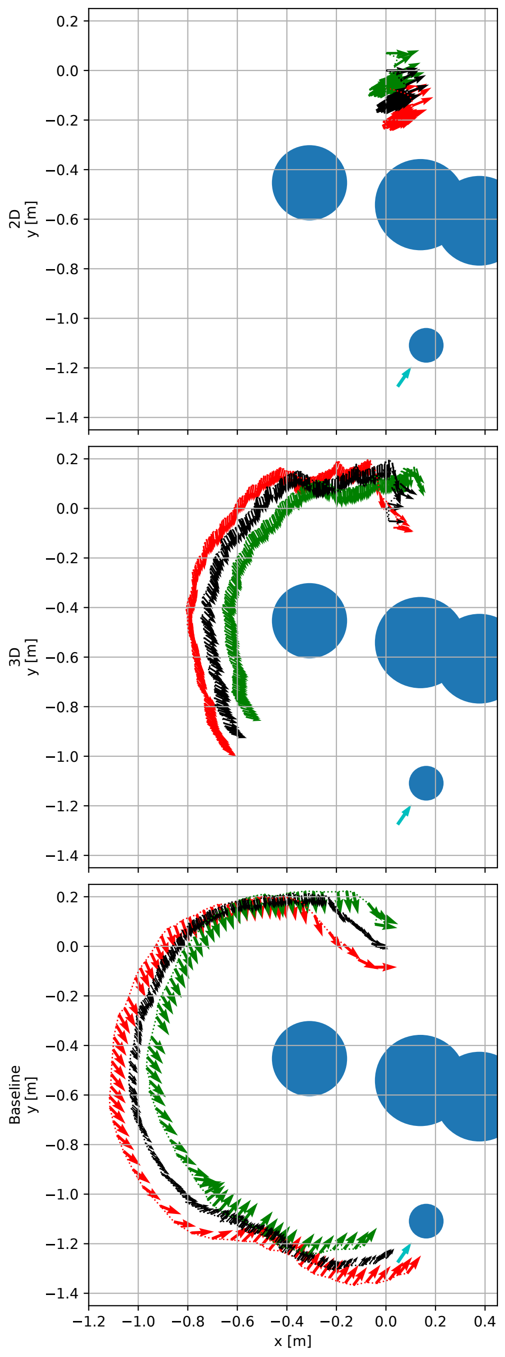


(b) Obstacles with noise

Figure 6.5.: Required time for reaching each pose in the evaluation set for the scenario with obstacles. Poses colored in red indicate that the robot fell. Poses colored in black show timeouts. Cyan-colored poses in the 2D environment indicate a collision with an obstacle. The color scale is logarithmic and different from Figure 6.3 to show the full range of measured times.

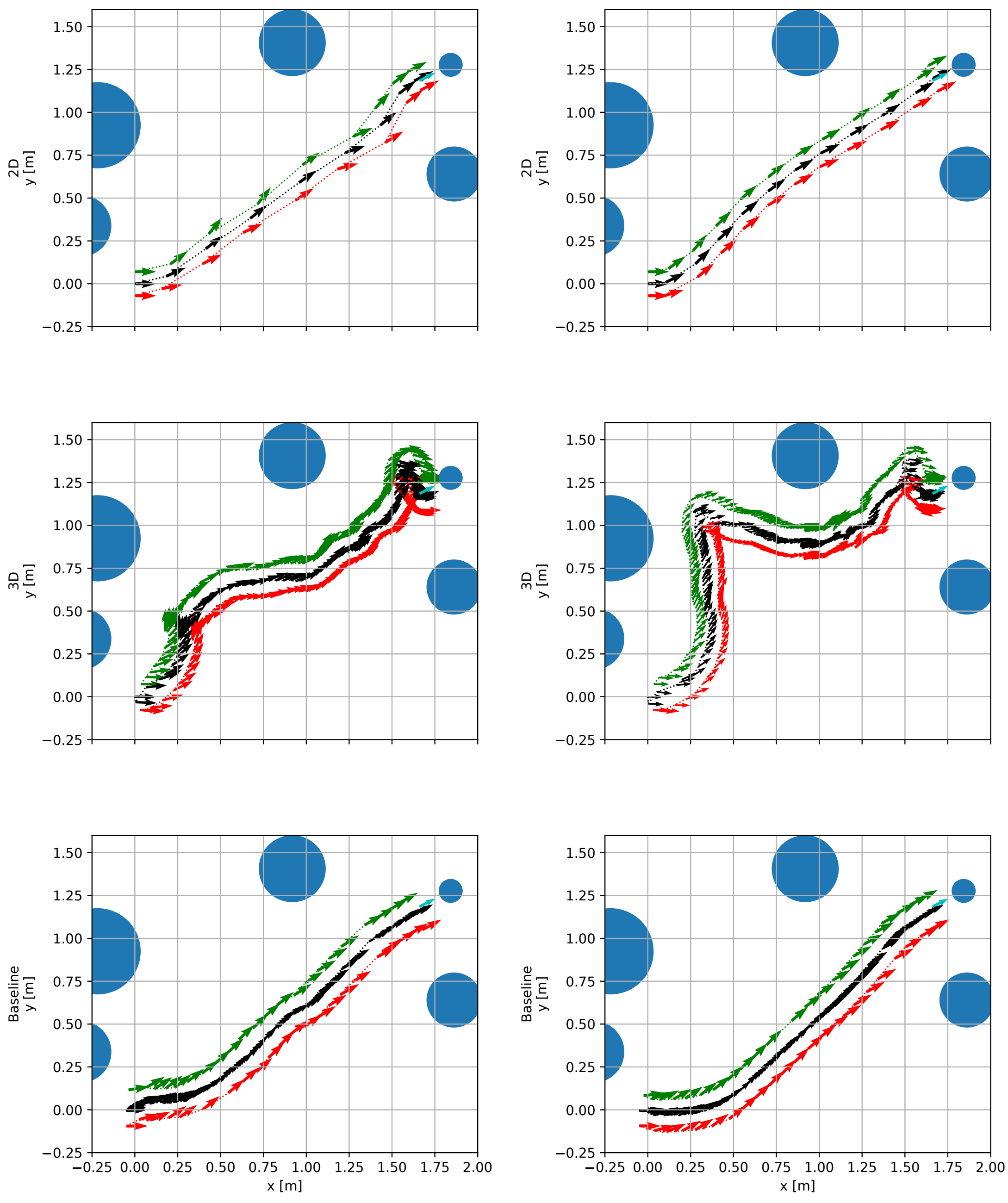


(a) Obstacles without noise



(b) Obstacles with noise

Figure 6.6.: Example run of the environment with obstacles. The policies trained in the 2D environment fall into a local minimum of the reward function and do not manage to reach the goal pose. The policy trained in the 3D environment achieves navigation around the obstacles. However, it takes very small steps, and the time limit is reached before the robot arrives at the goal. Both runs of the baseline were successful.



(a) Obstacles without noise

(b) Obstacles with noise

Figure 6.7.: Example run of the environment with obstacles. All approaches managed to reach the navigation goal.

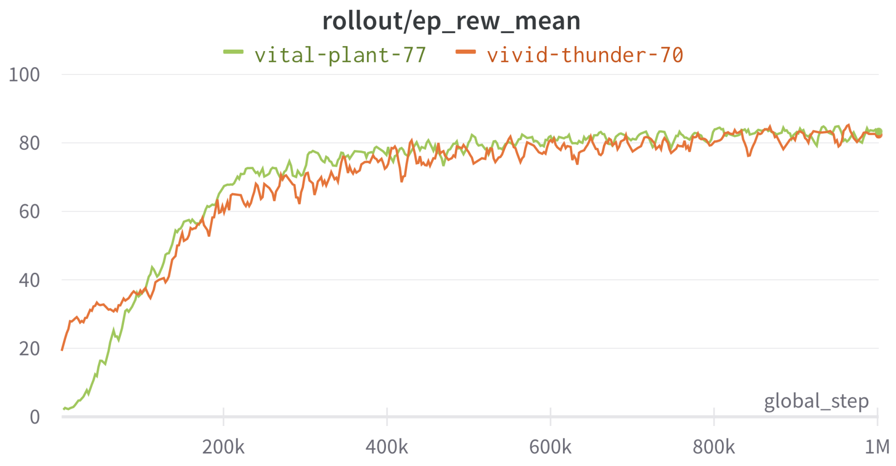


Figure 6.8.: Mean reward per episode during training when transfer learned (orange) as opposed to learning from scratch (green). While the transfer learned policy starts with a higher reward, the policy trained from scratch quickly achieves the same performance.

6.3. Transfer Learning

In general, the policy in the 2D environment performed significantly better. However, it can only do so as it does not need to consider the robot's stability and can not self-collide. The policy trained in the 2D environment was applied to the 3D environment (both without noise added to the observation) to test transferability. This was possible as they have the same observation and action space. The same set of 500 poses was used as in the other experiments. No obstacles were present in the environment. The agent reached the goal in only 9.6% of the cases. In all other cases, the robot fell. The fall usually occurred at the beginning of the episode with an average time to fall of $3.27\,\mathrm{s}$. It was caused by a rapid acceleration of the robot, which the walking engine was not capable of.

This, however, hints at the possibility of transfer learning being applicable. During a previous iteration of the environments, an experiment was conducted to evaluate whether this provides better results than training from scratch. Figure 6.8 shows the mean reward during the training of a policy trained from scratch and a transferred policy. However, no benefit was observed. The suspected reason for this is that the hyperparameters of the training algorithm are not adapted for transfer learning.

Furthermore, after the hyperparameter optimization for the 2D environment (see Section 5.6), policies using the beta distribution as output were found to perform significantly better, both in the 2D and the 3D environment. This coincides with the finding of Chou et al. [62]. However, the current implementation of the training framework stable-baselines3 [58] does not allow to use of policies with this output distribution as initializations for new trainings. The training framework was not adapted to allow this due to the limited scope of this work. Transfer learning remains an interesting topic for future work, however.

7. Discussion

Two research questions were proposed in the introduction of this thesis. Partial answers to these, which will be described in this chapter, were found in this thesis. Afterward, the limitations of the presented approach and evaluation are discussed.

Can an RL-trained policy plan more efficient footstep plans than velocity planning or classical footstep planning? The trained policy was able to outperform the baseline velocity planner in the obstacle-free environment in terms of time required to reach a navigation goal while using the same walking engine. In the environment with obstacles, the policy did produce a usable footstep plan. However, the proof of concept policy in the 2D environment showed promising results. This hints at an RL-trained policy being theoretically capable of solving this problem in the 3D environment as well. Local minima in the reward function caused by obstacles between the robot and its navigation goal were still a problem in the 2D environment. If these results are transferrable to a real robot remains to be investigated as these results were achieved in a simulated environment.

The approach was not compared to classical footstep planning. No approach compatible with the ROS2 framework used on the Wolfgang-OP was found. Adaptations of existing solutions in different frameworks were out of scope for this thesis.

Are the footstep plans generated by the policy more stable (the robot falls less often) than velocity planning or classical footstep planning? The trained policy was not able to reduce the number of falls compared to the baseline approach. However, the number of falls in the obstacle-free environment was comparably small in both approaches. In the virtual environment where the approaches were evaluated, the decrease in time required to reach a goal outweighs the downside of more frequent falls. In a physical environment, this may be different as falls can cause damage.

In the environment with obstacles, the presented approach managed to outperform the baseline in reducing falls. However, the trained policy was rarely able to reach the navigation goal.

Generally, stability and quickly reaching the navigation goal are conflicting goals. More conservative footstep plans with smaller steps are less likely to produce falls. When optimizing the parameters of the baseline approach or the reward function, a trade-off must be made between stability and quickly reaching the navigation goal.

Modifications to the observation or reward function could improve the performance of the presented approach in regard to this metric. Some directions for future research are outlined in Chapter 9.

No comparison was made between the presented approach and a classical footstep planner due to the same reasons described in the discussion of the previous research question.

7.1. Baseline Parameters

The walking parameters were optimized to produce better results on the robot platform by Bestmann et al. [2]. The local planner parameters were optimized by the Hamburg Bit-Bots for a different set of parameters used in the Humanoid League Virtual Season. As described in Section 4.3, the local planner parameters depend heavily on the parameters of the walking engine. Three options were possible to approach this problem. Firstly, both approaches could use the previous set of walking parameters. However, this would lead to worse overall performance. Secondly, the baseline could use the old parameters, and the presented approach could use the new parameters. It would not be clear from the experiments if a performance difference between the approaches is due to the underlying walking engine or the approaches themselves. Lastly, both approaches could use the optimized walking parameters. This implies that the parameters of the local planner must be adapted to allow it to function well. This manual process is error-prone and may result in local minima in the parameter space. However, the overall performance of the system is expected to be higher [2].

The third option was selected, and the parameters of the local planner were carefully tuned. Furthermore, this thesis provides foundational work for automatically optimizing the local planner parameters. The evaluation scenarios developed in this thesis can be used to assess the performance of the approach for automatic parameter optimization.

7.2. Localization

The approach used to model the noise of the localization is strongly simplified. However, it is chosen to model a worst-case scenario as Gaussian noise is applied to the measurements. The particle filter in the localization is stateful compared to the noise, as the resampled particles depend on the previous particles' likelihood. More work is required to evaluate the proposed approach in an integrated scenario with the localization or to model the localization's noise more accurately.

7.3. Sim-To-Real Transfer

The presented approach was trained and evaluated in simulation. The policy gradient method optimizes reaching the goal pose as quickly as possible with few other constraints from the reward function. While the continuity reward (see Section 5.5) tries to moderate guide the policy to moderate the change in footsteps, in practice, it can be outweighed by reaching the goal more quickly. This leads to aggressive footstep plans, which may only be stable in simulation. Before a transfer to the real robot can occur, the robot's stability during navigation must be addressed. Chapter 9 outlines possible approaches for achieving this.

The hypothesis that since the walking engine also works on real robots, the sim-to-real transfer may be eased remains to be tested in future real-world experiments.

8. Conclusion

This thesis presents a novel approach to solving the navigation problem on a humanoid robot. An RL-trained policy is combined with a walking engine based on quintic splines. The policy commands the walking engine by providing footstep poses for the next entire walking cycle.

Two RL environments were developed, the 2D environment for quick testing and iteration and the 3D environment for realistic simulation of the physical robot and the walking engine.

The trained policy manages to significantly outperform a previously successfully used baseline in the obstacle-free scenario using the same walking engine and the same simulation environment. It is able to control the walking much more effectively than the baseline approach as it learns the capabilities of the walking engine. Furthermore, no manual parameter tuning of the approach as in the baseline is required. Additionally, the policy was generally more successful as it reached all navigation goals where it did not cause the robot to fall in the given time. However, it produces more falls than the baseline. Strategies for reducing these are outlined in the following chapter.

In the scenario with obstacles, the policy performed significantly worse than the baseline approach. In both RL environments, the trained policies were often not able to overcome local minima of the reward function and, therefore, not reach the navigation goal. Possible solutions to overcome the shortcomings of the trained policy are discussed in the following chapter.

Additionally, a modified reproduction of an experiment measuring the accuracy of the localization approach is presented. It shows the noise of the localization under more realistic conditions than the original experiment. Explanations for the observed noise were proposed. Furthermore, a model of this noise was applied in the evaluation scenarios to increase realism.

More work is required to evaluate the performance of the approach compared to classical footstep planning, on different robot platforms, and in the real world.

9. Future Work

Many ideas and opportunities for future work arose during development and evaluation. This chapter outlines these in the following paragraphs, grouped by the current approach's problems.

Transfer Learning Some experiments on transfer learning were performed in this thesis. However, the method yielded no benefits compared to training the policy from scratch. A policy trained in the 2D environment can already achieve some success in the 3D environment. Modifying the 2D environment based on observations from the 3D environment to make it more realistic could allow for easing the transfer. This may include a more realistic collision model or a function that maps commanded footsteps to those executed by the walking engine. Furthermore, the hyperparameters used for adapting a policy trained in the 2D environment to the 3D environment should be investigated. Lastly, the framework must be adapted to allow policies using the beta distribution to be used as initialization for a new training process.

Obstacle Avoidance The trained policy did not perform well when obstacles were present in the environment. Three ideas are proposed to solve this problem, two of which rely on a global plan produced by a graph traversal algorithm. Firstly, calculating a global plan and using the policy to perform collision-free sections of this plan could provide a solution. However, this would not guarantee that the navigation is collision-free, and the actions decided by the policy would need to be characterized such that parameters for this approach can be found.

The second possibility is to include a global plan in the observation of the policy and reward the agent for staying close to the path. This could guide the policy around obstacles that otherwise create local minima in the reward function.

Lastly, the representation of obstacles currently relies on laserscan-like measurements. Transforming them to a costmap which is passed to a convolutional neural network (e.g., trained as the input layer of an autoencoder), could allow the policy to perform better. A similar approach has already been tested by Peng et al. [25]. In contrast, this costmap would only include local information the agent could actually observe to allow transferability to the real world.

Stability The policy caused more falls than the baseline approach. This should be avoided to prevent damage to the robot. Including information about the robot's orientation and angular velocities, which can be measured by an IMU, could help to reduce the number of falls as the agent can observe if the walking becomes unstable. Furthermore, the stabilization techniques available in the Quintic Walk engine [2] could be applied.

Additionally, the policy could be trained in conjunction with an RL-trained walking engine which is currently in development at the Hamburg Bit-Bots. This could improve both policies' performance as the walking engine experiences realistically occurring commands during training and can optimize for these.

A different approach would be to include stability metrics, such as a penalty for abnormally large deviations in roll and pitch, in the reward function to encourage the agent to perform stable motions.

Localization Uncertainty The policy developed in this thesis relied on a heavily simplified noise model of the robot's localization. Two possible solutions are proposed to increase the chance of the approach working in an integrated scenario: Firstly, the environment could include the localization approach for calculating the estimate of the robot's position. However, this would strongly increase computation time which is already a problem in the 3D environment. Secondly, the noise model of the localization could be improved by basing it on the robot's current and previous velocity. As data on the noise is easily collected in simulation, a neural network could be trained to model it.

Sim-to-Real Transfer Experiments in simulation provide evidence for the applicability of an approach. However, the real world is a much more challenging domain. Several techniques could be applied to transfer the approach:

Firstly, domain randomization could make the policy more robust against variations in the environment. This randomization could include:

- modifying the terrain to be uneven
- randomizing the weight distribution across the robot's links
- altering the parameters of the simulated actuators
- applying random forces during training
- variation of the friction coefficients

However, domain randomization can lead to suboptimal performance as the agent has to be more conservative in its actions [65].

Furthermore, an approach similar to the corrective function proposed by Hofer and Rouxel [30] could ease the transfer, as the robot would perform the steps the policy expects it to take.

Bibliography

[1] T. Moulard, "REP 120 – Coordinate Frames for Humanoid Robots (ROS.org)," https://www.ros.org/reps/rep-0120.html, 2012, Accessed on 12.08.2022.

[2] M. Bestmann and J. Zhang, "Bipedal walking on humanoid robots through parameter optimization," in *RoboCup 2022: Robot World Cup XXV*, Jul. 2022.

[3] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa, "Robocup: The robot world cup initiative," in *Proceedings of the First International Conference on Autonomous Agents*, 1997, pp. 340–347.

[4] M. Bestmann, J. Güldenstein, F. Vahl, and J. Zhang, "Wolfgang-OP: A robust humanoid robot platform for research and competitions," in *2020 IEEE-RAS 20th International Conference on Humanoid Robots (Humanoids)*, 2021, pp. 90–97.

[5] Cyberbotics Ltd., "Webots," http://www.cyberbotics.com, Open-source Mobile Robot Simulation Software, Accessed on 12.08.2022.

[6] RoboCup Humanoid League Technical Committee, "Humanoid League Virtual Season 2021/22," https://humanoid.robocup.org/hl-vs2022/, accessed on 12.08.2022.

[7] N. Perrin, "Biped footstep planning," in *Humanoid Robotics: A Reference*, A. Goswami and P. Vadakkepat, Eds. Dordrecht: Springer Netherlands, 2019, pp. 1697–1717.

[8] J. Chestnutt, M. Lau, G. Cheung, J. Kuffner, J. Hodgins, and T. Kanade, "Footstep Planning for the Honda ASIMO Humanoid," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, Apr. 2005, pp. 629–634.

[9] J. Garimort, A. Hornung, and M. Bennewitz, "Humanoid navigation with dynamic footstep plans," in *2011 IEEE International Conference on Robotics and Automation*. Shanghai, China: IEEE, May 2011, pp. 3982–3987.

[10] A. Hornung, A. Dornbush, M. Likhachev, and M. Bennewitz, "Anytime search-based footstep planning with suboptimality bounds," in *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*. Osaka, Japan: IEEE, Nov. 2012, pp. 674–679.

[11] J. Chestnutt, K. Nishiwaki, J. Kuffner, and S. Kagami, "An adaptive action model for legged navigation planning," in *2007 7th IEEE-RAS International Conference on Humanoid Robots*, Nov. 2007, pp. 196–202.

[12] D. Schleich, M. Hosseini, D. Rodriguez, and S. Behnke, "Real-time footstep planning with CoM dynamics," IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids), Late Breaking Report, Oct. 2019.

[13] E. Yoshida, I. Belousov, C. Esteves, and J.-P. Laumond, "Humanoid motion planning for dynamic tasks," in *5th IEEE-RAS International Conference on Humanoid Robots, 2005.*, Dec. 2005, pp. 1–6.

[14] J. Chestnutt and J. Kuffner, "A tiered planning strategy for biped navigation," in *4th IEEE/RAS International Conference on Humanoid Robots, 2004.*, vol. 1. Santa Monica, CA, USA: IEEE, 2004, pp. 422–436.

[15] J.-S. Gutmann, M. Fukuchi, and M. Fujita, "Real-time path planning for humanoid robot navigation," in *IJCAI*, 2005, pp. 1232–1237.

[16] N. Perrin, O. Stasse, L. Baudouin, F. Lamiraux, and E. Yoshida, "Fast Humanoid Robot Collision-Free Footstep Planning Using Swept Volume Approximations," *IEEE Transactions on Robotics*, vol. 28, no. 2, pp. 427–439, Apr. 2012.

[17] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects: Steven m. lavalle, iowa state university, a james j. kuffner, jr., university of tokyo, tokyo, japan," *Algorithmic and Computational Robotics*, pp. 303–307, 2001.

[18] M. Wermelinger, P. Fankhauser, R. Diethelm, P. Krüsi, R. Siegwart, and M. Hutter, "Navigation planning for legged robots in challenging terrain," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2016, pp. 1184–1189.

[19] B. Styler and R. Simmons, "Plan-time multi-model switching for motion planning," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 27, 2017.

[20] M. Brandao and I. Havoutis, "Learning sequences of approximations for hierarchical motion planning," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30, 2020, pp. 508–516.

[21] A. Herdt, H. Diedam, P.-B. Wieber, D. Dimitrov, K. Mombaur, and M. Diehl, "Online Walking Motion Generation with Automatic Footstep Placement," *Advanced Robotics*, vol. 24, no. 5-6, pp. 719–737, Jan. 2010.

[22] H. Diedam, D. Dimitrov, P.-B. Wieber, K. Mombaur, and M. Diehl, "Online walking gait generation with adaptive foot positioning through Linear Model Predictive control," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2008, pp. 1121–1126.

[23] S. Kuindersma, R. Deits, M. Fallon, A. Valenzuela, H. Dai, F. Permenter, T. Koolen, P. Marion, and R. Tedrake, "Optimization-based locomotion planning, estimation, and

control design for the atlas humanoid robot," *Autonomous Robots*, vol. 40, no. 3, pp. 429–455, Mar. 2016.

[24] G. Bledt and S. Kim, "Implementing Regularized Predictive Control for Simultaneous Real-Time Footstep and Ground Reaction Force Optimization," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Nov. 2019, pp. 6316–6323.

[25] X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne, "DeepLoco: Dynamic locomotion skills using hierarchical deep reinforcement learning," *ACM Transactions on Graphics*, vol. 36, no. 4, pp. 1–13, Jul. 2017.

[26] V. Tsounis, M. Alge, J. Lee, F. Farshidian, and M. Hutter, "DeepGait: Planning and Control of Quadrupedal Gaits Using Deep Reinforcement Learning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3699–3706, Apr. 2020.

[27] S. Gangapurwala, M. Geisert, R. Orsolino, M. Fallon, and I. Havoutis, "Rloc: Terrain-aware legged locomotion using reinforcement learning and optimal control," *IEEE Transactions on Robotics*, pp. 1–20, 2022.

[28] W. Yu, D. Jain, A. Escontrela, A. Iscen, P. Xu, E. Coumans, S. Ha, J. Tan, and T. Zhang, "Visual-locomotion: Learning to walk on complex terrains with vision," in *5th Annual Conference on Robot Learning*, 2021.

[29] M. Missura and S. Behnke, "Online learning of foot placement for balanced bipedal walking," in *2014 IEEE-RAS International Conference on Humanoid Robots*. Madrid, Spain: IEEE, Nov. 2014, pp. 322–328.

[30] L. Hofer and Q. Rouxel, "An operational method toward efficient walk control policies for humanoid robots," in *Twenty-Seventh International Conference on Automated Planning and Scheduling*, 2017.

[31] K. Lobos-Tsunekawa, F. Leiva, and J. Ruiz-del-Solar, "Visual Navigation for Biped Humanoid Robots Using Deep Reinforcement Learning," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3247–3254, Oct. 2018.

[32] A. Brandenburger, D. Rodriguez, and S. Behnke, "Mapless Humanoid Navigation Using Learned Latent Dynamics," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2021, pp. 1555–1561.

[33] T. Li, N. Lambert, R. Calandra, F. Meier, and A. Rai, "Learning Generalizable Locomotion Skills with Hierarchical Reinforcement Learning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, May 2020, pp. 413–419.

[34] D. Jain, A. Iscen, and K. Caluwaerts, "Hierarchical Reinforcement Learning for Quadruped Locomotion," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Nov. 2019, pp. 7551–7557.

[35] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT press, 2018.

[36] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, no. 5, pp. 834–846, Sep. 1983.

[37] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[38] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[39] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*. PMLR, 2015, pp. 1889–1897.

[40] M. Schwarz, J. Pastrana, P. Allgeuer, M. Schreiber, S. Schueller, M. Missura, and S. Behnke, "Humanoid teensize open platform nimbro-op," in *Robot Soccer World Cup*. Springer, 2013, pp. 568–575.

[41] F. Vahl, J. Gutsche, M. Bestmann, and J. Zhang, "YOEO — You only encode once: A CNN for embedded object detection and semantic segmentation," in *2021 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 2021.

[42] M. Poppinga and M. Bestmann, "DSD - Dynamic Stack Decider," *International Journal of Social Robotics*, vol. 14, no. 1, pp. 73–83, Jan. 2022.

[43] S. Stelter, M. Bestmann, N. Hendrich, and J. Zhang, "Fast and reliable stand-up motions for humanoid robots using spline interpolation and parameter optimization," in *2021 20th International Conference on Advanced Robotics (ICAR)*. IEEE, 2021, pp. 253–260.

[44] T. Engelke, "Learning to Kick from Demonstration with Deep Reinforcement Learning," Bachelor Thesis, Universität Hamburg, 2021.

[45] M. Bestmann and J. Zhang, "Humanoid control module: An abstraction layer for humanoid robots," in *2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*. IEEE, 2020, pp. 263–268.

[46] Hamburg Bit-Bots, "bitbots_meta," https://github.com/bit-bots/bitbots_meta, Accessed on 12.08.2022.

[47] Q. Rouxel, G. Passault, L. Hofer, S. N'Guyen, and O. Ly, "Rhoban hardware and software open source contributions for robocup humanoids," in *Proceedings of 10th Workshop on Humanoid Soccer Robots, IEEE-RAS Int. Conference on Humanoid Robots, Seoul, Korea*, 2015.

[48] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 2623–2631.

[49] J. Hartfill, "Feature-Based Monte Carlo Localization in the RoboCup Humanoid Soccer League," Master's thesis, Universität Hamburg, 2019.

[50] Hamburg Bit-Bots, "bitbots_navigation," https://github.com/bit-bots/bitbots_navigation, Accessed on 12.08.2022.

[51] H. R. Kam, S.-H. Lee, T. Park, and C.-H. Kim, "Rviz: A toolkit for real domain data visualization," *Telecommunication Systems*, vol. 60, no. 2, pp. 337–345, 2015.

[52] R. Smith *et al.*, "Open dynamics engine," 2007.

[53] F. Muratore, F. Treede, M. Gienger, and J. Peters, "Domain Randomization for Simulation-Based Policy Optimization with Transferability Assessment," in *Proceedings of The 2nd Conference on Robot Learning*. PMLR, Oct. 2018, pp. 700–713.

[54] S. Koos, J.-B. Mouret, and S. Doncieux, "The transferability approach: Crossing the reality gap in evolutionary robotics," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 1, pp. 122–145, 2012.

[55] S. Macenski, F. Martín, R. White, and J. G. Clavero, "The Marathon 2: A Navigation System," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2020, pp. 2718–2725.

[56] M. Colledanchise and P. Ögren, *Behavior Trees in Robotics and AI: An Introduction*. CRC Press, 2018.

[57] A. Nash, S. Koenig, and C. Tovey, "Lazy theta*: Any-angle path planning and path length analysis in 3d," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 24, no. 1, 2010, pp. 147–154.

[58] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[59] A. Raffin, "Rl baselines3 zoo," https://github.com/DLR-RM/rl-baselines3-zoo, 2020, accessed on 12.08.2022.

[60] P. Allgeuer and S. Behnke, "Fused angles: A representation of body orientation for balance," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2015, pp. 366–373.

[61] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li, "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.

[62] P.-W. Chou, D. Maturana, and S. Scherer, "Improving Stochastic Policy Gradients in Continuous Control with Deep Reinforcement Learning using the Beta Distribution," in *Proceedings of the 34th International Conference on Machine Learning*. PMLR, Jul. 2017, pp. 834–843.

[63] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," *Advances in neural information processing systems*, vol. 24, 2011.

[64] Hamburg Bit-Bots, "bitbots_behavior," https://github.com/bit-bots/bitbots_behavior, Accessed on 12.08.2022.

[65] B. Mehta, M. Diaz, F. Golemo, C. J. Pal, and L. Paull, "Active domain randomization," in *Conference on Robot Learning*. PMLR, 2020, pp. 1162–1176.

Appendices

A. Localization Experiments

The results of the localization experiments reproduced from Hartfill [49] are visualized on the following pages. Experiment 1 is the path represented by the blue line, experiment 2 corresponds to the green line in Figure 1 taken from the thesis. Experiment 0 presented in Section 6.1 corresponds to the red line. Experiment 3 consists of the robot turning while being positioned at the blue cross. The baseline approach described in Chapter 4 is used to navigate from the starting pose to the goal pose which causes the trajectories to not line up exactly. In the original thesis, the robot was simply teleported in simulation to extract the measurements.

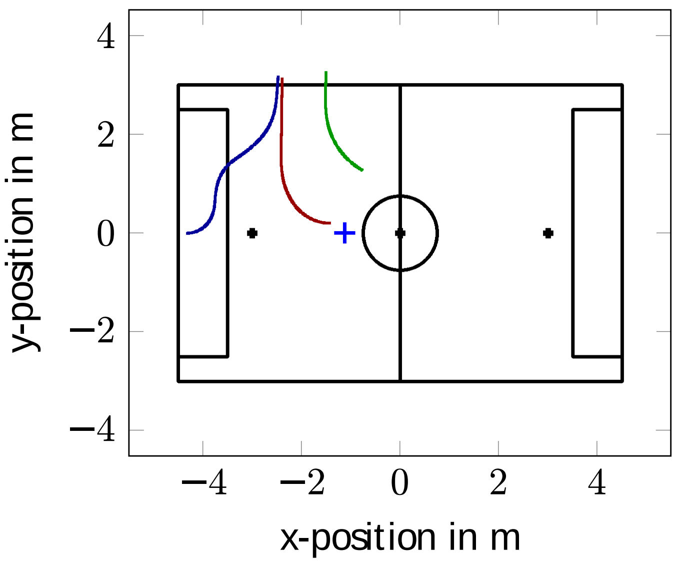
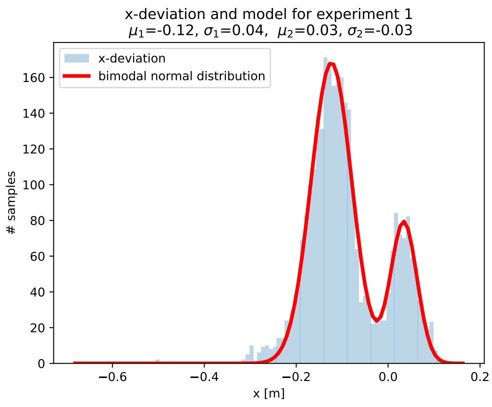
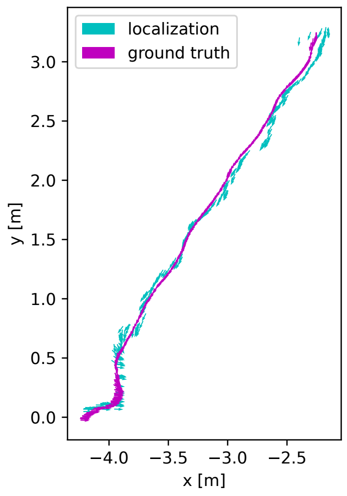


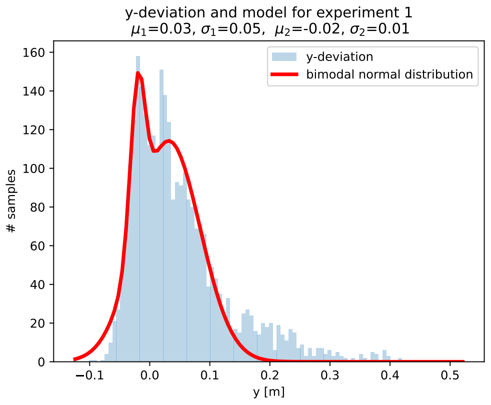
Figure 1.: Trajectories of the robot for the original experiments. The figure is taken from the thesis introducing the localization approach [49].



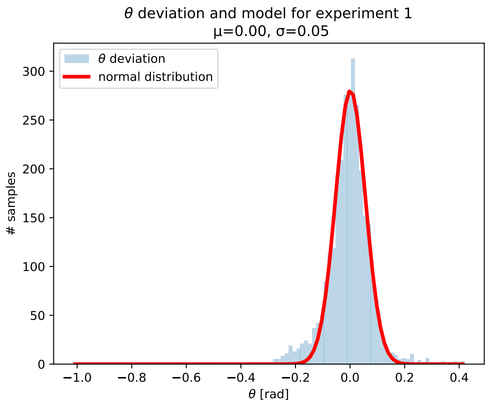
(b) Deviation between ground truth and localization pose in x-direction in the ground truth reference frame.



(a) Ground truth path taken from the simulator (magenta) and path measured by the localization pipeline (cyan) of one of the runs.

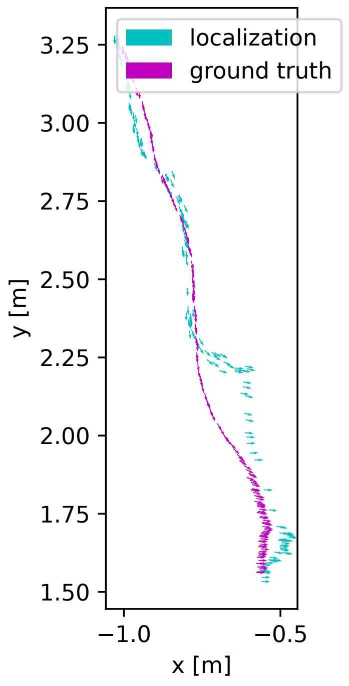


(c) Deviation between ground truth and localization pose in y-direction in the ground truth reference frame.

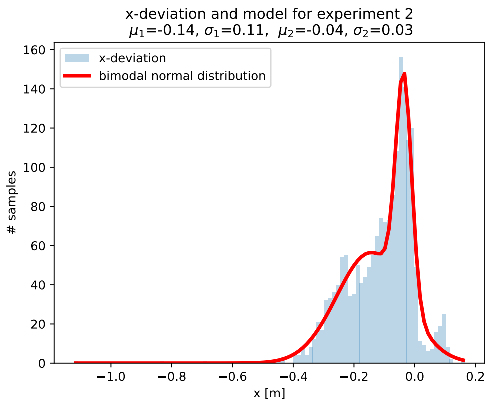


(d) Angle deviation between the ground truth and the localization pose.

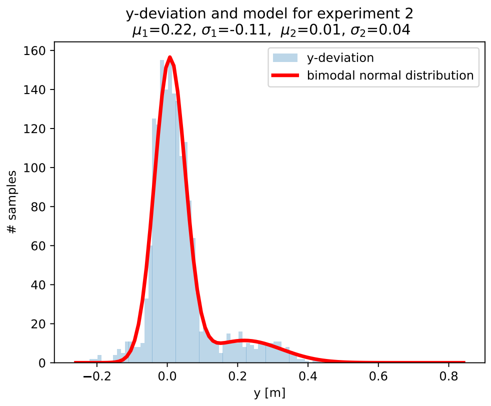
Figure 2.: Results in deviation between the ground truth and the localization estimate in localization experiment 1. The robot is walking from the sideline to the goalkeeper position.



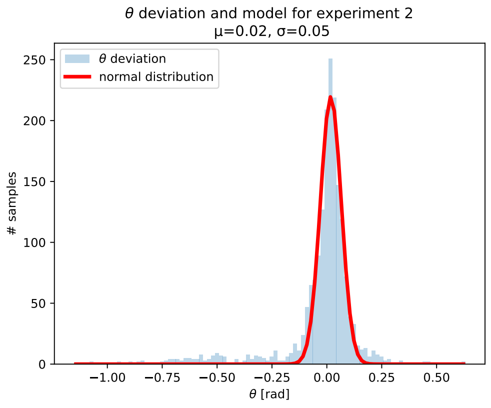
(a) Ground truth path taken from the simulator (magenta) and path measured by the localization pipeline (cyan) of one of the runs.



(b) Deviation between ground truth and localization pose in x-direction in the ground truth reference frame.

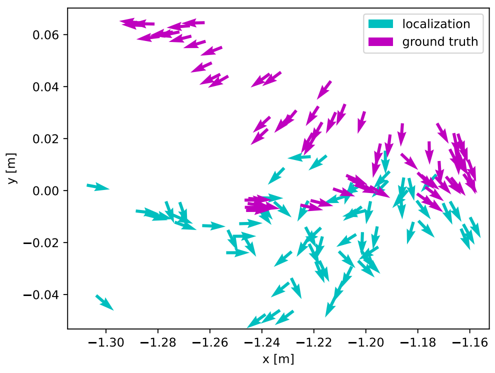


(c) Deviation between ground truth and localization pose in y-direction in the ground truth reference frame.

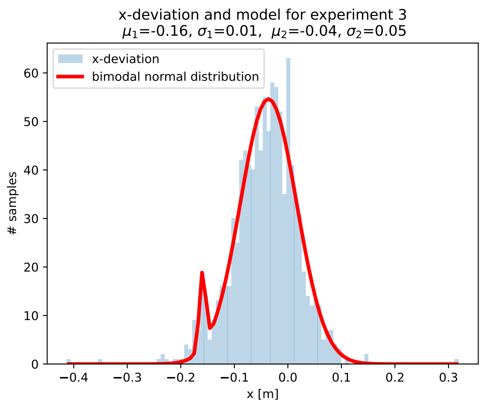


(d) Angle deviation between the ground truth and the localization pose.

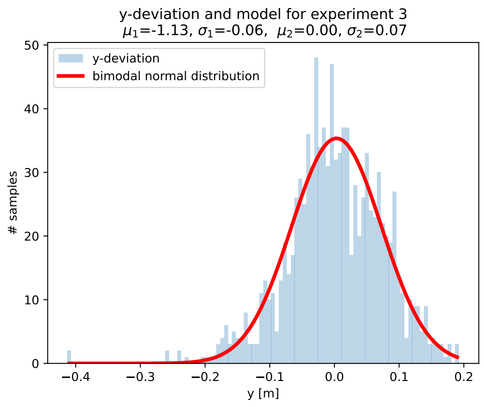
Figure 3.: Results in deviation between the ground truth and the localization estimate in localization experiment 2. The robot is walking from the sideline to a pose next to the center circle.



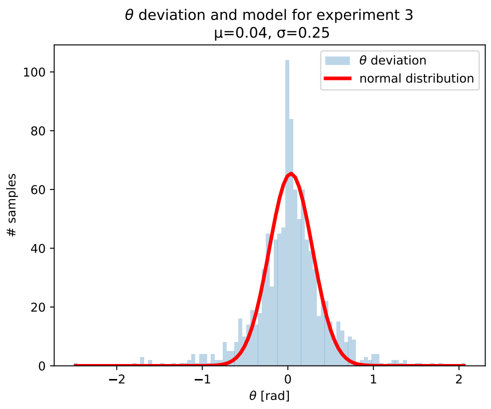
(a) Ground truth path taken from the simulator (magenta) and path measured by the localization pipeline (cyan) of one of the runs.



(b) Deviation between ground truth and localization pose in x-direction in the ground truth reference frame.



(c) Deviation between ground truth and localization pose in y-direction in the ground truth reference frame.



(d) Angle deviation between the ground truth and the localization pose.

Figure 4.: Results in deviation between the ground truth and the localization estimate in localization experiment 3. The robot is rotating in place.

Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Masterstudiengang Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel — insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen — benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der elektronischen Abgabe entspricht.

Hamburg, den 17.08.2022

Jasper Güldenstein

Veröffentlichung

Ich stimme der Einstellung der Arbeit in die Bibliothek des Fachbereichs Informatik zu.

Hamburg, den 17.08.2022

Jasper Güldenstein