
Rheinische
Friedrich-Wilhelms-
Universität Bonn

Institute for Computer Science
Department VI
Autonomous Intelligent Systems

# Rheinische Friedrich-Wilhelms-Universität Bonn

## Master Thesis

# Online Path Planning using Learned Latent Dynamics from Visual and Non-Visual Observations

*Author:*
André Brandenburger

*First Examiner:*
Prof. Dr. Sven Behnke

*Second Examiner:*
Prof. Dr. Maren Bennewitz

Date: September 20, 2020

# Declaration

I hereby declare that I am the sole author of this thesis and that none other than the specified sources and aids have been used. Passages and figures quoted from other works have been marked with appropriate mention of the source.

Place, Date

Signature

# Abstract

In this thesis, we propose a novel learning approach for bipedal path planning using visual and non-visual observations. The planning can be executed online thanks to a learned latent dynamics model of the environment. In contrast to previous works, we do not restrict the observation space only on color images, but we incorporate other non-visual observations such as angular accelerations from a gyroscope or detection outputs from a computer-vision pipeline. Furthermore, we introduce a termination likelihood prediction, which facilitates the agents anticipation of success and failure. Based on the fused observations, the agent, i.e. the NimbRo-OP2X robot, is able to navigate a scene with static and dynamic obstacles in order to reach a target position. The training of the model is performed in simulation, but the resulting policy is successfully transferred to the real world by using abstract image observations as inputs for the agent. In the evaluation, our model generates faster trajectories than a exteroceptive A$^*$ controller. In addition, our approach outperforms the *Dreamer* model on the simulated task, while it closely resembles the performance of the well-tested NimbRo obstacle avoidance algorithm in the real world.

# Zusammenfassung

In dieser Masterarbeit schlagen wir einen neuartigen Ansatz zum Erlernen der zweibeinigen Pfadplanung unter Nutzung von visuellen und nicht-visuellen Beobachtungen vor. Die resultierende Strategie kann dank eines gelernten latenten Dynamikmodells in Echtzeit ausgeführt werden. Im Gegensatz zu vorherigen Arbeiten beschränken wir den Beobachtungsraum nicht ausschließlich auf Farbbilder, sondern beziehen auch andere nicht-visuelle Beobachtungen mit ein. Diese beinhalten beispielsweise Gyroskop-Messungen und Detektionen eines Computer-Vision Algorithmus. Darüber hinaus wird die Terminierungswahrscheinlichkeit approximiert, um dem Roboter zu ermöglichen die Erfolg und Misserfolg zu antizipieren. Anhand der fusionierten Beobachtungen kann der NimbRo-OP2X Roboter durch verschiedene Szenen navigieren, während er sowohl statischen als auch dynamischen Hindernissen ausweicht um einen Zielort zu erreichen. Das Modell wurde ausschließlich in Simulationen trainiert, die resultierende Strategie wird jedoch mithilfe abstrakter Bildbeobachtungen erfolgreich in die echte Welt übertragen. Im Verlauf unserer Auswertung generiert unser Agent schnellere Trajektorien als der exterozeptive A*-Regler. Zudem übertrifft unser Ansatz das *Dreamer* Modell in der Simulation und ähnelt in der echten Welt stark der bewährten NimbRo Hindernisvermeidung.

# Contents

# Acknowledgments

First and foremost I would like to thank Diego Rodriguez for his supervision and support throughout the course of this thesis, as well as for many inspiring conversations. Furthermore I would like to thank Prof. Behnke for his important input and the remaining NimbRo soccer team, especially Hafez Farazi for his help on the technical integration with the existing code. Finally, I would like to express special thanks to my family for their support throughout the whole course of my studies.

# 1. Introduction

The use of robotics in every-day applications is becoming more common, which leads to harder tasks and environments the agent has to work in. This may manifest in projected observations, such as images, which can be harder to interpret than other, possibly exteroceptive measurements that are generally impossible to retrieve from the real world. On the other hand, images summarize the environment state concisely and show the spatial relation between objects in the scene, which is imperative for path planning tasks. Moreover, cameras are typically cheap and are available in small form factors, making them favorable for application in robotics.

Using traditional artificial intelligence methods like tree-search for planning on image observations is either expensive, or impossible due poor approximations. Thus, (deep-)learning based approaches are strongly on the advance, since they are highly adaptive and fairly easy to transfer to different tasks. This has been proven by many popular works, such as AlphaZero[1], D4PG[2], PlaNet[3] or Dreamer[4]. AlphaZero, a hybrid approach between a learned model and a tree-search method, is able competitively play games like Chess or Shogi and was able beat a human professional in the game of Go[1, 5, 6]. Additionally, D4PG is able to produce a bipedal gait for navigation through a complex parcour based on image inputs and was also applied on other complex tasks[2, 3]. PlaNet and Dreamer also utilize image observations and are able to solve a high variety of tasks, including some of the retro gaming platform Atari, while reducing the training effort drastically compared to D4PG[3, 4].

These methods, however, typically require a large amount of training samples, which is often impossible to acquire in real-world applications. Consequently, these approaches are trained in simulators and a real-world transfer has to be done with little or no re-training.

In general, all of the above methods have been applied on planning tasks, meaning the agent tries to select an optimal action based on predictions of possible futures. This procedure is reiterated, resulting in an agent that is able work on a given task. Still, on continuous environments, the agents often only learn reactive control policies, since they are trained without accounting for long-term planning over different trajectories. This is different, however in works on discrete environ-

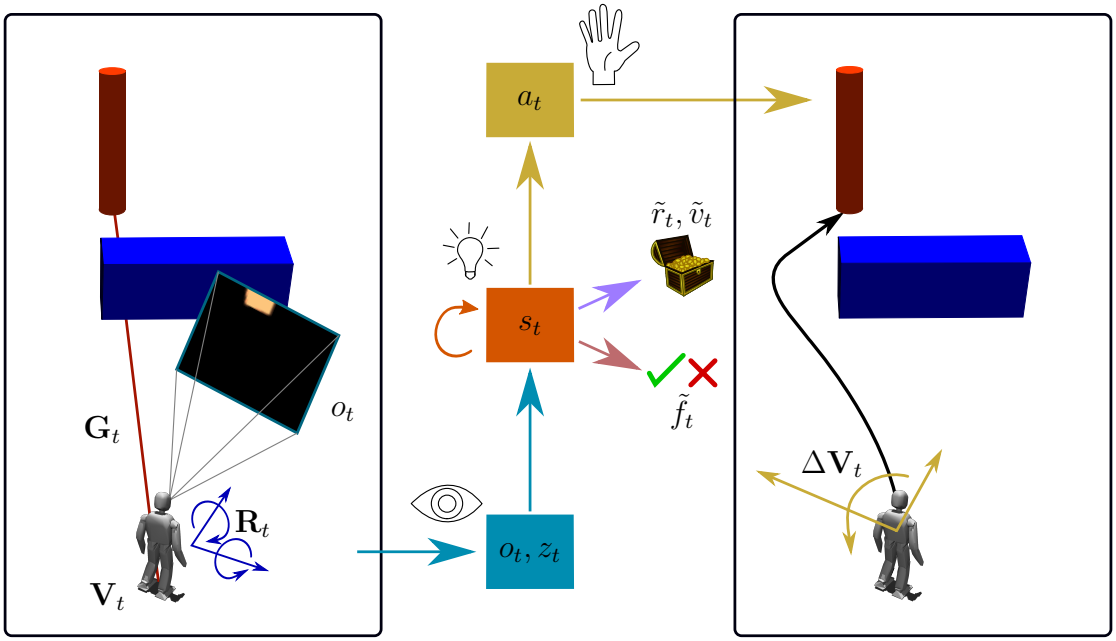


Figure 1.1: Overview of the approach. The agent utilizes a grayscale image input $o_t$ which is fused with non-visual observations $z_t$, consisting of the robot rotation $\mathbf{R}_t$, the relative goal position $\mathbf{G}_t$ and the current gait velocity $\mathbf{V}_t$. Using this input, the agent constructs a meaningful latent representation $s_t$, which is used to predict rewards $\tilde{r}_t$, values $\tilde{v}_t$ and termination likelihood $\tilde{f}_t$. Furthermore, the actor can utilize the latent state $s_t$ to predict an action $a_t$ for the next environment step, which consists of a change in gait velocity $\Delta\mathbf{V}_t$. The agent is trained to approach the target(red) as fast as possible, while avoiding obstacles in the scene(blue).

ments, such as the game Go, where the state evolution follows simple rules and the discrete structure can be used to apply decision tree-based methods.

In this thesis, we solve a visual control task, where the NimbRo-OP2X[7] robot navigates a parcour of obstacles to reach a predefined target location. The agent relies on both, image data to observe the objects in the scene and non-visual observations to infer gait stability, velocity and the relative target location. Additionally, we propose a termination model, which predicts the probability of termination for each environment state. This leads to a robust approach, which can be trained with a comparably small amount of samples and exceeds the performance of our baseline methods. An overall summary of our method is visualized in Fig. 1.1.

Since we investigate a continuous visual control task, we will first discuss previous works in this domain. In Chapter 3, we will review preliminary knowledge

on which we will build upon in the subsequent chapter. Then, the experimental results will be discussed, including both, simulated and real-world evaluation. In the end, we will conclude with proposals for future work and a short summary.

# 2. Related Work

In this thesis, we investigate *visual control* tasks, where the agent has to output actions based on image observations. Depending on the complexity of the task, it is possible to perform visual control using an analytical open-loop approach, as in many previous works[8–10].

Becerra *et al.*[10] steer a mobile robot to reach a target location by minimizing the error between a currently recorded image and an image from the target location. By projecting the camera image through a cone-shaped mirror, the robot acquires nearly a 360° view of the scene, which is then used to successfully control the robot, such that it reaches the target location. Nevertheless, since the method strongly relies on the observation of the whole scene at every time-step, the application in cluttered scenes or on different robotic platforms will not be possible.

Xu *et al.*[9] combine image segmentation and a feature selection algorithm to build a control policy for an arc welding robot. By fusing position and velocity estimates with a captured image, the method proves to successfully track the welding seam using a PID controller and thus solves the posed task. However, this method will solely work on the welding application and requires hand-tuning the gain matrices of the controller.

To address the shortcomings of the previous approaches, such as poor transferability, it is possible to utilize learning based methods. The broad field of Reinforcement Learning (RL) has been rapidly developing in the last years, where deep neural network approaches have become more and more popular. By modeling the environment as a function yielding an observation and a reward at each step given an action, an agent can explore the environment with the goal to optimize the expected reward. In order to work on a task, the agent can take actions from the action space of the environment, influencing its state evolution.

Generally, finding a good approximation of the state-action value (Q-value) of the environment is one of the main objectives in most RL algorithms. These types of approaches are typically called *Q-Learning* and can even find optimal solutions in sufficiently simple environments.

One of the most popular Q-Learning methods is the Deep Q-Network (DQN) by Mnih *et al.*[11], where the Q-value is estimated using a deep neural network.

At each step, the DQN agent selects the action that maximizes the accumulated Q-value for a given horizon under the learned Q-model. Furthermore, it utilizes experience replay, where the generated state trajectories are stored and used for training at a later point. Since this drastically improves data efficiency, experience replay is also popular in other present approaches.

A recent breakthrough in the field of RL is the defeat of a professional Go player by an artificial intelligence agent, namely AlphaGo by Silver *et al.*[6]. Using deep convolutional neural networks, the authors train policy and state-value networks, which are initially based on data collected from human players on an online Go Server. Using distributed Monte-Carlo Tree Search (MCTS), AlphaGo traverses the game tree to optimize the action value with addition to an exploration term. During training, the model plays matches against earlier versions of itself, thus learning new strategies, which can exceed human performance[6]. Subsequently, Silver *et al.*[6] increase the complexity of the task by removing the initial data collected from human players, resulting in AlphaGoZero, which is able to learn the game of Go from scratch by refining the learned policy network using MCTS at each step[5]. AlphaZero has been proposed by Silver *et al.*[1] as a generalization of AlphaGoZero. It removes some task-specific limitations of the model to apply it to tasks such as chess and shogi. Despite the unquestionable success of these approaches, they come with strong limitations for real-time tasks, as generally the agent is given one minute to evaluate its next step[1]. Also, hardware requirements and long training time can be restrictive for research and applications on a smaller scale. Lastly, these tree-based methods require a discrete action space, which often does not reflect the real world.

In addition to the above approaches, there is the popular family of *Policy Gradient*(PG) methods, which updates a policy by performing gradient ascent on a value function. In contrast to traditional PG methods which work under stochastic policies, the Deterministic Policy Gradient (DPG) algorithm assumes deterministic actions[12]. This reduces the complexity of the value calculation, since it is only necessary to iterate the state space instead of the state-action space[12]. Nevertheless, due to the strict nature of action selection, an additional sampling step is required to ensure sufficient exploration. Deep DPG (DDPG), an extension of the DPG algorithm was proposed by Lillicrap *et al.*[13] and combines the concepts of DPG and DQN. In contrast to DQN, DDPG works on a continuous action space and performs a soft update on the used networks. Furthermore, Distributed Distributional DDPG (D4PG) further extends the DDPG approach to incorporate multi-step returns and a non-uniformly sampled replay buffer. Additionally, the Q-value is modeled stochastically and multiple isolated actors are introduced. Together, these actors update a joint target network. In general, D4PG is more

robust than its predecessors and produces state-of-the-art performance on a series of tasks such as bipedal gait generation for navigating a complex parcour[13] or developing a quadruped gait[3, 4].

Since agents are more frequently required to work in human environments, image observations are typically imperative to achieve a reliable performance. Thus, it is crucial to enable agents to process images, which is one of the most important sources of information for human planning. Both DQN and D4PG have been applied to visual control tasks and the latter is able to produce competitive performance. However, training the D4PG algorithm on images requires large amount of data and training time. This can be explained by the absence of a learned world model, which could encode state evolution in a concise manner. Instead, the agent needs to work on a small view of previous images and extract all information from scratch. This lack of memory can be seen as one of the largest drawbacks of model-free approaches.

Model-based methods on the other hand are typically more sample efficient, since they model the state evolution explicitly. This ensures that the information of previous steps can be reused and even be refined over multiple iterations. Additionally, the state evolution model can be used to predict the state multiple steps into the future, enabling the employment of traditional planning algorithms[3]. However, these models are generally harder to train, as their performance strongly depends on the extracted features and the accuracy of their potentially complex dynamics.

A prominent example of model-based Reinforcement Learning is *World Models* by Ha & Schmidhuber[14]. The authors utilize a Variational Autoencoder (VAE) to gather the image input in a lower dimensional latent state, whose evolution over time is parameterized by a *Recurrent Neural Network* (RNN)[14]. Using both, the RNN memory state and the compressed observation, an action is chosen by a linear single-layer feed-forward network, which maximizes the aggregated expected reward. This way, Ha & Schmidhuber[14] are able to solve complex visual control tasks, such as subtasks in the computer game *Doom*.

Considering the task of path planning from visual inputs, many works suggest that the extraction of depth information is crucial[15–17]. Xie *et al.*[15] propose a depth prediction network which, based on the monocular RGB input, predicts the corresponding depth field. Using these depth approximations, a Q-value function is predicted and an action is chosen correspondingly. The model is successfully applied in simulation and on a real robot. The authors underline that the depth predictions are very inaccurate, but seem to be sufficient for solving the task[15].

A different approach by Schaub *et al.*[18] utilizes *optical flow* to predict the movement of objects in the scene. More precisely, the authors extract key points

from the calculated optical flow, which are clustered to resemble a specific object with individual dynamics. Subsequently, the trajectories of these clusters are predicted and checked for collision with the approximated trajectory of the robot. Consequently, the agent chooses the actions, that are most likely to avoid collisions, while moving towards a specific goal. Note that this approach is fully analytical and does not require any training. Nevertheless, manual tuning of hyperparameters may be required and the performance highly depends on the quality of the calculated optical flow. Furthermore, the algorithm only produces a reactive control policy and does not explicitly plan into the future.

Finally, Lobos-Tsunekawa *et al.*[19] investigate visual navigation on a biped platform, similarly to this thesis. The authors utilize DDPG, which is applied directly on low-resolution segmented images. Since it requires only the semantic segmentation of the images, simulation and real-world input have high similarity, which facilitates a sim-to-real transfer. This has been proven by successful application on the *NAO V5* platform without any re-training or parameter tuning[19].

Hafner *et al.* introduce the approaches *PlaNet* and *Dreamer*, which work very similar to the *World Models* method[3, 4]. Both models will be discussed in the course of the next chapter, as they form the foundation of the model used in this thesis.

# 3. Background

## 3.1. Model-Based Visual Control

In this chapter, we discuss two recent approaches by Hafner et al., who designed an agent that learns to solve different tasks with visual input.

Both methods are formulated as Reinforcement Learning (RL) approaches. Typically, an RL problem is modeled as a Markov Decision Process (MDP) and is denoted as a tuple $E = (S, A, P, R)$. This tuple consists of environment states $S$, an action space $A$, state transition probabilities $P$ and a reward function $R$. The environment state at time $t$ can be sensed through the observation $o_t$. Additionally, the agent receives the reward $r_t$ according to the reward function $R$. In order to alter the environment state, the agent can take an action $a_t \in A$, which determines the state according to the state evolution $P$. Note that the underlying MDP is unknown to the agent. Consequently, the agent needs to learn how the rewards are generated in order to produce actions that seek to maximize the expected reward.

The first approach, *PlaNet*[3], utilizes a *Variational Autoencoder* (VAE) $\mathcal{E}$ to compress the input images $o_t$ and a *Recurrent Neural Network* (RNN) $\mathcal{D}$ to build a dynamics model on the latent state $s_t$, similar to *World Models*[14]. In contrast to the work by Ha & Schmidhuber[14], Hafner *et al.*[3] not only consider a stochastic state representation $\hat{s}_t$, but also allow for deterministic paths $\hat{h}_t$ in the latent state space. This results in the *Recurrent State Space Model* (RSSM), which significantly improves performance and is one of the main contributions of the PlaNet model[3]. For sake of simplicity we will always refer to the tuple $(\hat{s}_t, \hat{h}_t)$ as $s_t$, since $\hat{h}_t$ can be described as a random variable with zero variance.

A schematic of the PlaNet model is depicted in Fig. 3.1. Generally, the model utilizes a VAE $\mathcal{E}$ to encode the input image $o_t$ to the latent state representation $s_t$ at time-step $t$. The VAE can also be used to decode $s_t$ to an estimated image $\tilde{o}_t$ during training. Based on the latent state, the reward $r_t$ can be predicted by use of the reward model $\mathcal{R}$. Additionally, the latent state $s_t$ can be propagated to the next time-step by using the latent state dynamics $\mathcal{D}$, which is modelled as a Gated Recurrent Unit (GRU)[3].

Using this dynamics model $\mathcal{D}$, it is possible to predict subsequent states $s_{t+1}$ to $s_{t+T}$ from a single starting state $s_t$ and thus the model can be considered as a

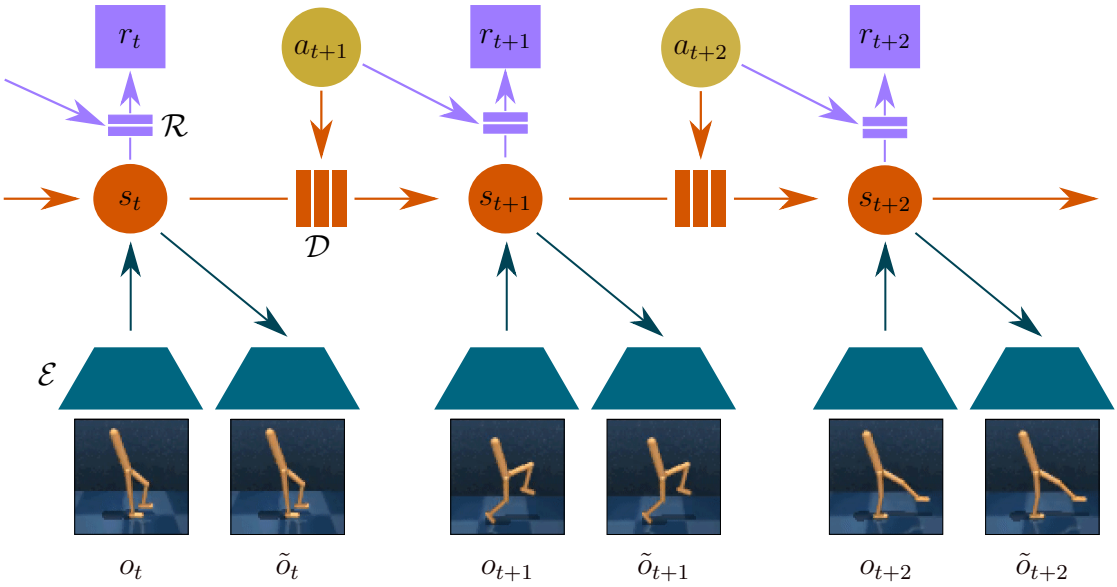


Figure 3.1: A schematic of the PlaNet model. At each time step $t$, the model utilizes a VAE $\mathcal{E}$ to encode an image $o_t$ into the latent state $s_t$. This latent representation is used to predict the reward $r_t$ using the reward model $\mathcal{R}$ and can be decoded to predict the corresponding image observation $\tilde{o}_t$ during training. Additionally, given an action $a_{t+1}$, which is selected using MPC, and a previous state $s_t$, the dynamics model $\mathcal{D}$ can predict the next latent state $s_{t+1}$. This way, it is possible to use the model as a simulator that reproduces all necessary information of the environment.

simulator resembling the evolution of core variables for the given task. Altogether, given an action $a_t$, the model can be described as

$$\tilde{s}_{t+1} \sim \mathcal{D}(\tilde{s}_{t+1}|s_t, \quad a_{t+1}) \tag{3.1}$$

$$s_{t+1} \sim \mathcal{E}(s_{t+1}\,|\tilde{s}_{t+1}, o_{t+1}) \tag{3.2}$$

$$r_t \quad \sim \mathcal{R}(r_t \quad |s_t, \quad a_t)\,. \tag{3.3}$$

Note that the prediction in Eq. (3.1) yields a state prior $\tilde{s}_{t+1}$ , which is then filtered to a state posterior $s_{t+1}$ in Eq. (3.2). Thus, the dynamics model can be seen as a non-linear Kalman filter in conjunction with the observation model [3]. Note also, that it is possible to recursively apply Eq. (3.1) to produce prior trajectories without filtering with images, enabling for fast generation of trajectories if $s_t$ is of sufficiently low dimension. Furthermore, the state priors $\tilde{s}_t$ can be used to predict the corresponding reward estimates $\tilde{r}_t$. This way, it is possible to employ traditional planning algorithms like Model Predictive Control (MPC), which samples trajectories $(s_t \dots s_{t+H})$ to optimize the accumulated reward $\sum_{t=t_0}^{t_0+H} r_t$ for a given planning horizon $H$.

To account for model errors and environment noise, the network predictions are modeled stochastically. Generally, all predictions are given as Gaussian distributions, where the mean is parameterized by the corresponding prediction networks. In addition, the standard deviation of the latent state $s_t$ and the action $a_t$ is also given by the model, whereas the reward prediction always has unit variance[3].

In contrast to *PlaNet*, its successor *Dreamer* does not rely on additional planning algorithms such as MPC, but employs an Actor-Critic model to learn the actions directly. The corresponding schematic of the Dreamer model can be seen in Fig. 3.2. Similar to *PlaNet*, the latent state dynamics is given by $\mathcal{D}$ and the reward is predicted by the network $\mathcal{R}$. Additionally, the value network $\mathcal{V}$ is trained on the predicted return calculated via $\mathcal{R}$. More precisely, it approximates the Bellman return

$$v_{t_0 \dots T} = \sum_{t=t_0}^{t_0+T} \gamma^{(t-t_0)} r_t \tag{3.4}$$

for a discount factor $\gamma < 1$. This way, $v_{t_0 \dots T}$ efficiently summarizes the return, i.e. the accumulated reward in the time interval from $t_0$ to $T$, while utilizing $\gamma$ to weigh more recent rewards higher than rewards in far future. Furthermore, the model can utilize the reward model $\mathcal{R}$ to predict rewards from latent states similar to Eq. (3.4)

$$\tilde{v}_{t_0 \dots T} = \sum_{t=t_0}^{t_0+T} \gamma^{(t-t_0)} \mathcal{R}(s_t)\,, \tag{3.5}$$

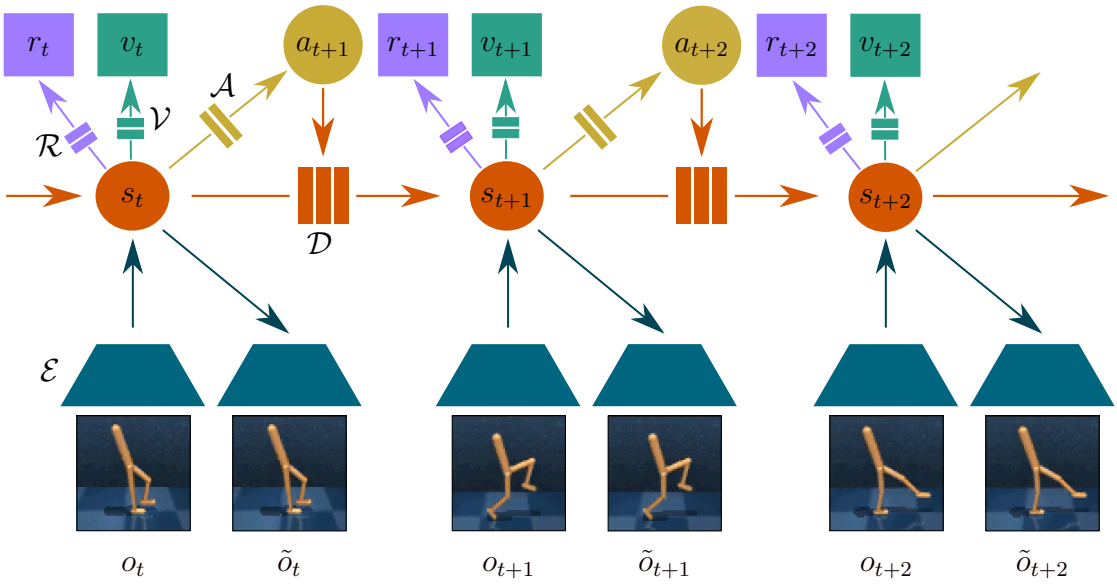


Figure 3.2: A schematic of the Dreamer model. Similar to PlaNet, Dreamer utilizes a VAE $\mathcal{E}$ to generate a latent state $s_t$ from the observations $o_t$, which can be used to predict the reward $r_t$ using $\mathcal{R}$. In contrast to PlaNet, this model introduces a value model $\mathcal{V}$ to predict the state value $v_t$. This value estimate can be used to train the actor $\mathcal{A}$, which directly generates the next action $a_{t+1}$. Thus, no additional planning algorithm (e.g. MPC) is required.

and thus does not require an actual observation, which enables for an extrapolated return $\tilde{v}$. Consequently, we can define the value model as

$$\tilde{v}_{t_0 \ldots T} \sim \mathcal{V}\left(\tilde{v}_{t_0 \ldots T} | s_t\right) . \tag{3.6}$$

The value network $\mathcal{V}$ is trained using the predictions from $\mathcal{R}$ and without any actually observed reward. However, it is important to underline that $\mathcal{R}$ is fitted on the observed environment rewards and thus will reconstruct the reward $r_t$ as well as possible. More precisely, the value model $\mathcal{V}$ optimizes the negative log likelihood

$$L_{\mathcal{V}} = -\mathbb{E}\left(\sum_{t=t_0}^{T} \ln \mathcal{V}\left(\tilde{v}_{t \ldots T} | s_t\right)\right) . \tag{3.7}$$

With a trained value model $\mathcal{V}$, it is possible to define an actor network

$$a_t \sim \mathcal{A}\left(a_t | s_{t-1}\right) , \tag{3.8}$$

that maximizes the predicted state values. Thus, $\mathcal{A}$ will learn to choose actions that lead to states with high values, consequently yielding an overall high reward.

**Algorithm 1** Dreamer

Initialize $\mathcal{E}, \mathcal{D}, \mathcal{R}, \mathcal{V}, \mathcal{A}$
$E \leftarrow \{\}$

Generate environment scenes
$e_0 \leftarrow$ Collect initial episodes using a random agent
$E \leftarrow E \cup e_0$

**while** training **do**
    Draw random subset $\hat{E}$ from $E$
    Fit $\mathcal{E}, \mathcal{D}, \mathcal{R}$ on $\hat{E}$ (see loss in Eq. (3.14))

    $\hat{E}^+ \leftarrow$ Extrapolate each state in $\hat{E}$ using $\mathcal{A}$ and $\mathcal{D}$
    $\tilde{r}^+ \leftarrow$ Predict rewards in $\hat{E}^+$ using $\mathcal{R}$
    $v^+ \leftarrow$ Calculate returns from $\tilde{r}^+$ (see Eq. (3.5))
    $\tilde{v}^+ \leftarrow$ Predict value using $\mathcal{V}$

    Fit $\mathcal{V}$ on $v^+$ (see loss in Eq. (3.7))
    Maximize $\mathcal{A}$ with respect to $\tilde{v}^+$ (see loss in Eq. (3.9))

    Generate environment scenes
    $e \leftarrow$ Generate episodes from policy
    $E \leftarrow E \cup e$

This resembles the well-known *Actor-Critic* concept, where the actor $\mathcal{A}$ selects the actions, whereas its performance is graded by a critic $\mathcal{V}$. Consequently, we formulate the loss term as

$$L_{\mathcal{A}} = -\mathbb{E}\left(\sum_{t=t_0}^{T} \mathcal{V}\left(\tilde{v}_{t\ldots T} | s_t\right)\right). \tag{3.9}$$

Observe that the actor loss $L_{\mathcal{A}}$ depends on the performance of $\mathcal{V}$, whereas the value loss $L_{\mathcal{V}}$ depends on the performance of $\mathcal{A}$, since it is trained on trajectories that are generated using the trained actor model. This can be particularly challenging for training the networks, since the different models can hinder each others optimization.

The full training procedure is summarized in Alg. 1. Note that $\mathcal{V}$ and $\mathcal{A}$ are solely trained on an extrapolated set of states. These states begin with sampled recorded states $\hat{E}$, which are used to produce a set of trajectories $\hat{E}^+$ through recursive open-loop application of $\mathcal{A}$ and $\mathcal{D}$(see Eq. (3.8), Eq. (3.1)) - a process, which Hafner *et al.*[4] refer to as "imagination"[4]. Also note that the extrapolated

$\hat{E}^+$ include significantly more states than the original $\hat{E}$, since each state in $\hat{E}$ is the starting point of a whole trajectory in $\hat{E}^+$.

To limit the amount of information that is transferred to the latent state, the approach utilizes the *information bottleneck objective* by Tishby *et al.*[20]. Intuitively, this objective ensures, that enough information is able to pass to the latent state, while discouraging the use of extra information. In general, the information bottleneck objective

$$L\left[p\left(\tilde{x}|x\right)\right] = I(\tilde{X};X) - \beta I(\tilde{X};Y) \tag{3.10}$$

is based on the *mutual information* $I$ of $\tilde{X}$, $X$ and $Y$. $\tilde{X}$ is a compressed representation of $X$, whereas $Y$ is a target distribution which $\tilde{X}$ is supposed to capture as well as possible[20]. This objective formulates a trade-off between keeping state information $X$ and reconstructing target information $Y$. Note that this general objective can be applied to our application by setting $X$ to the observation model, $\tilde{X}$ to the compressed latent state $s_t$ and $Y$ to the true data distribution. Consequently, the loss of the prediction model can be formulated as

$$L\left[p\left(o_{1:T}, r_{1:T}|s_{1:T}, a_{1:T}\right)\right] = I\left(s_{1:T};\left(o_{1:T}, r_{1:T}\right)|a_{1:T}\right) - \beta I\left(s_{1:T};i_{1:T}|a_{1:T}\right) \ , \tag{3.11}$$

for a scaling factor $\beta > 0$ and the auxiliary dataset indices $i_{1:T}$, defining the observations through a one-hot encoding, analogously to the approach by Alemi *et al.*[21]. Altogether, using the optimal probability distribution $p$ under the recorded data, we can derive the loss for the prediction model as follows:

$$\begin{aligned}
& I\left(s_{1:T};\left(o_{1:T}, r_{1:T}\right)|a_{1:T}\right) \\
= \ & \mathbb{E}_{p(o_{1:T}, r_{1:T}, s_{1:T}, a_{1:T})}\left[\sum_t \ln p\left(o_{1:T}, r_{1:T}|s_{1:T}, a_{1:T}\right) - \ln p\left(o_{1:T}, r_{1:T}|a_{1:T}\right)\right] \\
\overset{\text{const}}{=} \ & \mathbb{E}\left[\sum_t \ln p\left(o_{1:T}, r_{1:T}|s_{1:T}, a_{1:T}\right)\right] \\
\geq \ & \mathbb{E}\left[\sum_t \ln p\left(o_{1:T}, r_{1:T}|s_{1:T}, a_{1:T}\right)\right] \\
& - \mathrm{KL}\left[p\left(o_{1:T}, r_{1:T}|s_{1:T}, a_{1:T}\right) || \prod_t \mathcal{E}\left(o_t|s_t\right)\mathcal{R}\left(r_t|s_t\right)\right] \\
= \ & \mathbb{E}\left[\sum_t \ \ln \mathcal{E}\left(o_t|s_t\right) + \ln \mathcal{R}\left(r_t|s_t\right)\right]
\end{aligned} \tag{3.12}$$

$$
\begin{aligned}
& I\left(s_{1:T}; i_{1:T} | a_{1:T}\right) \\
= \; & \mathbb{E}_{p(o_{1:T}, r_{1:T}, s_{1:T}, a_{1:T}, i_{1:T})}\left[\sum_t \ln p\left(s_t | s_{t-1}, a_{t-1}, i_t\right) - \ln p\left(s_t | s_{t-1}, a_{t-1}\right)\right] \\
= \; & \mathbb{E}\left[\sum_t \ln p\left(s_t | s_{t-1}, a_{t-1}, o_t\right) - \ln p\left(s_t | s_{t-1}, a_{t-1}\right)\right] \\
= \; & \mathbb{E}\left[\sum_t \ln \mathcal{E}\left(s_t | \mathcal{D}\left(\tilde{s}_t | s_{t-1}, a_{t-1}\right), o_t\right) - \ln \mathcal{D}\left(s_t | s_{t-1}, a_{t-1}\right)\right] \\
= \; & \mathbb{E}\left[\sum_t \mathrm{KL}\left[\mathcal{E}\left(s_t | \mathcal{D}\left(\tilde{s}_t | s_{t-1}, a_{t-1}\right), o_t\right) || \mathcal{D}\left(s_t | s_{t-1}, a_{t-1}\right)\right]\right]
\end{aligned}
\tag{3.13}
$$

As a consequence, we can conclude with the information bottleneck objective

$$
\begin{aligned}
& I\left(s_{1:T}; (o_{1:T}, r_{1:T})\right) - \beta I\left(s_{1:T}; i_{1:T} | a_{1:T}\right) \\
\geq \; & \mathbb{E}\left[\sum_t \ln \mathcal{E}\left(o_t | s_t\right) + \ln \mathcal{R}\left(r_t | s_t\right)\right] \\
& \quad - \beta\, \mathbb{E}\left[\sum_t \mathrm{KL}\left[\mathcal{E}\left(s_t | \mathcal{D}\left(\tilde{s}_t | s_{t-1}, a_{t-1}\right), o_t\right) || \mathcal{D}\left(s_t | s_{t-1}, a_{t-1}\right)\right]\right] \\
= \; & \mathbb{E}\Bigg[\sum_t \ln \mathcal{E}\left(o_t | s_t\right) + \ln \mathcal{R}\left(r_t | s_t\right) \\
& \qquad\qquad - \beta\, \mathrm{KL}\left[\mathcal{E}\left(s_t | \mathcal{D}\left(\tilde{s}_t | s_{t-1}, a_{t-1}\right), o_t\right) || \mathcal{D}\left(s_t | s_{t-1}, a_{t-1}\right)\right]\Bigg] \\
= \; & - L_{\mathcal{E},\mathcal{R},\mathcal{D}}\,.
\end{aligned}
\tag{3.14}
$$

This loss term formulates a joint objective of the encoder $\mathcal{E}$ and the reward model $\mathcal{R}$, which are trained using the negative log likelihood. Furthermore, it utilizes the *Kullback-Leibler divergence*, which can be seen as a distance measure between two probability distributions. Moreover, since $p\left(s_t | s_{t-1}, a_{t-1}, o_t\right)$ resembles the posterior state, which has been filtered on the observation $o_t$, the Kullback-Leibler divergence term ensures, that the dynamics model $\mathcal{D}$ mimics the filtering process of the encoder $\mathcal{E}$ without having access to the actual observation $o_t$. Thus, the last term enables the model to produce open-loop predictions that resemble the original data distribution as well as possible.

# 4. Approach

## 4.1. Problem Formulation

Whenever an agent has to navigate a scene with obstacles, path planning is necessary. For instance, if a bipedal soccer robot should reach the ball on a field with opponents, it is required to plan a trajectory around those opponents to avoid a collision. Similarly, our task is based on the NimbRo-OP2X [7] robot, which is supposed to reach a target position while avoiding obstacles in the scene. The data is generated using the simulator *Multi-Joint dynamics with Contact* (MuJoCo)[1], but we also anticipate a real-world application.

The simulation utilizes the NimbRo-ROS framework[7], which includes a reliable bipedal gait engine. Additionally, the computer-vision pipeline is used to retrieve the ball position in the world application, resembling the target.

However, the use of the NimbRo-ROS framework required the implementation of a bridge between MuJoCo and ROS. Features such as offsceen-rendering, dynamic object placement and semantic segmentation are implemented in this bridge. Furthermore, a gym environment[22] is created. It communicates with the MuJoCo-ROS bridge by sending information such as sampled obstacle poses, and retrieving data such as the global pose of the robot. Note that the gym environment handles all sampling processes, whereas the MuJoCo-ROS bridge is only responsible for physics and rendering, which keeps the framework general.

Since the environment updates at discrete steps, it is required to set a step frequency that controls how many agent-environment interactions per second are allowed. This step frequency typically varies strongly between different tasks, as joint control tasks for instance may require frequencies over 100Hz, whereas our task will be evaluated with a frequency of 4Hz. As we are working with a gait, which generally reacts significantly slower to actions than the joints themselves, the comparably low frequency is justified. Furthermore, reducing the frequency results in covering the same amount of time in less steps, which will lead to a smaller amount of planning steps.

[1] http://www.mujoco.org/

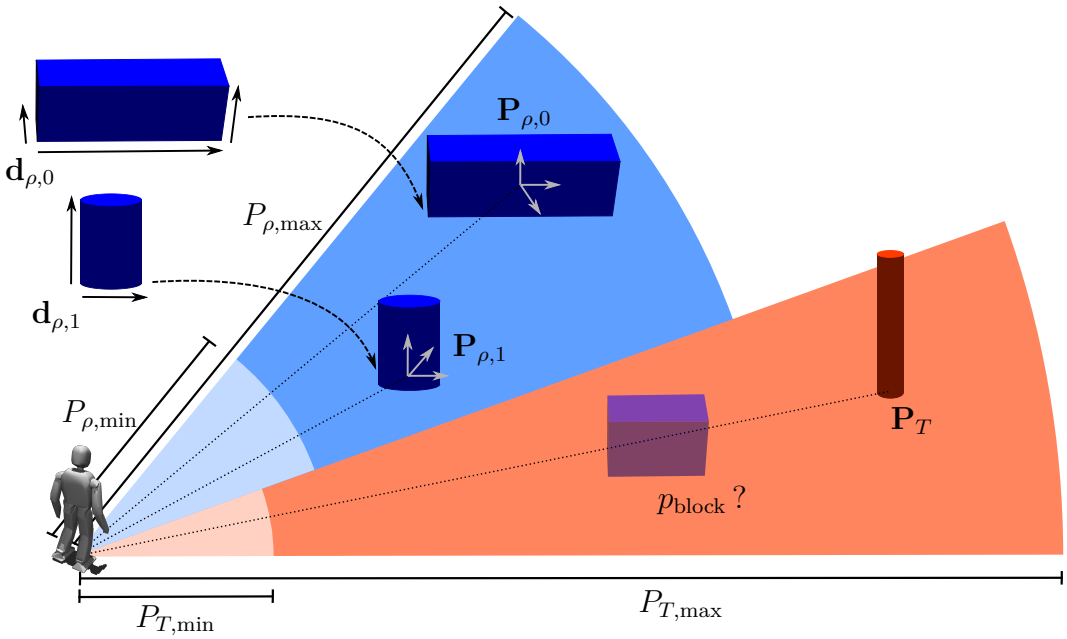


Figure 4.1: The environment sampling mechanism with two obstacles. First, the target position $\mathbf{P_T}$ is sampled and the direct path of the robot to the target is blocked with probability $p_{\text{block}}$. Then, a number $n_\rho$ of obstacles is drawn and corresponding random sizes $d_{\rho,i}$ are assigned. Subsequently, the objects are placed in the scene with random pose $P_{\rho,i}$. The target, as well as the obstacles, respect the individual minimum and maximum allowed distances $P_{\text{min}}$, $P_{\text{max}}$ to the origin.

### Scene Generation

Since the diversity of the simulation governs the diversity of the training data, the constructed scenes should be as general as possible. We address this by generating each environment scene pseudo-randomly. An illustration of the sampling process is shown in Fig. 4.1.

The target location $\mathbf{P}_T \in \mathbb{R}^2$ is sampled uniformly at random from a disk of radius $P_{T,\text{max}}$ around the initial pose of the robot, while excluding an inner disk of radius $P_{T,\text{min}}$ to ensure a minimum distance to the starting point. The target is represented as a cylinder of fixed diameter $d_T$ perpendicular to floor plane. Furthermore, the target is of infinite height, thus only the x- and y-position are randomized.

Additionally, we sample a number of objects $n_\rho \in [0, n_{\rho,\text{max}}]$ that shall be placed in the simulation, as well as their pose $\mathbf{P}_{\rho,i} \in \mathbb{R}^6$ for $i \in \{0 \dots n_\rho\}$ uniformly at random. The corresponding object positions are sampled uniformly at random, similar to $\mathbf{P}_T$, where we allow objects to be inside a disk of radius $P_{\rho,\text{max}} \in \mathbb{R}$, with a

minimum distance $P_{\rho,\min} \in \mathbb{R}$ to the origin. Furthermore, we allow object shapes of cylinders and boxes, whose dimensions $\mathbf{d}_{\rho,i} \in [\mathbf{d}_{\rho,\min}, \mathbf{d}_{\rho,\max}] \subset \mathbb{R}^{2\times 3}$ are also sampled uniformly at random. Additionally, we introduce a blocking probability $p_{\text{block}}$, which controls an enforced blocking of the direct path between the robot and the target to increase the mean difficulty of the scenes. With probability $p_{\text{block}}$ an obstacle is placed in the direct path.

Finally, we allow for texture and color mask randomization of the objects, as well as of the ground floor. This can facilitate a real world transfer of models which are trained on RGB data, by increasing the visual complexity of the simulator to match the real world more closely.

Note that the robot can be placed at the origin of the simulation w.l.o.g., since its position and rotation relative to the *randomized* objects in the scene can be considered random.

### Action Space

A substantial part of the environment is the *action space*, which defines the way how the agent can influence the environment state. To control the robot movement on the ground plane, we specify the target gait velocities $\mathbf{V}_t \in \mathbb{R}^3$ at time $t \in \mathbb{R}^+$, which we will refer to as gait commanded velocity (GCV). This GCV

$$\mathbf{V}_t = [v_{t;x}, v_{t;y}, v_{t;\theta}] \tag{4.1}$$

can be split into two translational velocities $v_{t;x}, v_{t;y} \in \mathbb{R}$ and one rotational velocity $v_{t;\theta} \in \mathbb{R}$ around the Z axis of the robot (see Fig. 4.2d).

These velocities are used by a gait engine to produce dynamic leg motions, which originate from a central pattern generator. Altogether, this results in a dynamically stable gait, which is able to recover from small disturbances without any actions[7, 23]. Additionally, the gait incorporates reactive actions, which enables closed-loop control of the limbs, correcting the estimated trunk orientation of the robot to match the nominal rotations. These corrective actions include movement of the arms and tilting of the feet[23].

However, to learn the direct control of the absolute target gait velocity has several difficulties for an agent. More precisely, it might result in oscillating action outputs which will lead to a fall of the robot. To address this issue, we propose

incremental control actions:

$$\Delta \mathbf{V}_t = [\Delta v_{t;x}, \Delta v_{t;y}, \Delta v_{t;\theta}] \tag{4.2}$$

$$\mathbf{V}_t = \mathbf{V}_{t-1} + \Delta \mathbf{V}_{t-1} \tag{4.3}$$

$$\mathbf{V}_0 = [0, 0, 0] \ . \tag{4.4}$$

In this manner, we conclude with an action space representation which is generally less vulnerable to unstable inferred agent outputs.

### Observations

While navigating through the environment, the agent relies on the observations it receives. Since cameras are cheap and available in small form factors, they are a often used on mobile platforms. Consequently, we propose to utilize image observations $o_t$, which consist of low-resolution RGB images ($64 \times 64$ in our experiments) and are typically the primary source of information. These images are taken from the ego perspective of the robot, more precisely from a camera at the right eye. Note that the image observations are processed by a segmentation algorithm $\zeta$ before they are used by the agent (see Fig. 4.2a).

In contrast to traditional visual-control environments, we do not limit the observation space to only visual input, but also provide a non-visual observation source called *direct observation*. These direct observations are formulated as a stacked vector

$$z_t = \left[\mathbf{V}_t, \mathbf{G}_t, \mathbf{R}_{t;(R,P)}\right] \ , \tag{4.5}$$

containing gait velocities $\mathbf{V}_t \in \mathbb{R}^3$ (see Fig. 4.2b), relative target position $\mathbf{G}_t \in \mathbb{R}^2$ (see Fig. 4.2c) and robot roll and pitch rotation $\mathbf{R}_{t;(R,P)} \in \mathbb{R}^2$ (see Fig. 4.2b). To make the learning process easier, the relative target position

$$\mathbf{G}_t = [d_t, \phi_t] = [||\mathbf{P}_t - \mathbf{x}_t||_2, \arctan2(\mathbf{P}_t - \mathbf{x}_t) - \theta_{t;\mathrm{Yaw}}] \tag{4.6}$$

is given in 2-dimensional polar coordinates using the target position $\mathbf{P}_t \in \mathbb{R}^2$, the current 2-D robot position $\mathbf{x}_t \in \mathbb{R}^2$ and the robot yaw rotation $\theta_{t;\mathrm{Yaw}} \in \mathbb{R}$. Seeing that the robot and the target are located on the same plane, we only consider the polar coordinate representation in 2-D. Encoding the target information in polar coordinates makes the learning process for the agent more intuitive, as we will expect the robot to turn towards the target and keep a forward motion for most of the episode. More precisely, $\phi_t$ will directly indicate whether the robot has turned towards the target, whereas the forward motion depends on the distance encoded in $d_t$.

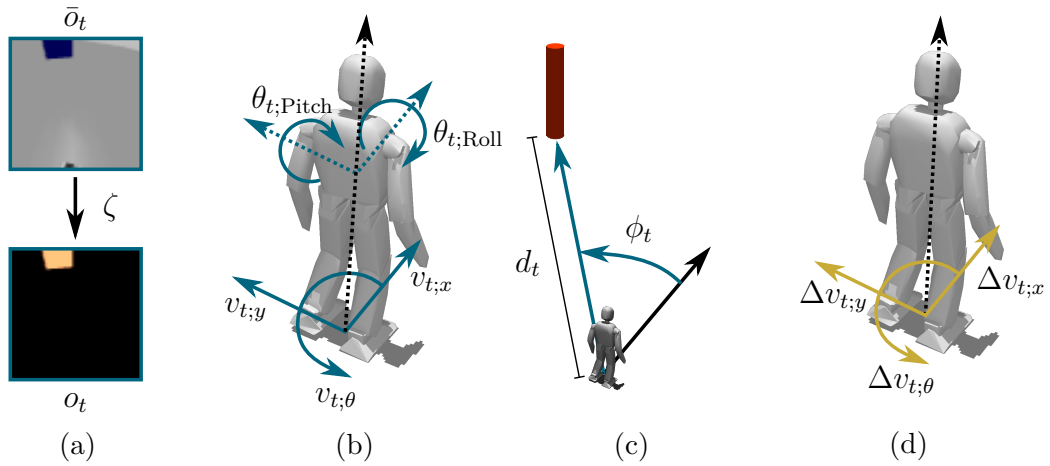


Figure 4.2: A visualization of the observation and action space. In (a), the image segmentation $\zeta$ is shown, transforming a captured RGB image $\bar{o}_t$ into a grayscale image, where only the obstacles are visible. The individual components of direct observation $z_t$ are displayed in (b) and (c). It consists of the robot center of mass rotations $R_{t;P}$ and $R_{t;R}$, the current gait velocities $v_{t;x}$, $v_{t;y}$, $v_{t;\theta}$ and the relative target position in polar coordinates $(d_t, \phi_t)$. In (d), the action-space is shown, displaying the target gait velocity increments $\Delta v_{t;x}$, $\Delta v_{t;y}$, $\Delta v_{t;\theta}$.

The last part of the direct observation incorporates the Euler rotations

$$\mathbf{R}_{t;(R,P)} = [\theta_{t;\text{Roll}}, \theta_{t;\text{Pitch}}] \tag{4.7}$$

around the respective $x$- and $y$-axis of the robot. Including the orientation enables the bipedal robot to anticipate falling by monitoring the pitch and roll. Note that we explicitly do not incorporate the yaw angle $\theta_{t;\text{Yaw}}$ into the direct observation, as it can be harder to track in the real world, complicating the real-world transfer. More precisely, the robot is expected to have an upright position during walking, resulting in zero-mean gyroscope measurements for the roll and pitch axes. In contrast to this, the yaw orientation can change significantly during walking, which results in the accumulation of errors and ultimately facilitates in a strong drift in the rotation estimate.

Furthermore, observe that the strict exclusion of $\theta_{t;\text{Yaw}}$ requires the measurability of $\mathbf{G}_t = [d_t, \phi_t]$, as the calculation of $\phi_t$ in Eq. (4.6) would involve $\theta_{t;\text{Yaw}}$. However, this is no issue in most applications as computer vision systems typically output detections in the robot coordinate frame, rendering the global yaw orientation $\theta_{t;\text{Yaw}}$ unnecessary for the relative target calculation.

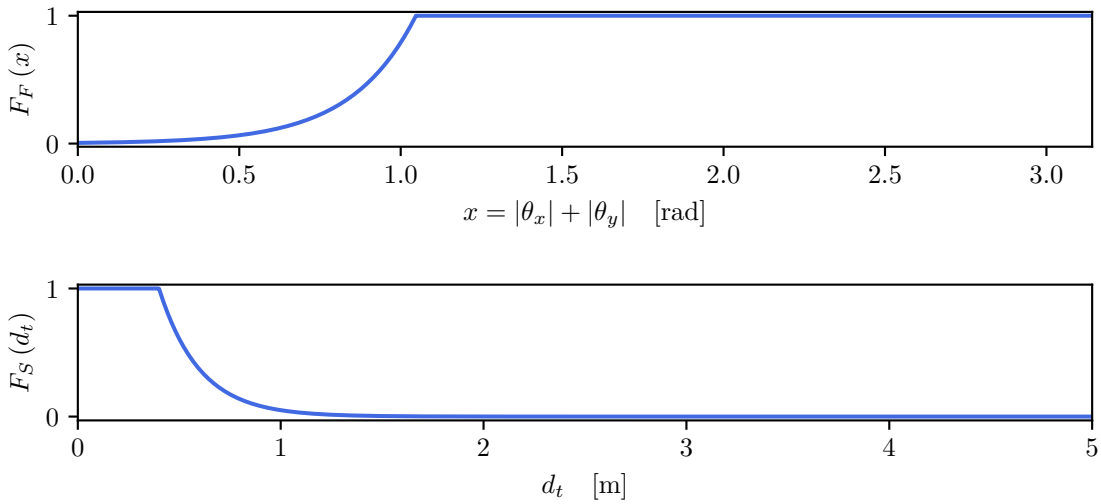


Figure 4.3: Termination signals. To enable for a better prediction of the termination states, a termination likelihood signal is generated. The vertical axes display the modeled probability of termination relative to the termination criteria on the corresponding horizontal axes. Both curves resemble an exponential decay, which is clipped at the value of 1 to match the notion of a probability. As soon as any $F_i = 1$, the environment terminates the episode.

### Termination Criteria

Many real-world problems include a notion of termination, for instance through success or failure. We consider an episodic task, where an episode can be terminated in three different ways. The first option terminates the episode if the robot falls, i.e. the trunk rotations exceed the limits

$$|\theta_{t;\text{Roll}}| + |\theta_{t;\text{Pitch}}| > \frac{\pi}{3}\,, \tag{4.8}$$

resulting in the termination signal $F_F = 1$. This limit corresponds to a rotational orientation from which the robot cannot recover by itself. Consequently, if the limit is exceeded, the robot will inevitably fall. Since the robot can not proceed further towards the target after falling, data efficiency justifies the use of this criterion.

Additionally, successful termination $F_S = 1$ is achieved when the robot position is close enough to the target, such that

$$d_t < 0.4\,\text{m}\,. \tag{4.9}$$

Finally, a timer of $T = 60$ seconds will lead to a timeout if no other terminal

states are reached, leading to $F_T = 1$.

Traditionally, the termination signals are set $F_i = 0$ for all remaining steps. However, to enable the use of continuous termination signals, we propose the smooth termination likelihood

$$F_F\left(\theta_{t;\text{Roll}}, \theta_{t;\text{Pitch}}\right) = \text{clip}\left[\exp^{\nu_F\left(|\theta_{t;\text{Roll}}|+|\theta_{t;\text{Pitch}}|-\frac{\pi}{3}\right)}; 0, 1\right] \tag{4.10}$$

$$F_S\left(d_t\right) = \text{clip}\left[\exp^{\nu_S(d_t-0.4)}; 0, 1\right] \tag{4.11}$$

for the failure and success termination states $F_F$, $F_S$, given the corresponding decay factors $\nu_F, \nu_S > 0$. A plot of the termination likelihoods can be seen in Fig. 4.3. Naturally, it is possible to incorporate an additional likelihood $F_T$, which handles the termination through timeout. In this implementation, modeling $F_T$ has been omitted, since the approximation of $F_T$ requires increased effort from the agent, harming other parts of the network. Additionally, if the current time $t$ is not an input to the model, the approximation of $F_T$ can not be inferred when training the model on sampled sequences of equal length $L$, since the first step $t_i$ of each sequence can be at an ambiguous step of the episodes. This makes the tracking of the time unfeasible, since the memory of the agent can be trained on a maximum of $L$ steps. However, if we consider time as an input to the agent, the approximation of $F_T$ will be a trivial task. Since the maximum length $T$ of the episodes may be exceeded in real-world applications, we propose to exclude the time $t$ from the state representation, allowing a more general application of the model.

Mathematically, these continuous termination signals mimic the parameterization of a Bernoulli distribution for each time step $t$. Moreover, the Bernoulli distribution captures the event of termination with probability $p_t$, whereas the episode continues with probability $1 - p_t$. Consequently, we parameterize the process by utilizing the termination signal $F_{i;t} =: p_t$.

Note that, for the considered task, the success and failure termination states serve mostly for the purpose of data-efficiency. Optimizing only the given reward structure can already be sufficient to solve the task. However, this would imply unnecessarily recorded steps at the target location or even of an incapacitated fallen robot. Therefore, we strongly suggest the use of terminal states. To balance for the potential lost reward in case of termination, we introduce two solutions: First, specific terminal subrewards $r_{S;t}$ and $r_{F;t}$, and second, the optimization of the model with respect to the new smooth termination likelihood $F_i$. The former will be introduced in the next paragraphs and the latter will be discussed in Chapter 4.2.

### Reward Structure

To assess its performance on the given task, the agent is supplied with a reward function. By maximizing this reward, the agent can update its policy to solve the task.

For our environment, the reward function is defined as a weighted sum of subrewards $r_{i;t} \in \mathbb{R}$ which are chosen from a finite index-set $I \subset \mathbb{N}$ and weighted using a set of $\lambda_i \in \mathbb{R}$.

$$r_t = \sum_{i \in I} \lambda_i r_{i;t} \tag{4.12}$$

Note that, by setting $\lambda_i = 0$, it is possible to disable a specific subreward with index $i \in I$.

First, we define the target subrewards

$$r_{TD;t} = 1 - \frac{d_t}{d_0} \in (-\infty, 1] \tag{4.13}$$

$$r_{TR;t} = \frac{\phi_t}{\pi} \qquad \in [-1, 1] \tag{4.14}$$

by normalizing the distance to the target at time $t$ by the distance at the start of the episode, and by normalizing the rotation. Intuitively, these are the subrewards which will encourage the agent to walk towards the target and thus can be seen as the most essential subrewards of the environment.

Additionally, we introduce a velocity regularization reward

$$r_{V;t} = 1 - \frac{1}{e^{-25(||\mathbf{V}_t||_2 - 0.65)}} \in [-1, 0]\,, \tag{4.15}$$

penalizing high target gait velocities $\mathbf{V}_t$ using a logistic function. In general, this subreward will enforce velocities to be below a threshold, ensuring that the chosen velocities of an agent will be safe to use on a real robot. An illustration of the GCV penalty can be seen in Fig. 4.4.

Furthermore, we want to utilize the obstacle positions $\mathbf{O}$ to penalize obstacle contacts using the subrewards

$$r_{OC;t} = \begin{cases} -1 & \text{if contact between robot and obstacle} \\ 0 & \text{otherwise} \end{cases} \in [-1, 0] \tag{4.16}$$

$$r_{OD;t} = -(1 - \min(\{\forall O_i \in \mathbf{O} : ||O_i - x_t||\} \cup \{1\})) \qquad \in [-1, 0]\,, \tag{4.17}$$

where $r_{OD;t}$ gives a linearly interpolated penalization for stepping closer than a predefined distance (1 m in our experiments) towards an obstacle, and $r_{OC;t}$ sparsely

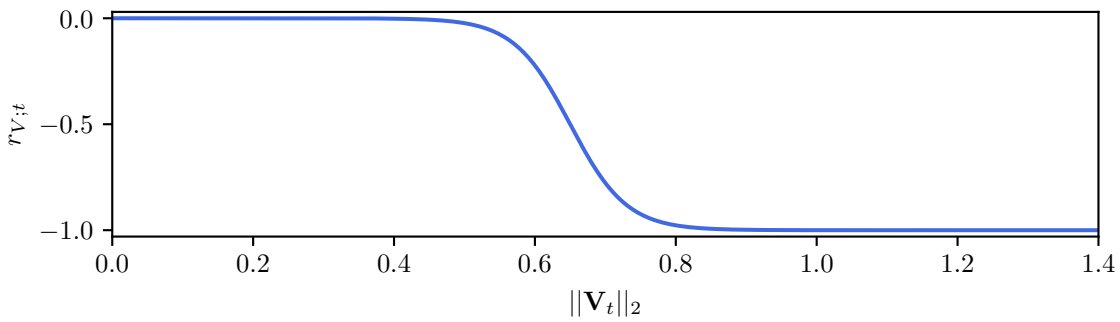


Figure 4.4: A visualization of the gait velocity penalization. When the norm of the GCV values above $||\mathbf{V}_t||_2 > 0.6$, the agent gets strongly penalized.

penalizes registered contacts of the robot body with an obstacle. Generally, these subrewards encourage the agent to avoid collisions and keep a safe distance to obstacles in the scene.

Finally, we consider success and failure rewards

$$r_{F;t} = \begin{cases} -(T-t) & \text{if robot has fallen} \\ 0 & \text{otherwise} \end{cases} \in [-T, 0] \tag{4.18}$$

$$r_{S;t} = \begin{cases} (T-t) & \text{if robot has reached the target} \\ 0 & \text{otherwise} \end{cases} \in [0, T]\,, \tag{4.19}$$

which will return either the positive or the negative number of remaining steps. This can be used to produce a subreward structure which can anticipate fast solutions and avoid early failure by setting

$$\lambda_S = \lambda_F = \sum_{i \in I \setminus \{S,F\}} \lambda_i\,. \tag{4.20}$$

Note that this subreward encapsules all future rewards that would be lost when terminating the episode at a specific time.

## 4.2. World Model

In this section, we will discuss the model for online path planning based on latent dynamics proposed in this thesis. The model utilizes *Dreamer* as a backbone, which has already been covered in the previous chapter. Our implementation

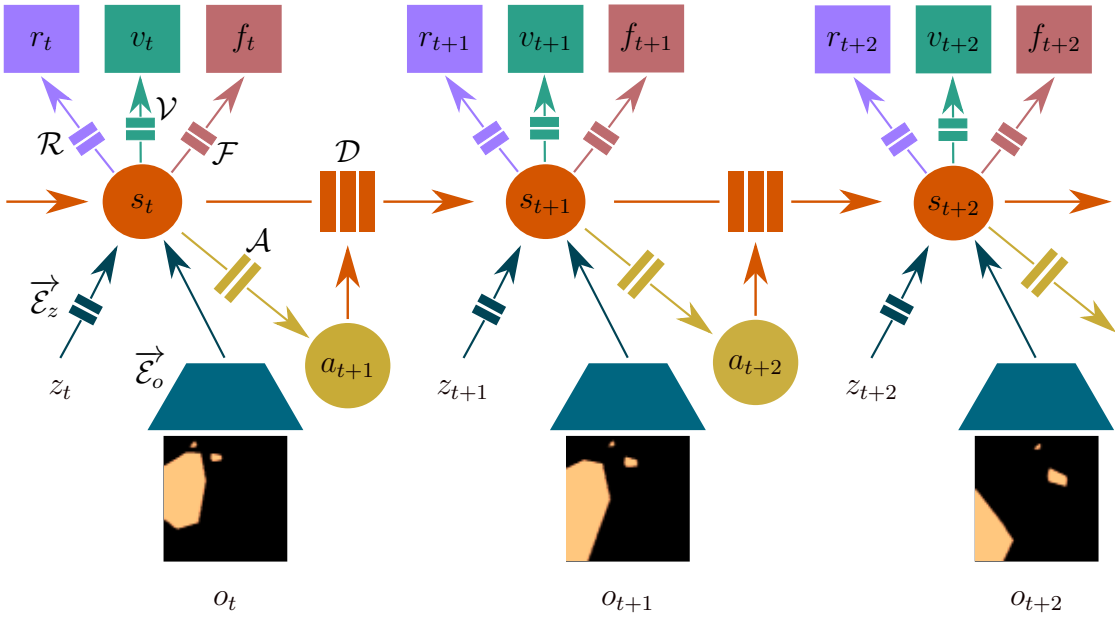


Figure 4.5: A schematic of our model. For sake of simplicity, the decoder parts $\overleftarrow{\mathcal{E}_o}$, $\overleftarrow{\mathcal{E}_z}$ of the input model have been omitted. On the bottom, the segmented image input $o_t$ and direct observation $z_t$ are shown, as well as the selected action $a_t$ for each step. Both inputs are encoded into the latent state $s_t$, which in turn can be used to predict the reward $r_t$, a state value $v_t$ and a set of termination probabilities $f_t$. Furthermore, using the learned dynamics model $\mathcal{D}$, it is possible to predict latent states into the future, given an action $a_t$.

builds upon the released version by Danijar Hafner[2], but the official Google release[3] has also been assessed[4].

An overview of our model can be seen in Fig. 4.5. Similarly to *Dreamer*, we utilize an observation model to construct a latent state $s_t$, which can be used to generate actions $a_t$, predict rewards $r_t$, values $v_t$ and the future step $s_{t+1}$. However, we also propose to predict the terminal likelihood $f_t$, which indicates the termination probability of the environment. Additionally, the model not only utilizes image input $o_t$, but also processes the *direct observation* $z_t$, which encodes concise target information, as well as sensor measurements.

### Observation Model

Intuitively, the model encodes the observations into a learned state representation, which is used for further computation. To construct this state space, an observation model capable of extracting all necessary information is required.

[2] https://github.com/danijar/dreamer/
[3] https://github.com/google-research/dreamer/

To reduce the visual complexity of the observed image, our approach applies a semantic segmentation on the input images similar to [19]. Thus, an observation $o_t$ can be described as

$$o_t = \zeta(\bar{o}_t)\,, \tag{4.21}$$

for a segmentation algorithm $\zeta$ and the original image observation $\bar{o}_t$. Note that $\zeta$ can be directly performed by a simulator, whereas a pre-trained segmentation model can be used in the real world. This not only makes the approach more general, as $\zeta$ can be changed to adapt the agent to different environments, but also facilitates an easier real-world transfer compared to using the raw images. The segmented images from the real world are significantly more similar to the segmented simulation images, while being substantially less complex compared to the original images, due to their lack of texture. This reduces the necessary model capacity of the Variational Autoencoder (VAE), since irrelevant data has already been removed through the segmentation and thus it does not need to be reconstructed. Consequently, the latent state $s_t$ does not need to contain the information that was already filtered through the segmentation, which results in a more stable and generally smaller model.

Moreover, most robot platforms are equipped with an array of sensors, which can supply the agent with rich information such as gyroscope data or force measurements. In addition, it can be advantageous to provide explicit internal states of the robot, such as target gait velocities. This way, it is possible to encourage the model to include this specific internal state into its world model, which can aid the learning process. Whereas traditional visual-control methods discard these additional inputs, we introduce a non-visual input $z_t$ called *direct observation*. This additional input space enriches the latent state, as the model will learn to fuse the different sensor data.

Since multiple inputs $o_t, z_t$ are considered, the input model has to be extended, resulting in two encoders

$$s_{t;[o_t]} \sim \overrightarrow{\mathcal{E}_o}(s_{t;[o_t]} | \tilde{s}_t, o_t) \tag{4.22}$$

$$s_{t;[o_t,z_t]} \sim \overrightarrow{\mathcal{E}_z}(s_{t;[o_t,z_t]} | s_{t;[o_t]}, z_t)\,, \tag{4.23}$$

where the second subscript of $s_t$ indicates which information source the posterior has been filtered on. Furthermore, we denote the corresponding decoder part of each VAE as

$$\tilde{o}_t \sim \overleftarrow{\mathcal{E}_o}(\tilde{o}_t | s_{t;[o_t,z_t]}) \tag{4.24}$$

$$\tilde{z}_t \sim \overleftarrow{\mathcal{E}_z}(\tilde{z}_t | s_{t;[o_t,z_t]})\,, \tag{4.25}$$

for image predictions $\tilde{o}_t$ and direct observation prediction $\tilde{z}_t$. In addition, we define the reward, value and actor model as

$$\tilde{r}_t \sim \mathcal{R}(\tilde{r}_t|s_{t;[o_t,z_t]}) \tag{4.26}$$
$$\tilde{v}_t \sim \mathcal{V}(\tilde{v}_t\,|s_{t;[o_t,z_t]}) \tag{4.27}$$
$$\tilde{a}_t \sim \mathcal{A}(\tilde{a}_t|s_{t;[o_t,z_t]})\,, \tag{4.28}$$

In contrast to previous approaches, we do not only filter the data once, but we build the full state posterior over two separate filtering steps. Since $z_t$ does not consist of image data, we utilize a shallow dense encoder $\overrightarrow{\mathcal{E}_z}$ instead of the convolutional encoder $\overrightarrow{\mathcal{E}_o}$. Furthermore, if $z_t$ is of sufficiently low dimension, it can be passed directly to the latent state $s_t$, omitting the encoder $\overrightarrow{\mathcal{E}_z}$ completely. An overview of the convolutional VAE $\mathcal{E}_o$ can be seen in Fig. 4.6, which essentially mimics the VAE of the *Dreamer* approach. Note that, since our approach utilizes semantic segmentation with a single label, the corresponding input image $o_t$ only consists of a single grayscale channel. In our task, the corresponding label resembles the obstacles in the scene. Still, our model technically supports any number of labels, allowing the application on potentially more complex tasks.

Generally, the VAE utilizes convolutional layers with kernel size $4 \times 4$ to encode the input image to a feature vector of length 1024. The state vector, on the other hand, can be decoded into the shape of $o_t$ using convolution transpose layers of size $5 \times 5$ and $6 \times 6$. This enables the model to utilize the information from the image input, which can be enforced by applying a loss between $\tilde{o}_t$ and $o_t$. More precisely, we formulate the loss similarly to Eq. (3.14) by defining

$$\begin{aligned}
& -L_{\overleftarrow{\mathcal{E}_o},\overleftarrow{\mathcal{E}_z},\mathcal{R},\mathcal{D}} \\
= \;& I\left(s_{1:T};(o_{1:T},z_{1:T},r_{1:T})\right) - \beta I\left(s_{1:T};i_{1:T}|a_{1:T}\right) \\
= \;& \mathbb{E}\left[\sum_t \ln\overleftarrow{\mathcal{E}_o}\left(o_t|s_t\right) + \ln\overleftarrow{\mathcal{E}_z}\left(z_t|s_t\right) + \ln\mathcal{R}\left(r_t|s_t\right)\right. \\
& \left. - \beta\,\mathrm{KL}\left[\overrightarrow{\mathcal{E}_z}\left(s_{t;[o_t,z_t]}\middle|\overrightarrow{\mathcal{E}_o}\left(s_{t;[o_t]}\middle|\mathcal{D}\left(\tilde{s}_t|s_{t-1},a_{t-1}\right),o_t\right),z_t\right)\middle|\middle|\mathcal{D}\left(s_t|s_{t-1},a_{t-1}\right)\right]\right]\,,
\end{aligned} \tag{4.29}$$

which can be derived analogously to Eq. (3.12) and Eq. (3.13) by extending the observation space by the direct observation $z_t$. The Kullback-Leibler divergence regularizes the difference between the filtered posterior state and the unfiltered prior state distribution to ensure a proper dynamics model.

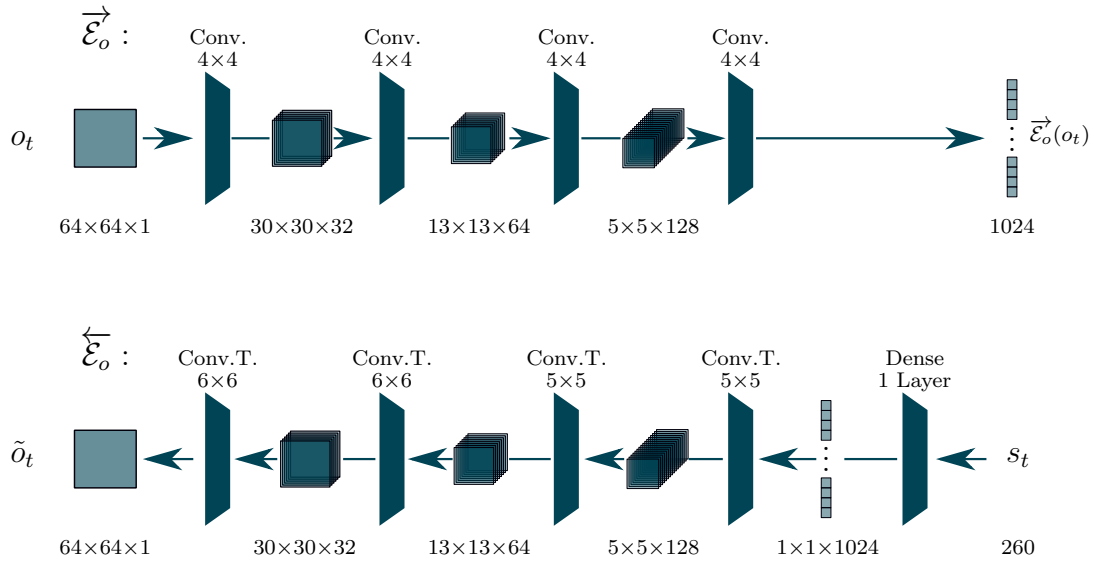


Figure 4.6: The network architecture of the variational autoencoder $\mathcal{E}_o$. On the top, the encoder $\overrightarrow{\mathcal{E}_o}$ is displayed, which utilizes convolutional layers with kernel size $4 \times 4$ to encode the image $o_t$ to a feature vector. The lower architecture displays the decoder $\overleftarrow{\mathcal{E}_o}$, which passes the state feature vector $s_t$ through multiple convolution-transpose layers with kernel sizes $5 \times 5$ and $6 \times 6$, to produce $\tilde{o}_t$ with the original image shape. Note that the higher dimension of the encoder output is reduced by the RNN to produce the lower dimensional $s_t$.

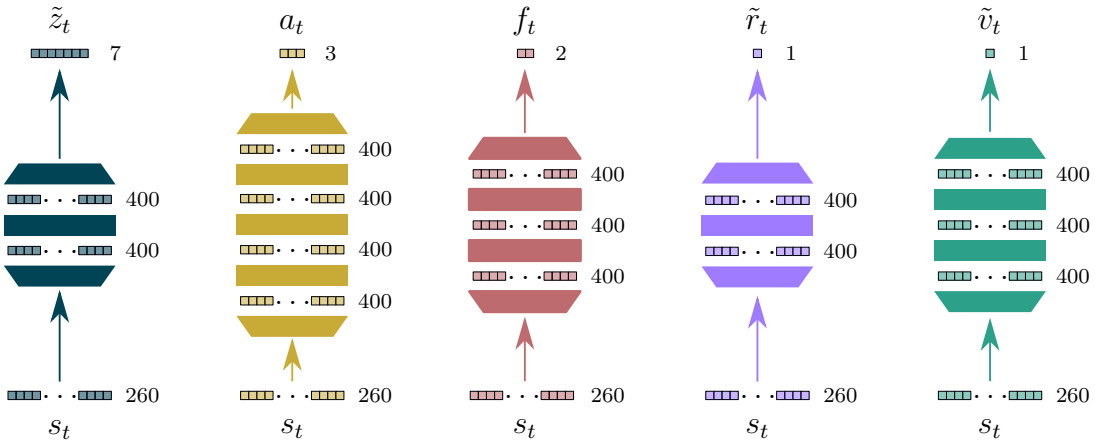


Figure 4.7: Prediction network architectures. Each network has the same number of hidden units $h_i = 400$ per layer, whereas the number of layers may vary between the networks. Based on the latent state representation $s_t$, predictions for $z_t$, $a_t$, $f_t$, $r_t$ and $v_t$ can be made using the presented models.

### Termination Model

Many applications are episodic, i.e. there exists a criterion to terminate the task upon success or failure. Examples for an autonomous vehicle may include the arrival at a destination for a successful termination and a collision with an object or person as a failed terminal state. Additionally, there can be supplementary terminal states, such as timeouts, which terminate the episode after a specific amount of time. The meaning to these terminal states are typically given through terminal rewards that are handed out as soon as the corresponding terminal state is reached. However, these terminal rewards can be hard to predict, as they only occur at a single state of the episode and thus lead to a discontinuous reward structure. To address this issue, we propose a dense network $\mathcal{F}$, which approximates the probabilities $f_t$ of reaching the separate terminal states, i.e.

$$f_t \sim \mathcal{F}\left(f_t | s_t\right) . \tag{4.30}$$

An overview over all prediction models is shown in Fig. 4.7.

The termination network models each individual $f_{t;i}$ as beta ($\beta$) distributed. The $\beta$-distribution is defined by its probability density function

$$\begin{aligned} P_{\alpha,\beta} : [0,1] &\to \mathbb{R}^+ \\ x &\mapsto x^{\alpha-1}(1-x)^{\beta-1}\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} , \end{aligned} \tag{4.31}$$

which, due to its domain of $[0,1]$, is well suited for approximating probabilities.

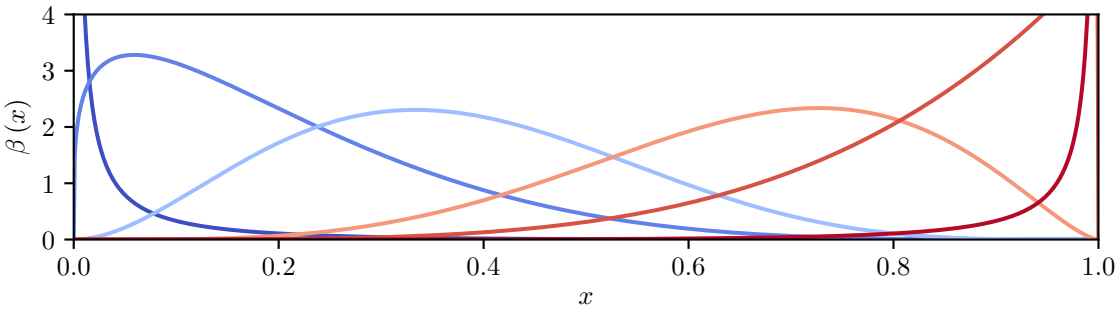


Figure 4.8: Probability density functions of different parameterized $\beta$-distributions. Using Eq. (4.32), we can interpolate between the target probability estimate, which corresponds to the colors of the curves. The mean $\mu_\beta$ of the $\beta$-distributions corresponding to red curves are closer to the probability of $x = 1$, whereas the blue curves approach a mean probability of $x = 0$.

Moreover, the $\beta$-distribution is parameterized through two values $\alpha, \beta \in \mathbb{R}$, which essentially resemble a weighing between the two domain borders 0 and 1. To make the approximation easier, we define the parameters as a function of a single scalar $\psi \in [-1, 1]$, such that

$$\begin{aligned} \alpha =& \text{clip}\left[\left(\psi_{max} - \psi_{min}\right)\psi + \psi_{max}; \psi_{min}, \psi_{max}\right] \\ \beta =& \text{clip}\left[-\left(\psi_{max} - \psi_{min}\right)\psi + \psi_{max}; \psi_{min}, \psi_{max}\right] . \end{aligned} \tag{4.32}$$

Consequently, it is possible to smoothly interpolate between estimating a mean probability of $\mu_\beta \approx 0$ with a corresponding $\psi = -1$ to $\mu_\beta \approx 1$ with $\psi = 1$. A visualization of multiple differently parameterized $\beta$-distributions can be seen in Fig. 4.8, where the color corresponds to $\psi$ and thus to the estimated mean probability $\mu_\beta$. Note that the $\beta$-distribution only addresses the uncertainty about the parameterization of the underlying Bernoulli distribution of termination.

Furthermore, to train $\mathcal{F}$, we employ the negative log-likelihood loss, which is incorporated into the total loss of the prediction model, resulting in the loss

$$\begin{aligned} & L_{\mathcal{E}_o, \mathcal{E}_z, \mathcal{R}, \mathcal{D}, \mathcal{F}} \\ =& -\mathbb{E}\left[\sum_t \ln \overleftarrow{\mathcal{E}_o}\left(o_t | s_t\right) + \ln \overleftarrow{\mathcal{E}_z}\left(z_t | s_t\right) + \ln \mathcal{R}\left(r_t | s_t\right) + \ln \mathcal{F}\left(f_t | s_t\right)\right. \\ & \left. - \beta\, \text{KL}\left[\overrightarrow{\mathcal{E}_z}\left(s_{t;[o_t,z_t]} \middle| \overrightarrow{\mathcal{E}_o}\left(s_{t;[o_t]} \middle| \mathcal{D}\left(\tilde{s}_t | s_{t-1}, a_{t-1}\right), o_t\right), z_t\right) \middle|\middle| \mathcal{D}\left(s_t | s_{t-1}, a_{t-1}\right)\right]\right] . \end{aligned} \tag{4.33}$$

Observe how the first distribution in the KL-divergence consists of the full filtered posterior of the one-step prediction using $\mathcal{D}$, whereas the second distribution is solely given through $\mathcal{D}$. Intuitively, this means we encourage the dynamics model $\mathcal{D}$ to mimic the filtering process as well as possible, leading to a more accurate state prediction. For the complete derivation of the loss, please follow Appendix A.

If the target probabilities of the environment are given as continuous signals, $\mathcal{F}$ will be able to predict the likelihood of reaching the specific terminal states. Consequently, we can utilize the termination model $\mathcal{F}$ to train the actor $\mathcal{A}$, such that

$$L_{\mathcal{A}} = -\mathbb{E}\left(\sum_{t=t_0}^{T} \mathcal{V}\left(\tilde{v}_{t\ldots T}|s_t\right) + \sum_i \lambda_{F_i} \mathcal{F}\left(F_{i;t}|s_t\right)\right), \tag{4.34}$$

for termination importance factors $\lambda_{F_i} \in \mathbb{R}$. Note that a negative importance factor signalizes a failed termination as $L_{\mathcal{A}}$ is minimized, and a positive importance factor implies a successful termination.

# 5. Evaluation

In this chapter, we will assess the overall performance of our model. First we will discuss the evaluation setup, followed by qualitative results. Later, we will show the contribution of specific model components and finalize with an assessment of the quantitative performance, which will also be compared to different baseline methods. In the end, we will prove the applicability of our learned model to the real world.

## 5.1. Experimental Setup

We evaluate different versions of our model and compare them to the baseline approach *Dreamer* and to A*-search combined with a closed-loop controller. Additionally, we perform real-world experiments, comparing our approach to the well-tested NimbRo obstacle avoidance algorithm, which has been used in the yearly RoboCup humanoid soccer competitions[24].

The exteroceptive closed-loop controller is following a path calculated by the A*-algorithm using a top-view image of the scene. Note that the observations are unrealistic in most applications, as a top-view image can generally not be provided, especially in unstructured environments. Moreover, the exact inference of a global pose requires a stable localization, which is often not available, e.g. due to sensor drifts.

In contrast to the baseline controller, the basic *Dreamer* model will be trained using only visual RGB inputs. These images include the obstacles, as well as the red target marker, which is not visible in the segmented observations. Additionally, the ground floor and the obstacles are visualized with randomized textures to mimic the visual complexity of the real world more closely.

In the end, the evaluation of our approach is based on the models trained using the hyperparameters found in Appendix B. We assess three different parameterizations and denote the resulting models with the letters A to C. The A-model utilizes direct observations and termination likelihood prediction, but does not incorporate the termination signal into the actor loss. In addition, the B-model applies the termination loss with a small weight. Furthermore, the the weight of

the termination loss is significantly increased in the C-model. Moreover, we also assess the performance after different steps during training. This way, we can observe the convergence behavior of the individual models.

The C-Model that has been trained for 2 million steps will be denoted as C-2.0M and is used in the evaluation if not stated otherwise. Our model represents the target position in local polar coordinates and utilizes segmented image observations, as well as gait velocities and the measured trunk orientation. All observations reflect the sensor capabilities of the real robot, facilitating the real-world transfer.

The C-2.0M model is trained for 2 million simulation steps, resulting in a total training time of around 2 days on a computer with an Intel i9-9990K CPU, 64GB of RAM and an nVidia GeForce 2080 Ti with 12GB of VRAM. This model is already able to solve simple scenes after 200.000 steps, resulting in less than five hours of training. The *Dreamer* model is trained for 3 million steps, after which the approach typically converges[4]. On the same computer, this results in a training time of approximately 3 days for the *Dreamer* model.

For the evaluation, we generate a set of 100 random scenes using the same sampling procedure as during training. Additionally, the qualitative assessment will be conducted on hand-crafted scenes, showing the behavior of the agent in difficult situations.

## 5.2. Experimental Results

Initially, we show qualitative results by presenting a random scenario our robot is able to navigate successfully. This scene can be seen in Fig. 5.1 and will be further denoted as R-11. Observe that the direct path to the target is blocked by an obstacle, thus the agent is forced to choose a trajectory around it. The resulting trajectory is shown in the right image, proving that the agent selects a short path to reach the target. Note that the obstacle sizes in the right image are increased by the use of the *Minkowski sum* (also known as *dilation*) with the robot diameter. This ensures, that the unoccupied pixels are valid locations for the agent. The same procedure is utilized for the calculation of an A* path. Observe that the agent chooses a tight trajectory around the obstacle without collisions, which is the main objective of our approach. In Fig. 5.2 screenshots of the episode at different time steps are shown.

Moreover, we display the policy for one of the handcrafted scenes in Fig. 5.3, where the agent has to navigate beside a line of obstacles to a target at the end of it. Note that the objects are structurally aligned, which stands in contrast with the random obstacles used during training. Therefore, we show a case where the agent

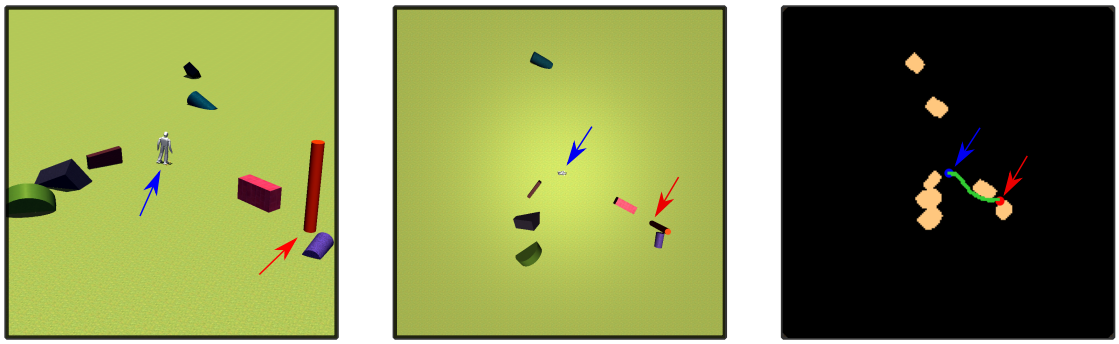

Figure 5.1: Random sample scene R-11. On the left, a third-person-view image of a part of the scene is shown, in the center we can see an excerpt of the scene from the top. On the right, a segmented, full image from the top is shown, where the obstacle size has been increased by use of the Minkowski sum with the robot diameter. Furthermore, we mark the target location as red and the starting position as blue. The walked path of our model is marked as a green trajectory in the rightmost image.

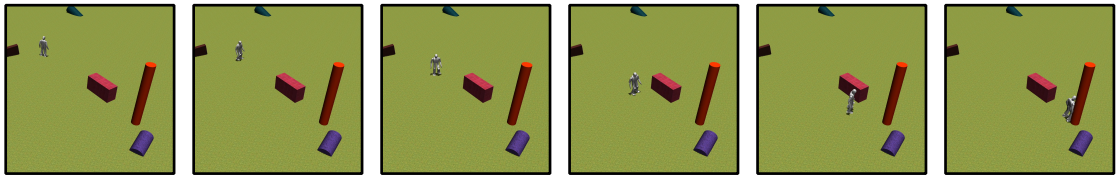


Figure 5.2: Screenshots of the random episode R-11, where the robot navigates around an obstacle.

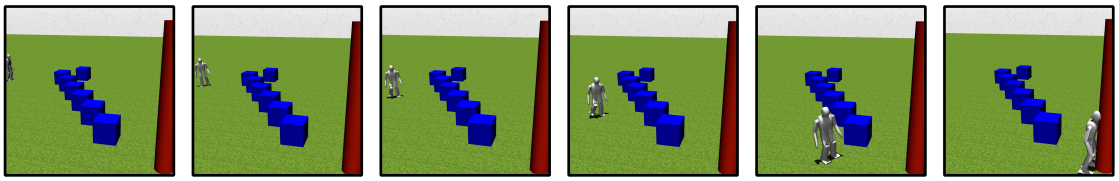


Figure 5.3: A sequence of images showing the path of the A-0.5M agent in a handcrafted scene. The agent follows a line of obstacles until it reaches the target at the end of the episode. Note that the direct path to the target is blocked by the obstacles.

successfully generalizes to a specific human-made scenario. Other hand-crafted and random episodes can be found in Appendix C, where we show simulation screenshots together with the approximate trajectory of the robot.

However, when evaluating several different episodes, the agent has a small tendency towards walking around only *one* obstacle. This can include instances where the path is blocked after the first obstacle has been overcome. This can result in the agent walking into the next obstacle and falling. Situations where two obstacles are very close to each other, while also blocking the path to the target are very unlikely to get sampled as training data. Therefore, these scenes were artificially constructed. Allowing larger obstacles or small, connected groups of obstacles that are explicitly sampled to block the way to the target, will lead to richer data and thus to a better performance of the agent in such cases.

In most episodes, the agent first turns towards the target and then continues walking with high forward velocity as seen in Fig. 5.4. This is the desired behavior as it is encouraged by the target rotation reward $r_{TR}$ in conjunction with the target distance reward $r_{TD}$. In this manner, the agent chooses gait velocities (GCVs) that keep the GCV penalty $r_V$ low. This behavior results in a stable but fast walk towards the target. Note that the robot also utilizes diagonal steps towards the end of the episode to correct for potential misalignment with the target.

Since the real world is only static in specific applications, we also evaluate the A-0.5M agent performance on a scene with dynamic obstacles. A manually created episode with dynamic obstacles can be seen in Fig. 5.5. Observe that the agent anticipates the movement of the object towards the end of the episode, choosing a particularly large trajectory around it. However, since the agent is not able to control the head movement, it is possible that the robot is pushed over by approaching obstacles that are outside of its field-of-view. We hypothesize that, this issue can be addressed by incorporating the head control into the action space of the agent. Even though the model did not encounter any dynamic objects

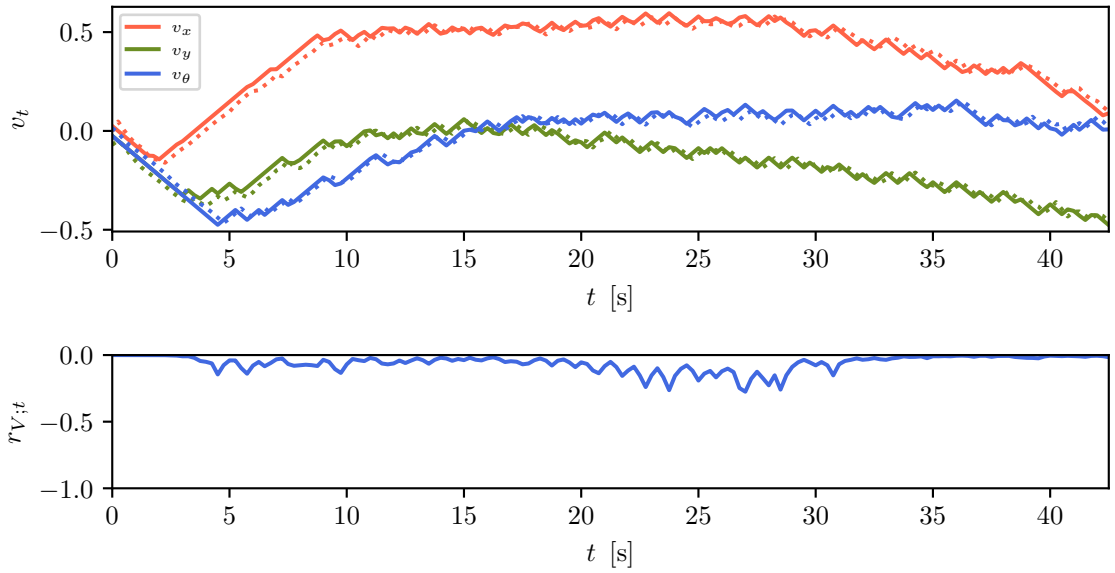


Figure 5.4: Gait commanded velocity (GCV) $v_t$ in its components (top), together with the GCV penalty (bottom). On the top, the dotted lines resemble the predicted direct observations $z_{0...2;t}$, which correspond to the GCV of the timestep. The red line resembled the forward $x$ velocity, the green line shows the sideways $y$ velocity, whereas the blue line displays the rotational $z$ velocity. Note that the agent first orientates using the rotational velocity and then proceeds with a continual forward motion (top), while mostly respecting the GCV limits set through the penalization (bottom). Observe that the agent utilizes diagonal movement towards the end of the episode to correct a misalignment to the target. These results are generated using our trained C-2.0M model, executed on the scene R-11.

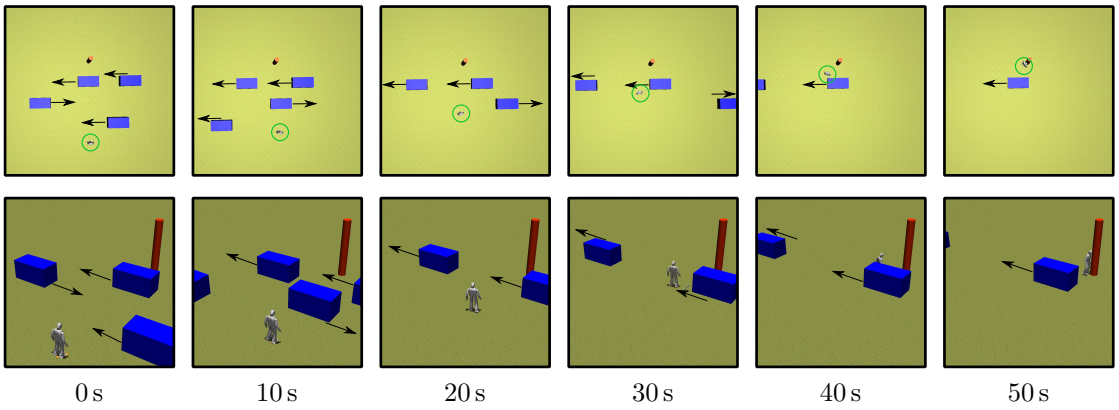


Figure 5.5: A sequence of images from an episode with dynamic obstacles. In the upper row a top-view of the scene is shown, whereas the lower row displays a third-person view. The A-0.5M agent is able to solve this scene reliably. Furthermore, the agent seems to anticipate the movement of the obstacles in the last four images of the sequence, as it chooses a trajectory with a large distance to the object.

during training, it is able to solve scenes with dynamic obstacles.

## 5.3. Model Accuracy

Since the performance of the agent strongly depends on the ability of the model to process and replicate the individual parts of the environment, it is crucial to assess its reconstruction. Generally, we evaluate the quality of approximation using the mean absolute error

$$\kappa_x = \frac{1}{t_{\max}} \sum_{t=0}^{t_{\max}} \|x_t - \tilde{x}_t\|_1 \ , \tag{5.1}$$

for a variable $x_t$ and its prediction $\tilde{x}_t$ over course of an episode of length $t_{\max}$.

The GCV with their corresponding predictions of the C-2.0M model have already been shown in Fig. 5.4, where the model is able to reconstruct the target gait velocities up to a very small error. More precisely, the mean absolute errors of each GCV component is below $\kappa_V < 0.026$ for the R-11 episode. Figures for all individual components of the direct observation $z_t$ are shown in Appendix D.

Similar to the direct observations, the C-2.0M model is also able to predict the reward accurately as seen in Fig. 5.6 with an accuracy of $\kappa_r = 0.029$. However, the model fails to predict the terminal reward that is collected at the end of the episode (not shown in the figure). Since the terminal rewards are discontinuous signals that are activated at only one specific time-step, it is hard to learn their

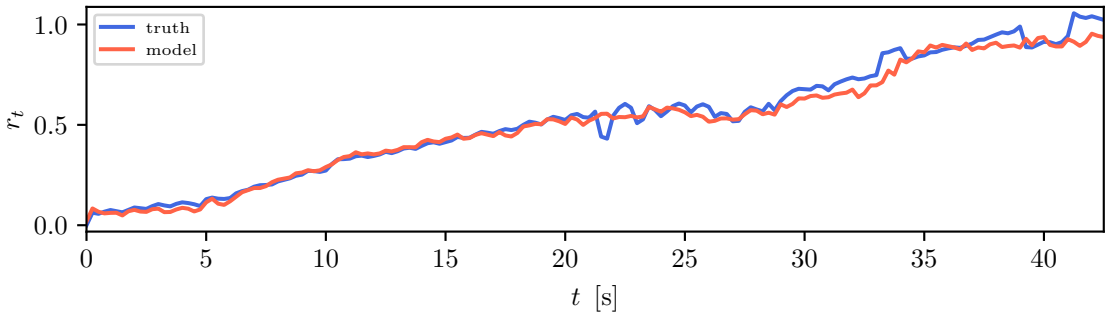


Figure 5.6: A sample of the true reward function $r_t$ (blue) without terminal reward is shown together with the model 1-step prediction $\tilde{r}_t$ (red). The model reconstructs the reward signal well, resulting in a mean absolute error of 0.029 for this sample. These graphs correspond to the performance of our trained C-2.0M model on the scene R-11.

predictions. Intuitively, if the model is not certain about the exact time-step of termination, the best way to minimize the reward reconstruction loss is to not try to predict the terminal reward at all, since the prediction at a wrong step would introduce a high loss. Consequently, these discontinuous subrewards will not be included in the calculation of the value and thus the actor will not be optimized with respect to them. This issue can be addressed by the introduction of a smooth terminal reward or, more generally, through the use of the termination likelihood as proposed in this thesis.

In contrast to the discontinuous terminal reward, the C-2.0M model is able to predict the termination likelihood (success/failure) up to a systematic model error as shown in Fig. 5.7. More precisely, the agent approximates the signals with accuracy $\kappa_{f0} = 0.082$ and $\kappa_{f1} = 0.072$. This proves that the model can successfully extract the notion of termination from the gathered experiences.

Finally, the image predictions are shown in Fig. 5.8. Note that the predicted images are closely resembling the shapes seen in the ground truth. More precisely, observe in the error images that the biggest errors occur around the edges of the objects, as well as for smaller objects in the distance. Both are only of subordinate importance for the task, as the rough shape of close objects is sufficient for obstacle avoidance. However, this is different if we require more complex path planning with longer planning horizons, since small objects in the distance may gain importance. For all images in the sequence, we calculate a mean absolute error of $\kappa_o = 0.025$.

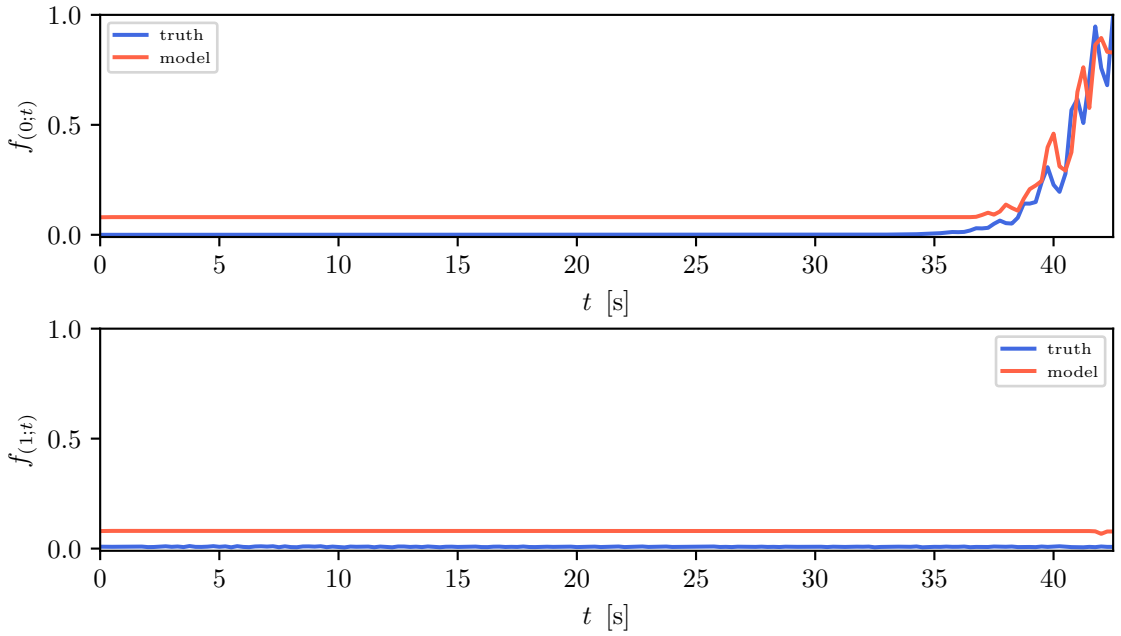


Figure 5.7: The terminal success and failure likelihoods $f_0, f_1$ are displayed for a sample episode. The true terminal likelihoods $f_i$ are depicted in blue, whereas the 1-step predictions are shown in red. Note that the model produces accurate predictions resulting in a mean absolute error of 0.082 for the success likelihood $\tilde{f}_0$ and 0.072 for the failure likelihood $\tilde{f}_1$. It is important to note, that the model can not reconstruct the signals precisely, since the beta distribution has been restricted to avoid singularities at probabilities of 0 and 1. Thus, the bottom graph mainly captures this systematic model error, as the robot does not experience strong disturbances in this episode. The measurements are taken from our trained C-2.0M model, executed on the scene R-11.

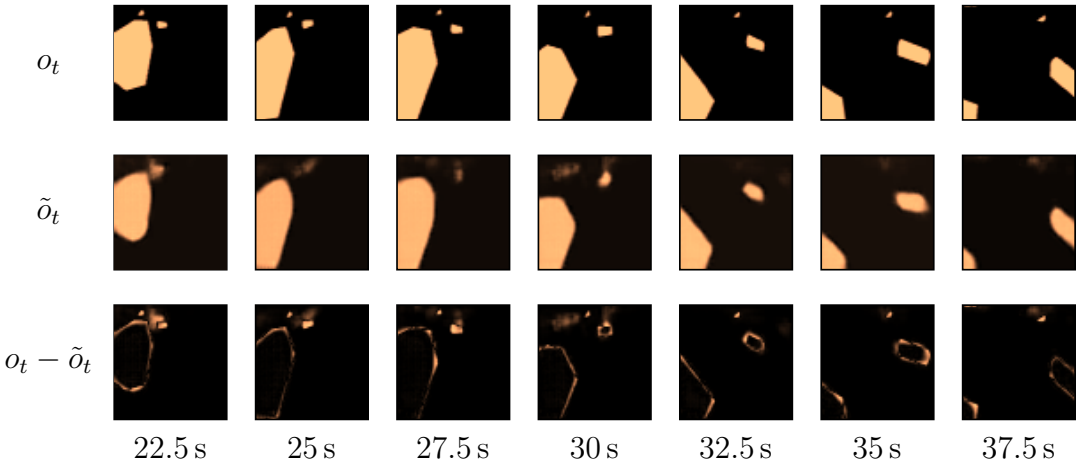


Figure 5.8: A comparison between the model image predictions and the ground truth. The true observations $o_t$ (top row) are shown with the 1-step predictions $\tilde{o}_t$ of the trained model (center row) for 10 different steps. Furthermore, the prediction error is shown in the bottom row. The model is able to reliably reconstruct the images from the significantly lower dimensional latent space. Additionally, the largest errors occur for objects in far distance, whereas close objects only suffer small errors around the edges. These observations correspond to the experiment R-11, evaluated with our trained C-2.0M model.

## 5.4. Performance

We evaluate the performance of our approach using different hyperparameters and total training times. An overview over the general performance of the model can be seen in Fig. 5.9, where the individual learning curves are shown. The C-Model converges the fastest and reaches a similar mean return to the A- and B-Model. However, the C-Model generates shorter episodes as seen in the bottom figure. In conjunction with the high return, this indicates that a successful termination is reached faster.

In addition, we execute each model on the same 100 random scenes, where each of the scenes is either executed once or 25 times, depending on the type of evaluation. Since the extensive evaluation of every intermediate model would require days of computation, we decide to assess some models only on 100 trials in contrast to the 2500 experiments of the full evaluation.

In total, 10 trained versions of our network have been evaluated, together with one evaluation of *Dreamer* and the A*-controller. The results of our different models can be seen in Table 5.1. Compared to our other models, the A-0.5M model performs best over all criteria, except for the critical failure rate. Still, the

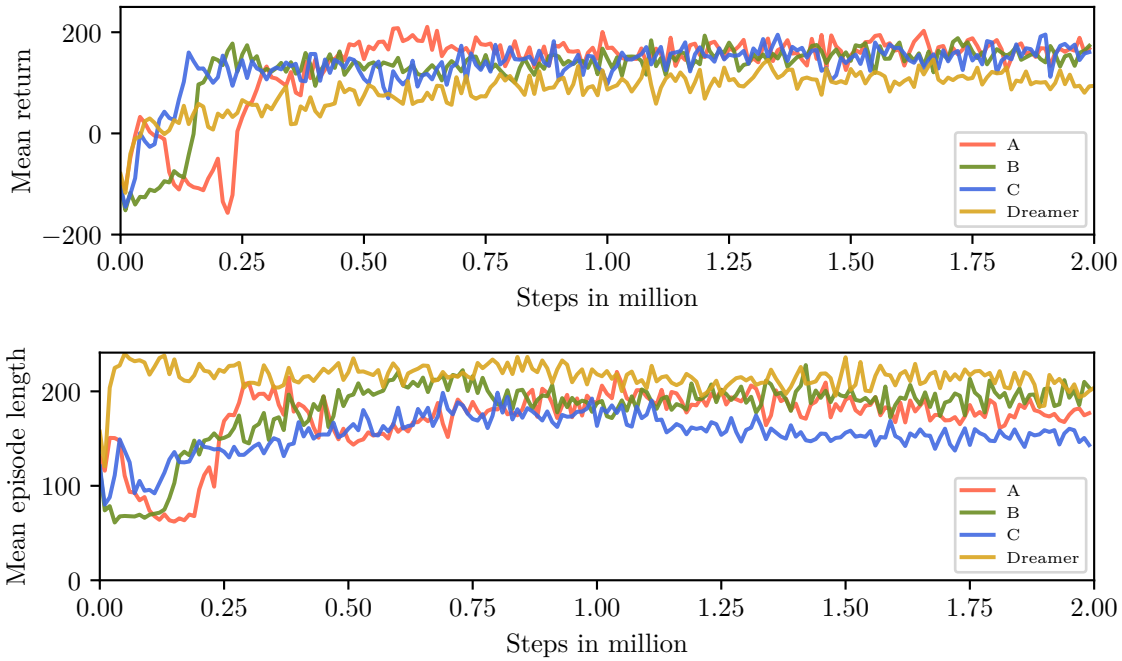


Figure 5.9: The learning curves are depicted for the mean return and mean episode length. The mean is generated by averaging the episodes in chunks of 10000 steps. Observe that the C-Model is the fastest to reach its maximum return, followed by the B- and A-Model. Furthermore, the C-Model generates noticeably shorter episodes. Note that the mean episode length has to be observed in conjunction with the mean return, as an episode can end in failure and success.

critical failure rate is only slightly above 1%, which is small enough to be neglected.

| Model | | Return | Ep. length | Success | Failure |
|---|---|---|---|---|---|
| A-0.5M | X | **247.11** | **149.26** | **98.7**% | 1.2% |
| A-1.5M | | 186.98 | 211.77 | 29.0% | **0.0**% |
| B-0.5M | X | 213.10 | 187.93 | 66.6% | 0.7% |
| B-1.0M | X | 206.33 | 194.06 | 53.7% | 0.8% |
| B-1.5M | X | 202.06 | 198.81 | 44.4% | **0.0**% |
| B-2.0M | X | 212.55 | 185.89 | 56.5% | **0.0**% |
| C-0.5M | | 204.00 | 187.67 | 66.0% | 6.0% |
| C-1.0M | | 182.25 | 211.02 | 45.0% | 8.0% |
| C-1.5M | | **236.47** | **157.75** | **92.0**% | 1.0% |
| C-2.0M | X | **216.20** | **167.79** | **80.3**% | 8.5% |

Table 5.1: Three different models compared at different training times. The A-model resembles a basic model with direct observations and termination likelihood approximation, but without a termination term in the actor loss. Additionally, the B-model includes an actor termination loss with low weight. In the C-model, the a higher value is chosen for the termination loss. Furthermore, lines marked with an "X" have been evaluated over 100 different scenes with 25 samples per scene, whereas the other evaluations only use 1 sample per scene. The three best results in each category are marked with bold font. It is possible to see that the A-0.5 model generally performs best, followed by C-1.5M and C-2.0M.

However, when training the A-model for more steps, the performance decreases. Since the terminal subreward can not be predicted due to the discontinuity it introduces, it is not included in the value prediction. This results in the actor being trained only on the other subrewards, while avoiding termination to achieve as many high rewards near the target as possible. Thus, the overall performance will decrease when the policy is refined. When considering the C-model on the other hand, its evolution is more stable as the performance does not decrease significantly at the end of training. Still, the overall performance does not match the A-0.5M model, which means that the hyperparameters have to be tuned to achieve a stable convergence. Overall, an adaptive learning rate for the actor model may alleviate this problem, as the oscillating training behavior of the C-model can indicate a high learning rate.

Furthermore, our approach outperforms the previous *Dreamer* method in most criteria as seen in Table 5.2. This is most apparent in comparison to our best model A-0.5M, which manages to solve 99% of the scenes, with a critical failure rate of

| | A* controller | Dreamer | Ours (A-0.5M) |
|---|---|---|---|
| Return | $178.42 \pm 38.29$ | $133.87 \pm 118.53$ | $\mathbf{247.11 \pm 31.09}$ |
| Episode length | $192.70 \pm 30.51$ | $208.68 \pm 52.62$ | $\mathbf{149.26 \pm 22.81}$ |
| Mean reward (no term.) | $0.510 \pm 0.053$ | $0.518 \pm 0.356$ | $\mathbf{0.593 \pm 0.049}$ |
| Success rate | 75% | 19% | **99**% |
| Critical failure rate | **1**% | 13% | **1**% |
| Training steps | – | 3M | **0.5M** |

Table 5.2: Overall performance comparison of our model with the original *Dreamer* approach, as well as a exteroceptive A* controller. Note that our model outperforms the other approaches in almost every aspect of the 100 test episodes. However, our agent produces a more aggressive policy than the A* controller, leading to a increase in failed episodes. Nevertheless, the A* controller depends on exteroceptive observations, which are impossible to retrieve from the real world.

1%. More precisely, our model not only exceeds the performance of *Dreamer*, but it also outperforms the exteroceptive A* controller, which has significantly more information about the scene than our agent. Additionally, the mean reward without the termination subrewards of the episodes generated with the A-0.5M model exceed the mean reward collected by the *Dreamer* method. Therefore, our policy not only accomplishes higher success rate through the incorporation of termination signals, but also outperforms the *Dreamer* approach in the task of reward maximization.

In addition, our models only require low computational power, executing in 2.9ms on a system with an Intel i7-3770K, 16GB of RAM and an nVidia GeForce GTX 1070 with 8GB of VRAM. On a computer equipped with an Intel i9-9990K CPU, 64GB of RAM and an nVidia GeForce RTX 2080Ti with 12GB of VRAM, the inference model takes approximately 1.7ms. In conjunction with the low step frequency $\tau = 4\,\mathrm{Hz}$ of the environment, the overall computational load of our model on the system performance is insignificant. Moreover, the execution time only varies slightly across different models, rendering each of the models useful for real-time control tasks. Additionally, the low computational effort and a VRAM footprint of approximately 1GB allow the models to be executed on portable platforms, such as robots or even mobile phones with GPUs.

## 5.5. Real-World Transfer

Since training a deep neural network typically requires a large amount of data, training in the real world is unfeasible in most applications. Furthermore, executing many experiments on a robot leads to wearing off the hardware, requiring expensive part replacements.

Therefore, we strive to enable a real-world transfer without retraining the model. Consequently, our proposed state representation mimics the sensor system of the robot, while reducing the visual complexity of the camera images through semantic segmentation. This allows a real world transfer with low additional effort and no retraining.

In Fig. 5.10 you can see a sequence of real world pictures comparing the NimbRo obstacle avoidance with our model. More precisely, the top row shows the well tested NimbRo obstacle avoidance, which is able to maneuver around robots placed in its way towards the soccer ball. Generally, this algorithm produces high gait velocities when there are no obstacles, but strongly reduces the velocities when approaching an obstacle. Furthermore, it tends to walk sideways to dodge the obstacle, utilizing the rotational gait velocity only slightly.

In contrast to the NimbRo obstacle avoidance algorithm, our approach navigates around a blue obstacle, blocking the path in the same way as the robots in the previous experiment. Since our model requires a segmented observation, we filter the camera image by a blue color-key in the HSV color space. Additionally, we apply an erosion and dilation step on the image to fill possible false negative pixels. This leads to a segmentation that includes only the obstacles, as we do not allow blue objects in the scene for the real world evaluation. Note that this simple segmentation technique is chosen as a proof of concept and can be easily exchanged with more complex segmentation models.

Generally, our model utilizes the rotational component of the gait velocity significantly stronger. The agent commands the robot to turn right to line up with the corner of the obstacle and then proceeds to walk in a smooth curve around it. During this experiment it was most apparent that the classical obstacle avoidance algorithm leads to a GCV that varies strongly, whereas our model seems to produce a trajectory where the robot walks with a constant speed. In summary, both approaches solve the scenes in approximately the same time, but show different behaviors.

Fig. 5.11 displays a sequence of real world and simulation images of a more challenging scene. Note that the two lines of obstacles overlap, leaving the robot only a tight corridor to pass through. In simulation, as well as in the real world, our C-2.0M model is able to solve this episode by navigating through the gap

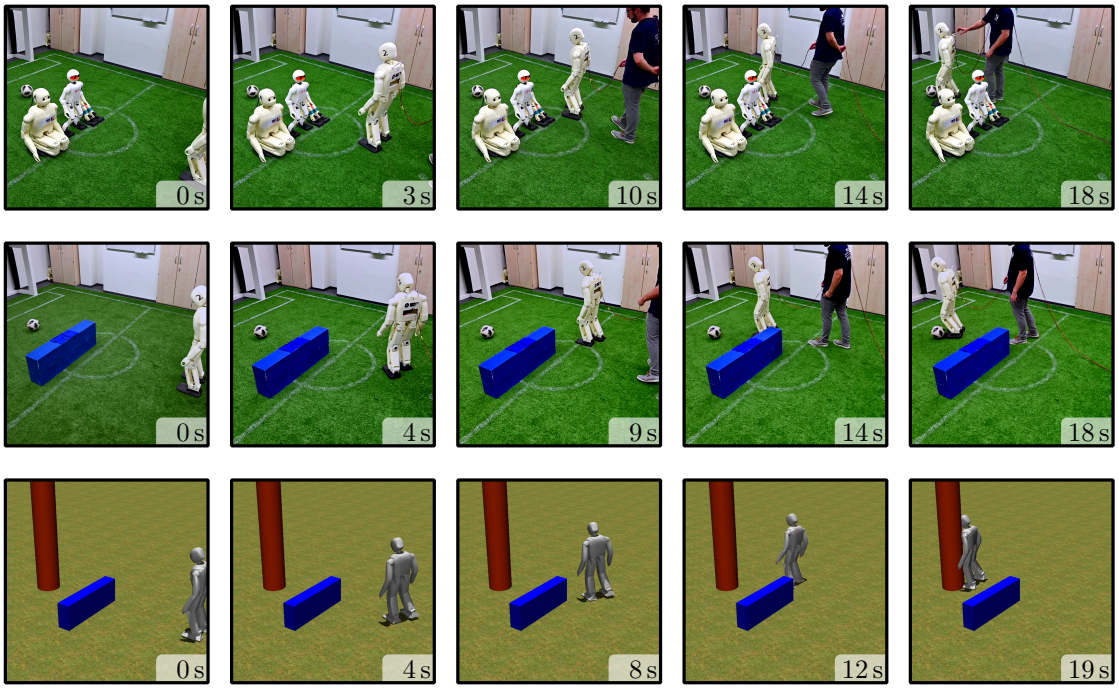


Figure 5.10: Image sequences from the real world. The top row shows the baseline NimbRo obstacle avoidance for robots, whereas the center row displays our C-2.0M model. Note that we use blue obstacle for simplifying the semantic segmentation. Nevertheless, we block the same path in both sequences, as the target is resembled by the soccer ball. Both approaches walk around the shorter path of the obstacle and reach the target after approximately the same time. In contrast to our method, the NimbRo algorithm moves slower near the obstacle, which it compensates with high velocities whenever the path is clear. On the bottom row, the same scene is displayed in simulation.

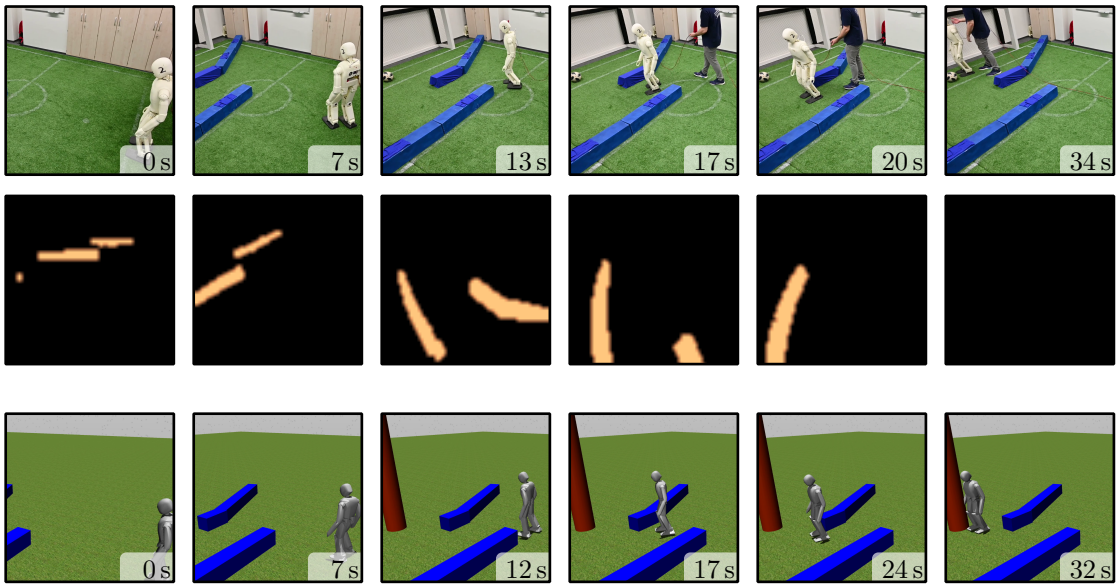


Figure 5.11: Pictures of a challenging scene in simulation and the real world. The top row shows the real robot navigating the scene by taking a right turn, followed by rotating left to walk through the tight corridor between the obstacles. In the end, the robot walks into the target by stepping laterally. The corresponding grayscale image observations of the real experiment are depicted in the center row. At the bottom, a recreated simulation scene is shown, where the agent chooses a route similar to the real-world episode. Both episodes have been generated using the C-2.0M model.

between the obstacles. In addition, the model manages to avoid contact with the real obstacles completely.

Moreover, we evaluate the real-world transfer with a dynamic obstacle as seen in Fig. 5.12. More precisely, the obstacle is resembled by a blue fabric, which is carried with a slightly slower velocity than the robots gait. Observe that the C-2.0M agent walks parallel to the obstacle, until it passes it towards the end of the episode. Subsequently, the robot walks towards the target with diagonal steps. In addition, we evaluate the exact same scene with the obstacle remaining stationary. Note that the robot maneuvers around the obstacle and chooses a short path towards the target. This proves that our approach is able to adapt to dynamic obstacles in the real world transfer, even though the agent does not encounter dynamic objects during training.

In the final experiment, we asses the capability of the C-2.0M model to plan with a dynamic target. As seen in Fig. 5.13, we disable the terminal success state and let the robot follow the target. Once the robot is close to the target, we move it to a new location in the agents field of view. Observe that the robot is

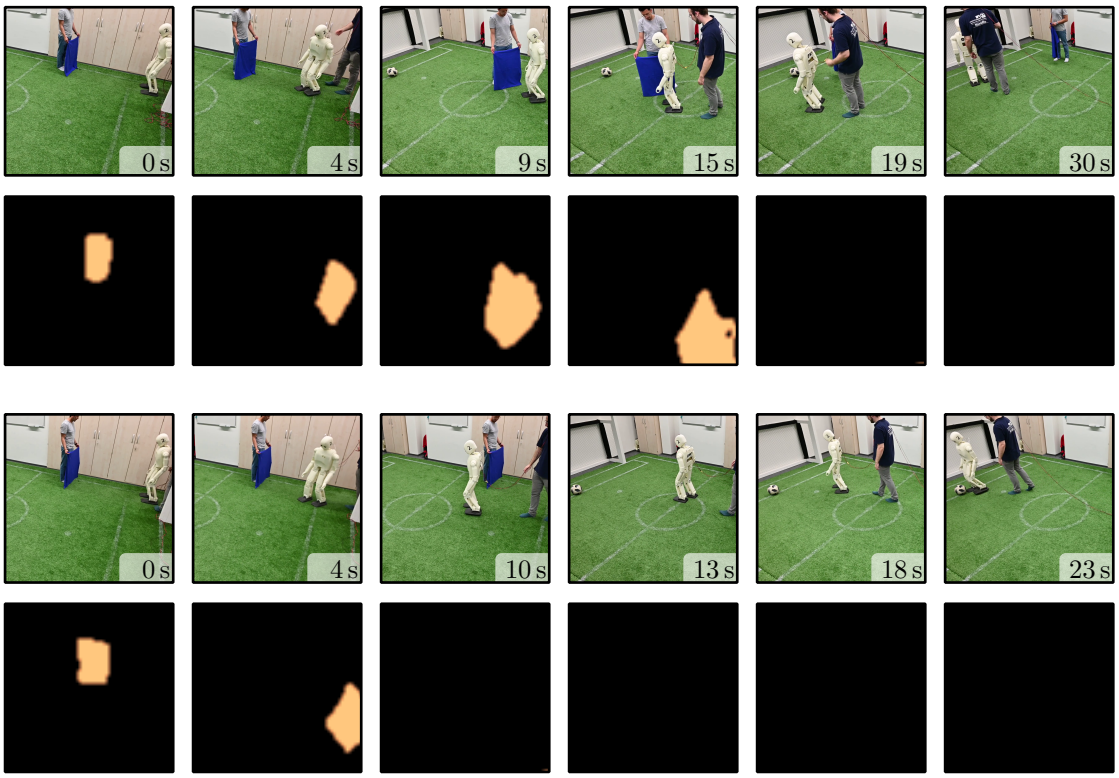


Figure 5.12: The C-2.0M agent is applied to a real-world scene with a dynamic obstacle. Observe in the two top rows that the object is moved with a velocity slower than the robot, requiring the agent to walk parallel to the obstacle. When the object is removed, the robot directly walks to the target with diagonal steps. In the two bottom rows, the same obstacle is kept static, allowing the agent to shortly maneuver around it before it continuing to walk straight towards the target. Below each real picture, the corresponding segmented image observation is depicted.

able to track the moving target and it successfully approaches the different target locations, despite only experiencing static targets during training.

Generally, it is observed that the A-0.5M agent performs significantly worse in the real world compared to the C-2.0M agent. This can be caused by the agent taking actions that behave differently in the real world than in the simulation. More precisely, the lateral velocities are far less responsive in the real world than in simulation. In the end, our final model with termination likelihood loss leads to a more robust model, which is easier to transfer to real world applications.


Figure 5.13: A sequence of images from the real-world transfer with a dynamic target (from left to right, top to bottom). Whenever the C-2.0M agent approaches the target closely, the ball is moved, resulting in a new target position. The model successfully manages to track the moving target, approaching it at different locations. Note that the images are taken from a single episode.

# 6. Future Work

## 6.1. Task Adaption

In this thesis, we have applied all models to the simulation of a NimbRo-OP2X robot and we evaluated the performance in the real world. The task consists of obstacle avoidance while approaching a pre-defined target position. While this is highly relevant for real-world application such as humanoid soccer, it can not fully showcase the long-term planning capabilities of the models. To keep the necessary effort in adaption of the simulation low, it is possible to utilize the same framework, but sample the environment differently. Most importantly this may include the construction of a closed maze, which the robot has to navigate in order to reach a given target position. This way it would be possible to assess whether the model is able to track its position over a longer horizon and revert choices that it has previously made. In general, this task is significantly harder and would establish a good base challenge for future investigation. Furthermore, this problem is highly relevant for real-world applications, where an agent may need to navigate through complex room structures to reach its target.

Moreover, by incorporating the control of the head into the action space, it is possible to utilize the full capabilities of the robot. This will especially improve the agents ability to avoid dynamic obstacles, as well as help to keep the target in the agents field of view in the real world application. Furthermore, the data can be trained to reconstruct depth data from the simulator, similar to Xie *et al.*[15]. In addition to an adaptive learning rate, these points can lead to a more robust and versatile model.

## 6.2. Hierarchical Model

Since our basic model does not address long-term planning directly, this shortcoming may be discussed in future work. Due to the inherent inaccuracy of image predictions over longer horizons, we propose to split the model into two parts as seen in Fig. 6.1: A lower model which includes the image processing and a higher model, which processes the direct observations. More precisely, the lower model

has a similar shape to our current model, where it consists of a latent state $s_t$ that is constructed using an image observation $o_t$. Furthermore, it can utilize the L-dynamics $\mathcal{L}$ to predict the following step given an action $a_{t+1}$. However, this action is not only generated using the lower latent state, but also utilizes a high-level action $k_{t+1}$ which is chosen based on the high-level state representation $h_t$. Additionally, it is possible to use the high-level representation to generate the reward and value prediction, along with the termination likelihood if desired. To pass low-level state information to the higher level state, an Autoencoder is trained between $s_t$ and $h_t$, thus $h_t$ handles the lower level state similar to an observation.

Note that the high-level action $k_{t+1}$ is completely latent, as a specific $k_{t+1}$ only has a distinct meaning to the environment in context with the low-level actor. This makes training the model more challenging, as the model can not utilize experience replay for training the $\mathcal{H}$ network. Since $k_{t+1}$ is interpreted by the low-level actor, the meaning may change over the course of training, leading to other observations and thus to other actions that will be taken for the rest of the episode. Still, the model can be trained online or utilize only a portion of previous episodes as experience replay. The last training option would be to train the high-level model solely on extrapolated states from the lower model, similar to how the value function is trained in the original approach. As this solution is the most straight-forward of the mentioned options, we have already trained a first model. However, training $\mathcal{H}$ only in imagination means that the direct observation $z_t$ would need to be part of the lower model, and consequently the accuracy of the higher level model would be limited by the accuracy of the lower level model. Thus, it is advised to assess the other options for training the higher level.

Furthermore, as $k_t$ is completely latent, it is necessary to choose its shape. In our first experiments we use a relaxed one-hot encoding of the actions, which allows to build a tree over trajectories of different sequences of actions. To enforce that $k_t$ has an actual effect on the latent state, we propose a regularization using the latent states $h_t$ in the constructed tree of trajectories. Intuitively, by calculating the difference between these $h_t$, e.g. by using a generalization of the Jensen-Shannon divergence, it is possible to regularize different $k_t$ to result in different $h_t$, thus forcing $k_t$ to have an actual effect on the state.

In summary, we propose a hierarchical model with a completely latent action component as an interesting subject for further research.

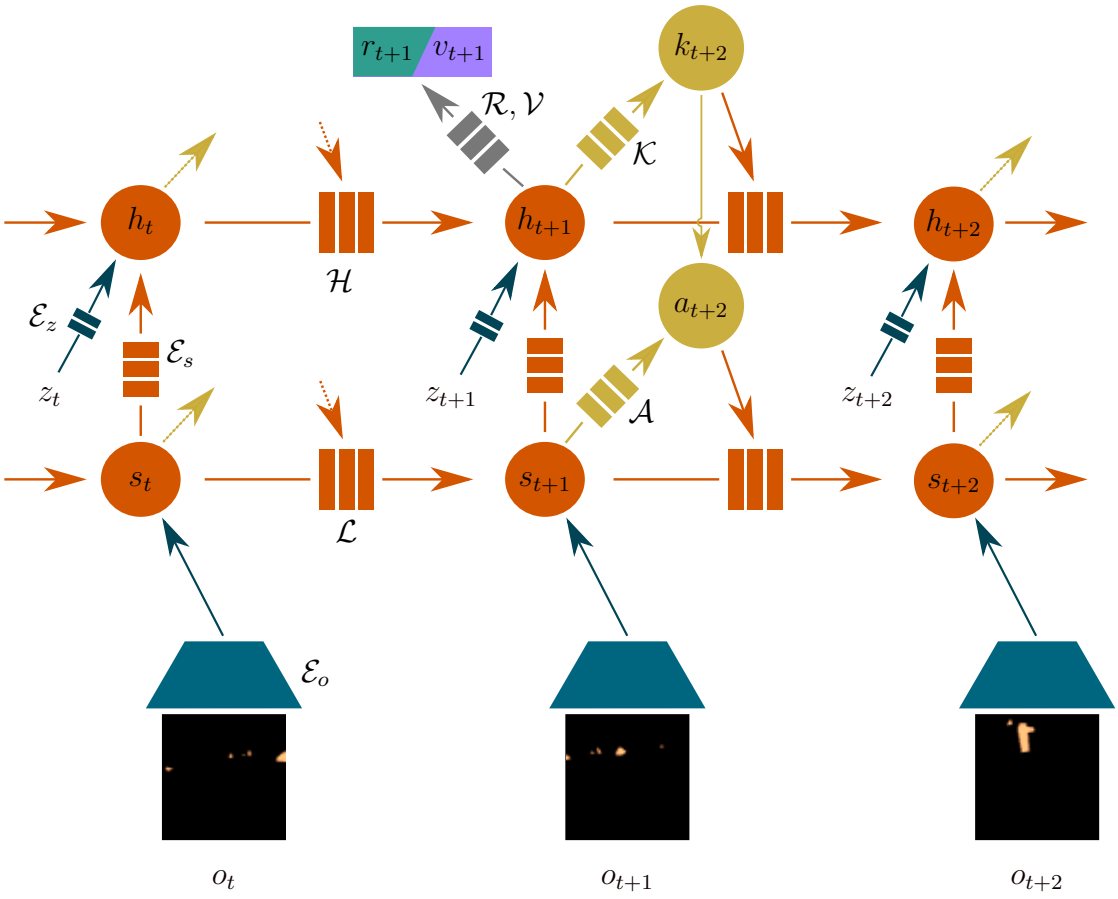


Figure 6.1: A preview of the hierarchical model. Similarly to the original approach, the images are encoded to a state representation $s_t$, which is then used to construct a second latent state $h_t$. This higher-level representation also incorporates the high-level observation $z_t$ and outputs a fully latent high-level action $k_{t+1}$. This $k_{t+1}$ is utilized by the low-level actor to produce an output action $a_{t+1}$. Furthermore, the high-level state can be used to predict rewards $r_t$ or values $v_t$.

# 7. Conclusion

In this thesis, we have demonstrated a learned approach that utilizes a latent dynamics model to solve a navigation problem in scenes with obstacles. Furthermore, this latent representation is constructed by combining segmented images with the non-visual observations, which supply the agent with rich information about the environment. Additionally, our model anticipates its success and failure by approximating the termination likelihood. This leads to a fast model with high performance.

Our model exceeds the performance of the *Dreamer* approach and outperforms the exteroceptive A* controller. Therefore, we have shown that our model produces a competitive policy, utilizing only the available sensors of the real robot. Subsequently, by taking advantage of an abstract image representation, we successfully apply our model to the real world. More precisely, we deploy our agent to the "NimbRo-OP2X" robot, resulting in a similar performance to the hand crafted obstacle avoidance method, which is regularly used in the annual RoboCup humanoid soccer competitions. Finally, we have shown that our model generalizes to dynamic obstacles and is able to track and follow a dynamic target in the simulation, as well as the real world.

# A. Loss Derivation

Similar to [4], we will rely on the information bottleneck objective

$$L\left[p\left(o_{1:T}, r_{1:T} | s_{1:T}, a_{1:T}\right)\right] = I\left(s_{1:T}; \left(o_{1:T}, r_{1:T}\right) | a_{1:T}\right) - \beta I\left(s_{1:T}; i_{1:T} | a_{1:T}\right) , \quad \text{(A.1)}$$

which has already been covered in Chapter 3. For sake of brevity, we write $s_{t;[o_t,z_t]} =: s_t$ whenever the filtering is not of direct interest in the following derivation.

We derive the first mutual information as follows by discarding the constant data likelihood, followed by the subtraction of the non-negative KL divergence. Using the definition of the KL divergence in the last step, we find the final lower bound for the first term:

$$\begin{aligned}
& I\left(s_{1:T}; \left(o_{1:T}, z_{1:T}, r_{1:T}, f_{1:T}\right) | a_{1:T}\right) \\
= \; & \mathbb{E}_{p(o_{1:T}, z_{1:T}, r_{1:T}, f_{1:T}, s_{1:T}, a_{1:T})} \\
& \left[\sum_t \ln p\left(o_{1:T}, z_{1:T}, r_{1:T}, f_{1:T} | s_{1:T}, a_{1:T}\right) - \ln p\left(o_{1:T}, z_{1:T}, r_{1:T}, f_{1:T} | a_{1:T}\right)\right] \\
\overset{\text{const}}{=} \; & \mathbb{E}\left[\sum_t \ln p\left(o_{1:T}, z_{1:T}, r_{1:T}, f_{1:T} | s_{1:T}, a_{1:T}\right)\right] \\
\geq \; & \mathbb{E}\left[\sum_t \ln p\left(o_{1:T}, z_{1:T}, r_{1:T}, f_{1:T} | s_{1:T}, a_{1:T}\right)\right] \\
& - \mathrm{KL}\left[p\left(o_{1:T}, z_{1:T}, r_{1:T}, f_{1:T} | s_{1:T}, a_{1:T}\right) || \prod_t \overleftarrow{\mathcal{E}_o}\left(o_t | s_t\right) \overleftarrow{\mathcal{E}_z}\left(z_t | s_t\right) \mathcal{R}\left(r_t | s_t\right) \mathcal{F}\left(f_t | s_t\right)\right] \\
= \; & \mathbb{E}\left[\sum_t \ln \overleftarrow{\mathcal{E}_o}\left(o_t | s_t\right) + \ln \overleftarrow{\mathcal{E}_z}\left(z_t | s_t\right) + \ln \mathcal{R}\left(r_t | s_t\right) + \ln \mathcal{F}\left(f_t | s_t\right)\right] .
\end{aligned} \quad \text{(A.2)}$$

The second term of the mutual information is derived through substitution of the true data distribution with our models. This can be done w.l.o.g., since the latent state $s_t$ is solely defined through our models. Thus, we can substitute the true distribution $p$ with those models that process the given data that defines $p$.

By applying the definition of the KL divergence in the last step, we can conclude with a representation for the second mutual information:

$$\begin{aligned}
& I\left(s_{1:T}; i_{1:T} | a_{1:T}\right) \\
= \; & \mathbb{E}_{p(o_{1:T}, z_{1:T}, r_{1:T}, f_{1:T}, s_{1:T}, a_{1:T}, i_{1:T})} \\
& \left[\sum_t \ln p\left(s_t | s_{t-1}, a_{t-1}, i_t\right) - \ln p\left(s_t | s_{t-1}, a_{t-1}\right)\right] \\
= \; & \mathbb{E}\left[\sum_t \ln p\left(s_t | s_{t-1}, a_{t-1}, o_t, z_t\right) - \ln p\left(s_t | s_{t-1}, a_{t-1}\right)\right] \\
= \; & \mathbb{E}\left[\sum_t \ln \overrightarrow{\mathcal{E}_z}\left(s_{t;[o_t,z_t]} \middle| \overrightarrow{\mathcal{E}_o}\left(s_{t;[o_t]} \middle| \mathcal{D}\left(\tilde{s}_t | s_{t-1}, a_{t-1}\right), o_t\right), z_t\right) - \ln \mathcal{D}\left(s_t | s_{t-1}, a_{t-1}\right)\right] \\
= \; & \mathbb{E}\left[\sum_t \mathrm{KL}\left[\overrightarrow{\mathcal{E}_z}\left(s_{t;[o_t,z_t]} \middle| \overrightarrow{\mathcal{E}_o}\left(s_{t;[o_t]} \middle| \mathcal{D}\left(\tilde{s}_t | s_{t-1}, a_{t-1}\right), o_t\right), z_t\right) \middle|\middle| \mathcal{D}\left(s_t | s_{t-1}, a_{t-1}\right)\right]\right] .
\end{aligned} \tag{A.3}$$

Therefore, we can formulate the information bottleneck objective

$$\begin{aligned}
& I\left(s_{1:T}; (o_{1:T}, r_{1:T})\right) - \beta I\left(s_{1:T}; i_{1:T} | a_{1:T}\right) \\
\geq \; & \mathbb{E}\left[\sum_t \ln \mathcal{E}\left(o_t | s_t\right) + \ln \mathcal{R}\left(r_t | s_t\right)\right] \\
& - \beta\, \mathbb{E}\left[\sum_t \mathrm{KL}\left[\overrightarrow{\mathcal{E}_z}\left(s_{t;[o_t,z_t]} \middle| \overrightarrow{\mathcal{E}_o}\left(s_{t;[o_t]} \middle| \mathcal{D}\left(\tilde{s}_t | s_{t-1}, a_{t-1}\right), o_t\right), z_t\right) || \mathcal{D}\left(s_t | s_{t-1}, a_{t-1}\right)\right]\right] \\
= \; & \mathbb{E}\Bigg[\sum_t \ln \overleftarrow{\mathcal{E}_o}\left(o_t | s_t\right) + \ln \overleftarrow{\mathcal{E}_z}\left(z_t | s_t\right) + \ln \mathcal{R}\left(r_t | s_t\right) + \ln \mathcal{F}\left(f_t | s_t\right) \\
& \qquad - \beta\, \mathrm{KL}\left[\overrightarrow{\mathcal{E}_z}\left(s_{t;[o_t,z_t]} \middle| \overrightarrow{\mathcal{E}_o}\left(s_{t;[o_t]} \middle| \mathcal{D}\left(\tilde{s}_t | s_{t-1}, a_{t-1}\right), o_t\right), z_t\right) || \mathcal{D}\left(s_t | s_{t-1}, a_{t-1}\right)\right]\Bigg] \\
= \; & -L_{\mathcal{E}_o, \mathcal{E}_z, \mathcal{R}, \mathcal{F}, \mathcal{D}} .
\end{aligned} \tag{A.4}$$

# B. Hyperparameters

This appendix summarizes the hyperparameters used to train our different models. For model A, all parameters will be listed, whereas for the other agents, we will only display changes. Furthermore a model denoted as "Y-$x$M" uses the model parameters Y and has been trained for $x \cdot 10^6$ steps. Follow the code if more information is required.

**Model A:**

precision: float16
direct observation size: 7
image shape: $(64, 64, 1)$
parallel environments: 8
test environments: 1
parallelization: Threads
action repeat: no
random dataset prefill: 5000 steps
batch size: 50
batch length: 50
train at every: 1000 recorded steps
training iterations: 100
imagination horizon: 15 steps
termination dimension: 2
term. signal threshold: 0.98
exploration noise: additive gaussian
exploration scale: 0.3
minimum exploration: 0
exploration decay: no

RSSM deterministic size: 200 units
RSSM stochastic size: $2 \times 30$ units
dense layer activation: ELU
CNN layer activation: ReLU
CNN channel multiplier: 32
direct observation loss scale: 512
termination loss scale: 10
KL loss scale: 1
minimum nats in KL loss: 3.0
observation model LR: 6e−4
value model LR: 8e−5
actor model LR: 8e−5
initial action std: 5
clip gradient to: 100
bellman discount: 0.95
random number generator seed: 42
**termination scale in $\mathcal{A}$-loss:** $[0, 0]$

**Model B:**

termination scale in $\mathcal{A}$-loss: $[-200, 125]$

**Model C:**

termination scale in $\mathcal{A}$-loss: $[-1000, 1000]$

**Environment Parameters:**

| | |
|---|---|
| maximum number of obstacles $n_{\rho,\max}$: | 12 |
| image resolution: | $64 \times 64$ |
| image channels: | grayscale (showing obstacles) |
| step frequency $\tau$: | 4 Hz |
| action space: | $[-0.025, 0.025]^3$ |
| sample number of obstacles: | yes |
| dynamic obstacles: | no |
| blocking probability $p_{\text{block}}$: | 0.2 |
| blocking minimum path overlap: | 10 cm |
| resample blocking: | no |
| curriculum/learning stages: | no |
| multidimensional termination: | yes (success, failure, timeout) |
| randomize textures: | no (yes for *Dreamer* evaluation) |
| randomize colors: | no (yes for *Dreamer* evaluation) |

| | |
|---|---|
| $\lambda_{TD}$: | 1.0 |
| $\lambda_{TO}$: | 0.2 |
| $\lambda_{V}$: | 0.35 |
| $\lambda_{OD}$: | 0.25 |
| $\lambda_{S}$: | 1.75 |
| $\lambda_{F}$: | 0.85 |

| | |
|---|---|
| $P_{T,\min}$: | 4 m |
| $P_{T,\max}$: | 10 m |
| $P_{\rho,\min}$: | 4 m |
| $P_{\rho,\max}$: | 15 m |
| $d_{\rho,\min}$: | [0.2 m, 0.2 m, 0.2 m] |
| $d_{\rho,\max}$: | [2 m, 2 m, 2 m] |

# C. Policy in Different Scenes

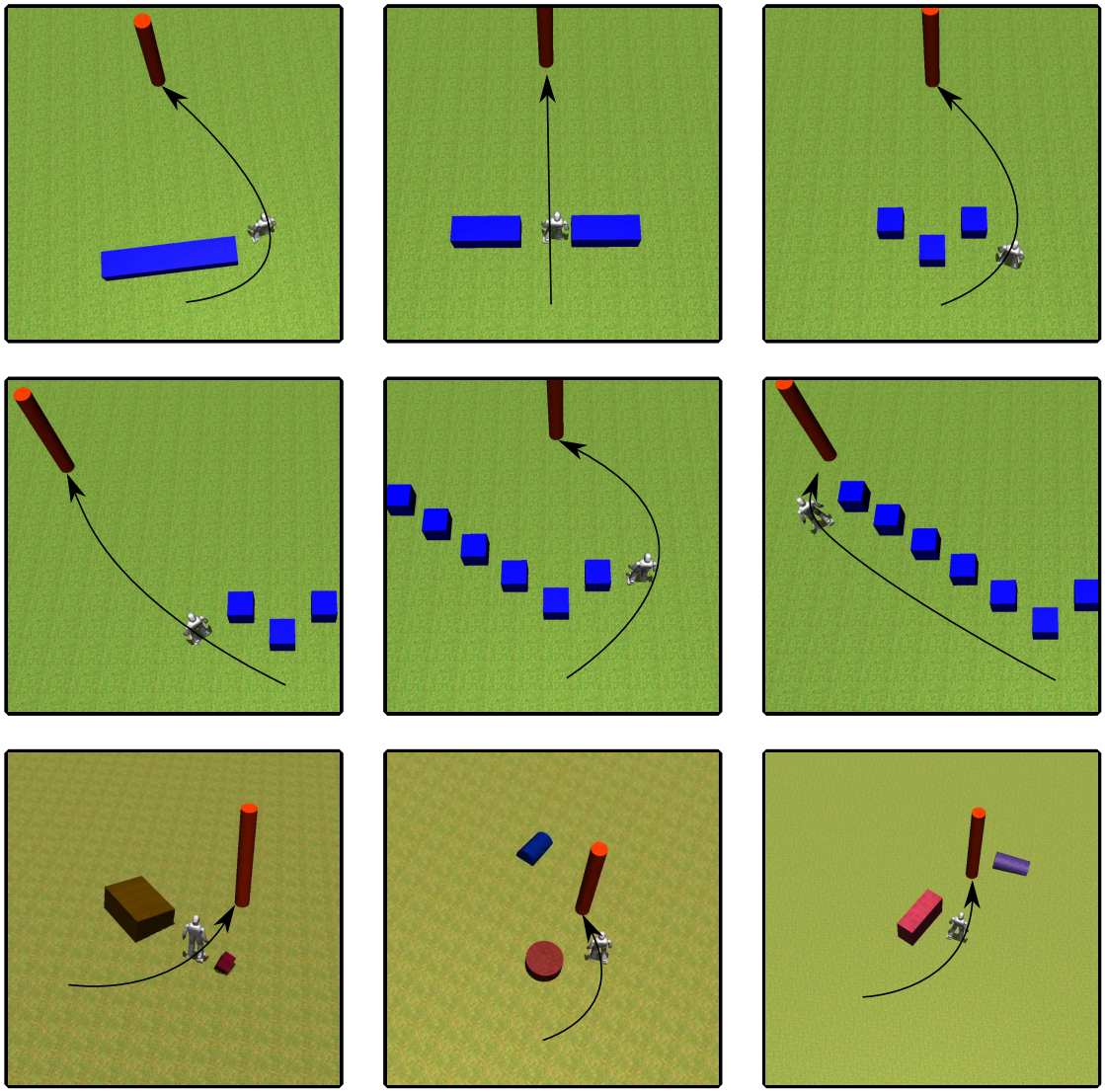

Figure C.1: The C-2.0M policy is shown in multiple scenes. The environments include hand-crafted situations (top and center row), as well as randomly generated scenes (bottom row). The arrows mark the approximate path of the robot.

# D. Direct Observation Predictions

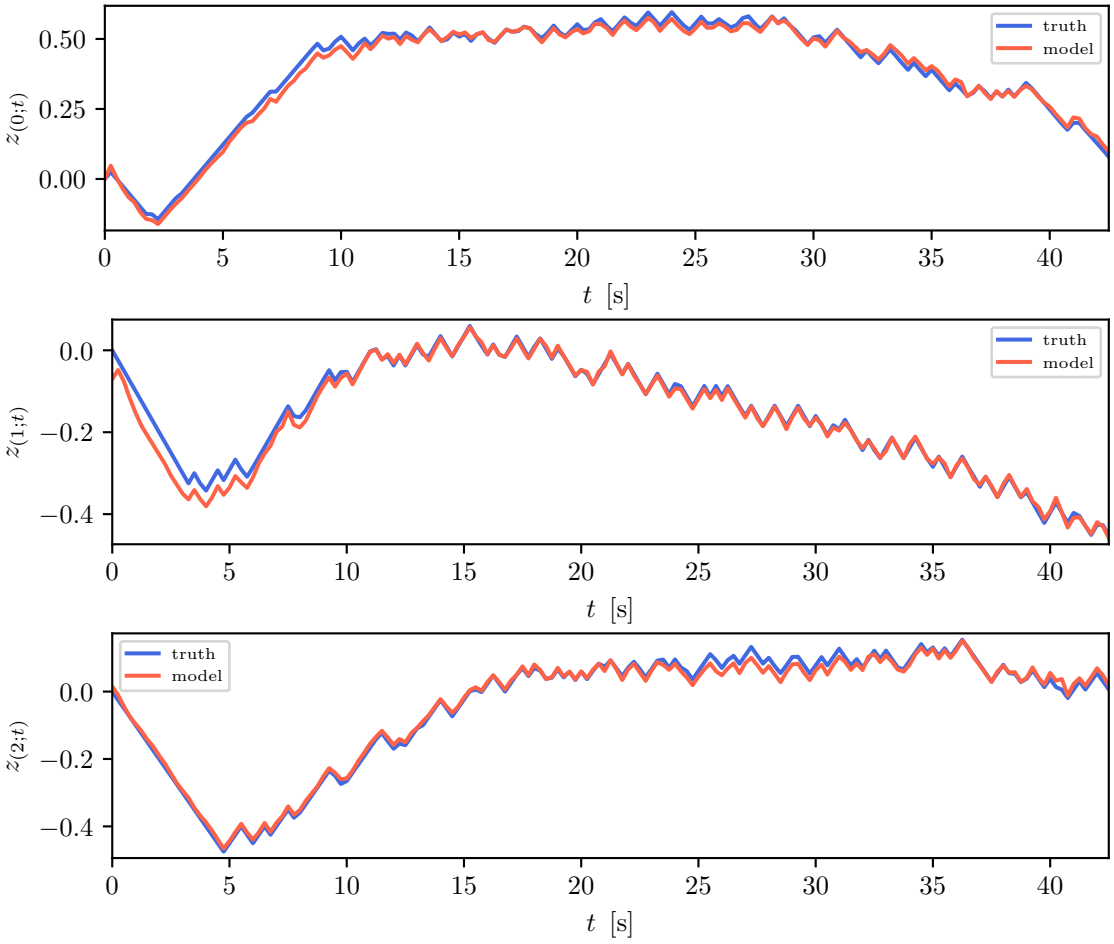


Figure D.1: The predictions of the GCV part in the direct observation (red) and the ground truth (blue). Furthermore, $z_{(0;t)}$ corresponds to the $x$ velocity, $z_{(1;t)}$ shows the $y$ velocity and the rotational $z$ velocity is resembled by $z_{(2;t)}$.

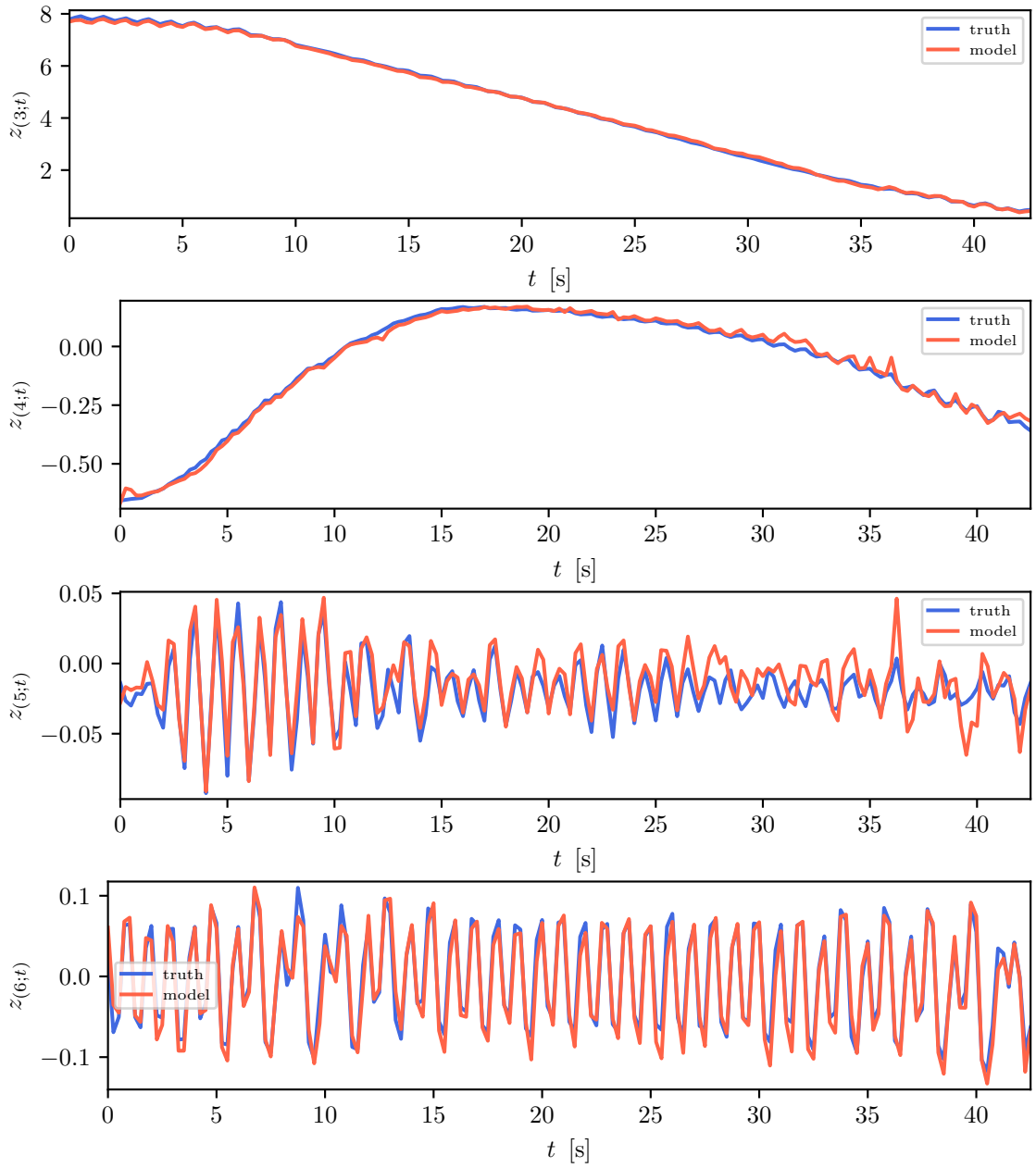


Figure D.2: The predictions of the relative target location $z_{(3;t)}$, $z_{(4;t)}$ and the robot roll and pitch rotations $z_{(5;t)}$, $z_{(6;t)}$ are shown in red. The blue curves display the corresponding ground truths.

# List of Figures

# Bibliography


1. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K. & Hassabis, D. *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm* 2017. arXiv: `1712.01815 [cs.AI]`.

2. Barth-Maron, G., Hoffman, M. W., Budden, D., Dabney, W., Horgan, D., TB, D., Muldal, A., Heess, N. & Lillicrap, T. P. *Distributed Distributional Deterministic Policy Gradients* in *6th International Conference on Learning Representations (ICLR), Canada* (2018). `https://openreview.net/forum?id=SyZipzbCb`.

3. Hafner, D., Lillicrap, T. P., Fischer, I., Villegas, R., Ha, D., Lee, H. & Davidson, J. *Learning Latent Dynamics for Planning from Pixels* in *Proceedings of the 36th International Conference on Machine Learning (ICML), California, USA* (eds Chaudhuri, K. & Salakhutdinov, R.) **97** (2019), 2555–2565. `http://proceedings.mlr.press/v97/hafner19a.html`.

4. Hafner, D., Lillicrap, T. P., Ba, J. & Norouzi, M. *Dream to Control: Learning Behaviors by Latent Imagination* in *8th International Conference on Learning Representations (ICLR), Ethiopia* (2020). `https://openreview.net/forum?id=S1lOTC4tDS`.

5. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T. & Hassabis, D. Mastering the game of Go without human knowledge. *Nature* **550,** 354– (Oct. 2017).

6. Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T. & Hassabis, D. Mastering the game of go with deep neural networks and tree search. *Nature* **529,** 484–503 (2016).

7. Ficht, G., Farazi, H., Brandenburger, A., Rodriguez, D., Pavlichenko, D., Allgeuer, P., Hosseini, M. & Behnke, S. *NimbRo-OP2X: Adult-Sized Open-Source 3D Printed Humanoid Robot* in *18th IEEE-RAS International Conference on Humanoid Robots (Humanoids), China* (2018), 1–9. `https://doi.org/10.1109/HUMANOIDS.2018.8625038`.

8. López-Nicolás, G., Sagüés, C., Guerrero, J. J., Kragic, D. & Jensfelt, P. Switching visual control based on epipoles for mobile robots. *Robotics auton. syst.* **56,** 592–603 (2008).

9. Xu, D., Wang, L. & Tan, M. *Image Processing and Visual Control Method for Arc Welding Robot* in *2004 IEEE International Conference on Robotics and Biomimetics (ROBIO), China* (2004), 727–732. https://doi.org/10.1109/ROBIO.2004.1521871.

10. Becerra, H. M., López-Nicolás, G. & Sagüés, C. Omnidirectional visual control of mobile robots based on the 1D trifocal tensor. *Robotics auton. syst.* **58,** 796–808 (2010).

11. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. & Hassabis, D. Human-level control through deep reinforcement learning. *Nature* **518,** 529–533 (2015).

12. Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D. & Riedmiller, M. A. *Deterministic Policy Gradient Algorithms* in *Proceedings of the 31th International Conference on Machine Learning (ICML), China* **32** (2014), 387–395. http://proceedings.mlr.press/v32/silver14.html.

13. Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. & Wierstra, D. *Continuous control with deep reinforcement learning* in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings* (eds Bengio, Y. & LeCun, Y.) (2016). http://arxiv.org/abs/1509.02971.

14. Ha, D. & Schmidhuber, J. *Recurrent World Models Facilitate Policy Evolution* in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems (NeurIPS), Canada* (eds Bengio, S., Wallach, H. M., Larochelle, H., Grauman, K., Cesa-Bianchi, N. & Garnett, R.) (2018), 2455–2467. http://papers.nips.cc/paper/7512-recurrent-world-models-facilitate-policy-evolution.

15. Xie, L., Wang, S., Markham, A. & Trigoni, N. Towards Monocular Vision based Obstacle Avoidance through Deep Reinforcement Learning. *CoRR.* arXiv: 1706.09829. http://arxiv.org/abs/1706.09829 (2017).

16. Biswas, J. & Veloso, M. M. *Depth camera based indoor mobile robot localization and navigation* in *IEEE International Conference on Robotics and Automation (ICRA), Minnesota, USA* (2012), 1697–1702. https://doi.org/10.1109/ICRA.2012.6224766.

17. Iacono, M. & Sgorbissa, A. Path following and obstacle avoidance for an autonomous UAV using a depth camera. *Robotics auton. syst.* **106,** 38–46 (2018).

18. Schaub, A., Baumgartner, D. & Burschka, D. Reactive Obstacle Avoidance for Highly Maneuverable Vehicles Based on a Two-Stage Optical Flow Clustering. *IEEE trans. intell. transp. syst.* **18,** 2137–2152 (2017).

19. Lobos-Tsunekawa, K., Leiva, F. & Ruiz-del-Solar, J. Visual Navigation for Biped Humanoid Robots Using Deep Reinforcement Learning. *IEEE robotics autom. lett.* **3,** 3247–3254 (2018).

20. Tishby, N., Pereira, F. C. & Bialek, W. *The information bottleneck method* in *Proc. of the 37-th Annual Allerton Conference on Communication, Control and Computing* (1999), 368–377. https://arxiv.org/abs/physics/0004057.

21. Alemi, A. A., Fischer, I., Dillon, J. V. & Murphy, K. *Deep Variational Information Bottleneck* in *5th International Conference on Learning Representations (ICLR), France* (2017). https://openreview.net/forum?id=HyxQzBceg.

22. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J. & Zaremba, W. OpenAI Gym. *Corr.* arXiv: 1606.01540. http://arxiv.org/abs/1606.01540 (2016).

23. Allgeuer, P. & Behnke, S. *Bipedal Walking with Corrective Actions in the Tilt Phase Space* in *18th IEEE-RAS International Conference on Humanoid Robots (Humanoids), China* (2018), 1–9. https://doi.org/10.1109/HUMANOIDS.2018.8624957.

24. Rodriguez, D., Farazi, H., Ficht, G., Pavlichenko, D., Brandenburger, A., Hosseini, M., Kosenko, O., Schreiber, M., Missura, M. & Behnke, S. *RoboCup 2019 AdultSize Winner NimbRo: Deep Learning Perception, In-Walk Kick, Push Recovery, and Team Play Capabilities* in *RoboCup 2019: Robot World Cup XXIII* (eds Chalup, S. K., Niemüller, T., Suthakorn, J. & Williams, M.) **11531** (2019), 631–645. https://doi.org/10.1007/978-3-030-35699-6%5C_51.