Radboud University Nijmegen

Faculty of Science

Policy Distillation from World Models

Parametric and Non-Parametric Policy Models for Robotic Control with Model-Based Reinforcement Learning

Thesis MSc Computing Science

Author:
Matthijs Biondina

Promotor Ghent University:
prof. dr. Francis wyffels

Counsellor Ghent University:
Andreas Verleysen

Supervisor:
prof. dr. Elena Marchiori

Second reader:
dr. Serge Thill

June 2020

Contents

1 Abstract

Model-Based Reinforcement Learning (MBRL) is a significantly more sample-efficient alternative to model-free reinforcement learning. In MBRL, supervised learning is used to optimize a deep neural network model of the transition and reward dynamics in an environment (i.e. a world model). This leads to a rich training signal which enables training such a world model with fewer environment interactions than typically required to train model-free methods. However, an open question in MBRL remains how to distill effective policies from world models. Here we show that PlaNet, an MBRL method that has been shown to perform well in simulation, is ineffective for an in-vivo environment with real-time constraints. In this work, we propose a better alternative by exploiting the gradients of the world model to train a parametric policy model with back-propagation through physics. We have evaluated the performance of PlaNet on an inverted pendulum swing-up task on an in-vivo robot setup. We have found that it is unable to reach an acceptable control frequency and is too intolerant to latency to effectively solve the task. We have evaluated two alternative policy distillation methods: imitation learning and a gradient-based approach. We have found imitation learning ineffective and have shown that our gradient-based approach not only allows for faster inference but also achieves better performance than PlaNet. Our results demonstrate that with the right architectures to alleviate model-bias, exploiting the gradients of learned world models can be an effective method for policy distillation in model-based reinforcement learning.

2 Introduction

One of the basic paradigms in machine learning is Reinforcement Learning (RL), a computational approach to learning from interaction [48]. Breakthroughs in deep learning and gains in computing power have formed the foundations for numerous recent advances with RL on complex simulated domains [23, 36, 56] and to a limited degree in the physical domain [1, 33]. Although applications in the simulated domain can have intrinsic value [16, 24], many of these applications are valuable as demonstrations of theoretical fitness for a purpose in the natural world [40, 44, 58]. Yet, the successful application of RL methods is often much more challenging in the physical domain than in simulation [43].

Two major factors contribute to this discrepancy between the performance of RL in the simulated and the physical domains. First, RL agents require large amounts of environment interaction [38]. In the simulated domain, this is usually not an issue, because environment interaction comes at little-to-no cost, and environments can be computed at a faster-than-real pace. However, in the physical domain, extensive interaction with e.g. a robotic setup is costly and can often be risky [55]. Additionally, transition dynamics in physical systems are not guaranteed to be stationary. As motors wear or lighting conditions change, the transition dynamics of the system change correspondingly [39]. Second, even the best simulation environments are simplified interpretations of the real world that fail to capture a combination of key challenges in the physical domain [14]. The natural world is dynamic and random; and RL methods have been shown to be intolerant to perturbations in the environment [28]. Therefore, methods that work in simulation often do not work in the natural world. Extensive research has been done in closing this 'reality gap', mainly for purposes of transferring experience from the simulated to the physical domain [3, 17, 45, 54]. However, this approach has so far only resulted in successful applications of RL methods on relatively simple tasks such as moving a robot arm with external resistance [7], grasping simple objects [51], or pick-and-place tasks with toy blocks [26]. These limited results with the transfer of experience from simulation to the natural world, show that there is still some way to go

in closing the reality gap.

It must be noted that if the reality gap could be sufficiently closed to accommodate transfer learning for complex physical tasks, counterintuitively, RL for the physical domain becomes largely obsolete. If transferring experience from simulation to the natural world inherently requires simulation models that are very close to the ground truth of their physical environments, the benefits of RL fade against those of classic search [19] and classic control theory [31]. These classic techniques outperform RL in complex environments by optimizing actions against a ground truth model of the environment, assuming that such a model is available. Hence, to exploit the major benefit of RL over the classic approaches, the fact that RL is not dependent on ground truth models, more sample-efficient training regimes and model architectures are required that make training RL agents in the physical domain viable.

A recent field of study that has significantly improved the sample-efficiency of RL methods is Model-Based Reinforcement Learning (MBRL) [8, 20, 21, 22, 27]. In contrast to model-free RL, MBRL methods use environment interaction to learn an explicit model of the transition dynamics of the environment: a world model. Since training a world model can be done in a fully supervised manner, MBRL requires fewer environment interactions than model-free RL and allows for training larger models. Yet, an open question in the field of MBRL remains how to distill effective policies from the learned world model. Often policy models in MBRL architecture learn to exploit deficiencies in the world model, rather than to produce effective behaviors in the environment. This phenomenon is known as model-bias [13]. However, methods that have the world model explicitly model stochasticity in the environment have alleviated this issue [5, 9, 27].

Three major types of algorithms can be distinguished for policy distillation with MBRL [57]. First, dyna-style algorithms use model-free algorithms to search for a policy by interacting with the world model rather than with the real environment [9, 32, 34]. Although dyna-style algorithms require less environment-interaction than pure model-free RL, much computing power is required for the model-free optimization of the policy. Second, gradient-based algorithms exploit the analytic gradient of the reward score in the world model to directly optimize a policy through gradient descent. However, this approach tends to be restricted to small domains with non-parametric world models [13] or in domains where the dynamics of the system are known to the model [50]. Considerable advances have been made recently on gradient-based policy optimization with differentiable physics simulators [11, 12]. Third, shooting algorithms do not train a parametric policy, but directly optimize actions at inference under the model predictive control framework [6] by testing action sequences against the world model [8, 41]. Since this optimization procedure needs to be repeated at each time step, shooting methods tend to be time-inefficient at inference.

The Deep Planning Network (PlaNet) is a shooting algorithm that combines several recent advances in MBRL [21]. To counteract model-bias, the former's authors explicitly model the deterministic and stochastic components of state transition in PlaNet's dynamics model and introduce latent overshooting, a training objective that promotes accurate multi-step predictions. In order to speed up inference, action sequences are evaluated in latent embedding space. This removes obsolete computational strain compared to methods that generate synthetic rollouts in observation or pixel space [27]. Although PlaNet shows promising results on a variety of simulated environments, the innovations it brings have not been shown to provide viable solutions for robotic control tasks in the physical domain.

The focus of this paper lies on evaluating the applicability of PlaNet on in-vivo robotic control. While planning in latent space allows for faster inference, it remains to be proven that the method is flexible enough for robotic control tasks with strict time-dependency constraints. In conjunction, we propose adaptations to PlaNet that

have the potential for more flexibility. We choose an inverted pendulum swing-up task, in which a pendulum on a cart must be swung up from its stable equilibrium to its unstable equilibrium and must be kept balanced in the unstable equilibrium. This task requires the agent to act on two different time scales. The swing-up component of the task has loose restrictions on control frequency and fine-grained control since it only requires pumping energy into the system. Balancing requires more fine-grained control and puts a stricter restriction on control frequency. The remainder of this paper is structured as follows: in section 3 we give an overview of the world model in the standard PlaNet architecture; in section 4 we describe the policy model deployed in the standard PlaNet architecture and we propose two alternative policy distillation methods to train parametric policy models that enable faster inference; in section 5 we define the inverted pendulum swing-up task and describe our simulated and physical setups; in section 6 we report and discuss our experimental results; finally, in section 7 we provide a summary of our work and discuss open areas for future research.

3 World Models

Our initial aim is to evaluate the applicability of PlaNet [21] on in-vivo robotic control, because PlaNet combines several recent advances in MBRL and promises faster inference through planning in latent space. To that end, we start from the former's authors' work on PlaNet. Because action selection in the PlaNet framework is performed through optimization at inference time, PlaNet may still be too slow to reach an acceptable control frequency for our real-world setup, despite the speed-ups gained from planning in latent space. Therefore, we additionally propose several adaptations to the PlaNet framework to allow for faster inference. A full implementations of our models has been published at: www.github.com/MatthijsBiondina/WorldModels/

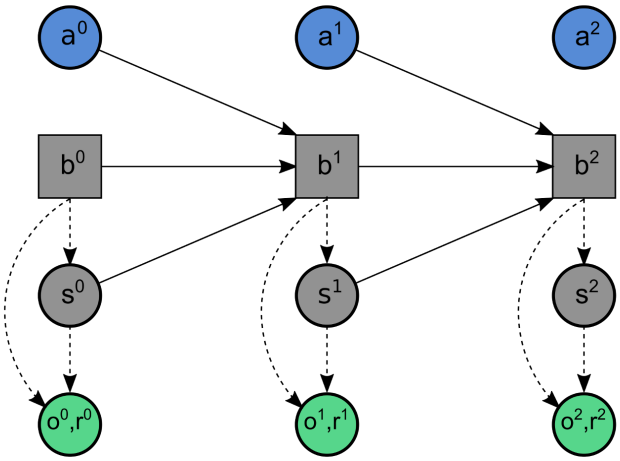


Figure 1: A Partially Observable Markov Decision Problem. Blue nodes represent inputs, green nodes represent observable outputs, and gray nodes represent hidden variables. Solid lines denote deterministic relationships and dashed lines denote stochastic relationships.

PlaNet is a model-based reinforcement learning method for discrete-time Partially Observable Markov Decision Processes (POMDP) [21]. PlaNet learns a neural dynamics model of the world that incorporates both deterministic and stochastic elements of state-transition. During runtime, PlaNet chooses actions by repeatedly optimizing a sequence of actions with its world model and executing the first action in the sequence. PlaNet

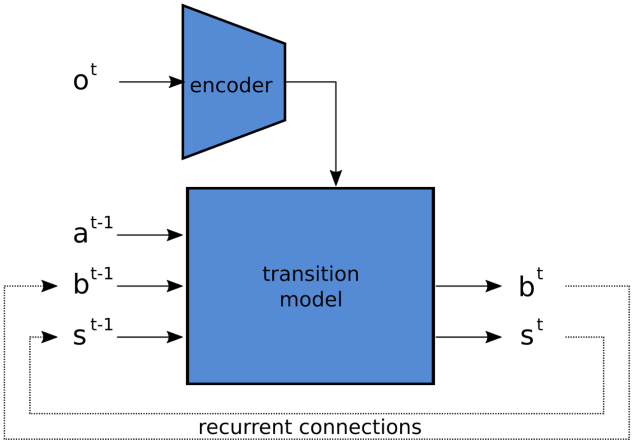


Figure 2: The encoder and transition model predict a posterior latent state distribution $(b, s_{prior})^t$ from the previous latent state distribution $(b, s)^{t-1}$, the previous action a^{t-1}, and the current observation o^t.

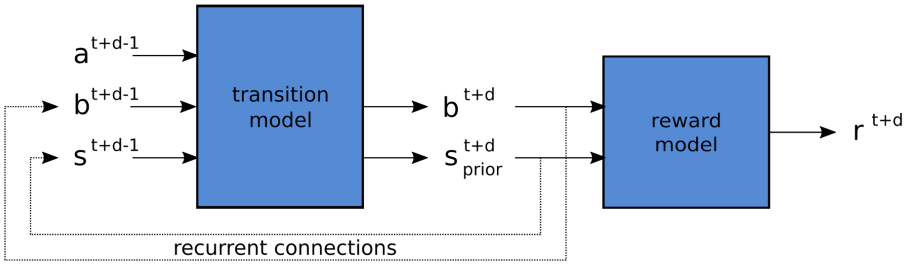


Figure 3: The transition and reward model predict prior distributions over future latent states $(b, s_{prior})^{t+d}$ and expected reward r^{t+d}, given the previous latent state distribution $(b, s)^{t+d-1}$ and an action a^{t+d-1}.

can be dissected in two distinct components: the world model and the planner. The world model will be covered here, while the planner will be described in section 4.1.

A POMDP models an agent in an environment of which the state-transition is governed by a Markov decision process. The state, however, is not directly observable to the agent and must be deduced from observations sampled from a distribution conditional on the state. For example, the state of a physical system is determined by the position and energy of the particles that make up the system. An agent acting in this system may deduce the state from sensor observations such as a camera feed.

Assuming a fixed time step t, we describe our POMDP with an 8-tuple ($\boldsymbol{B}$, $\boldsymbol{S}$, $\boldsymbol{A}$, $\boldsymbol{T}$, $\boldsymbol{R}$, $\boldsymbol{\Omega}$, $\boldsymbol{O}$, $\boldsymbol{\gamma}$).[1] $\boldsymbol{B} \subseteq \mathbb{R}^{\beta}$ and $\boldsymbol{S} \subseteq \mathbb{R}^{\sigma}$ are the sets of real-valued vectors respectively describing the deterministic and stochastic components of the hidden states of the environment (see Figure 1). $\boldsymbol{A} \subseteq \mathbb{R}$ is a set of real-valued actions. $\boldsymbol{T} : b_t, s_t \sim p(b_t, s_t | b_{t-1}, s_{t-1}, a_{t-1})$ with $b \in \boldsymbol{B}$, $s \in \boldsymbol{S}$ and $a \in \boldsymbol{A}$ is a state-transition function, describing the deterministic and the stochastic components of the environment dynamics. $\boldsymbol{R} : r_t \sim p(r_t | b_t, s_t)$ with $r_t \in \mathbb{R}$ is the reward-function. $\boldsymbol{\Omega} \subset \mathbb{R}^{\omega}$ is the set of ω-dimensional, real-valued observations. $\boldsymbol{O} : o_t \sim p(o_t | b_t, s_t)$ with $o \in \boldsymbol{\Omega}$ is an observation-

[1] POMDPs are commonly described as 7-tuples, making no distinction between the deterministic and stochastic components of hidden states. It has been shown that explicitly modeling stochasticity alleviates model-bias [5]. To maintain consistency, we adopt this notation here for the general case of a POMDP.

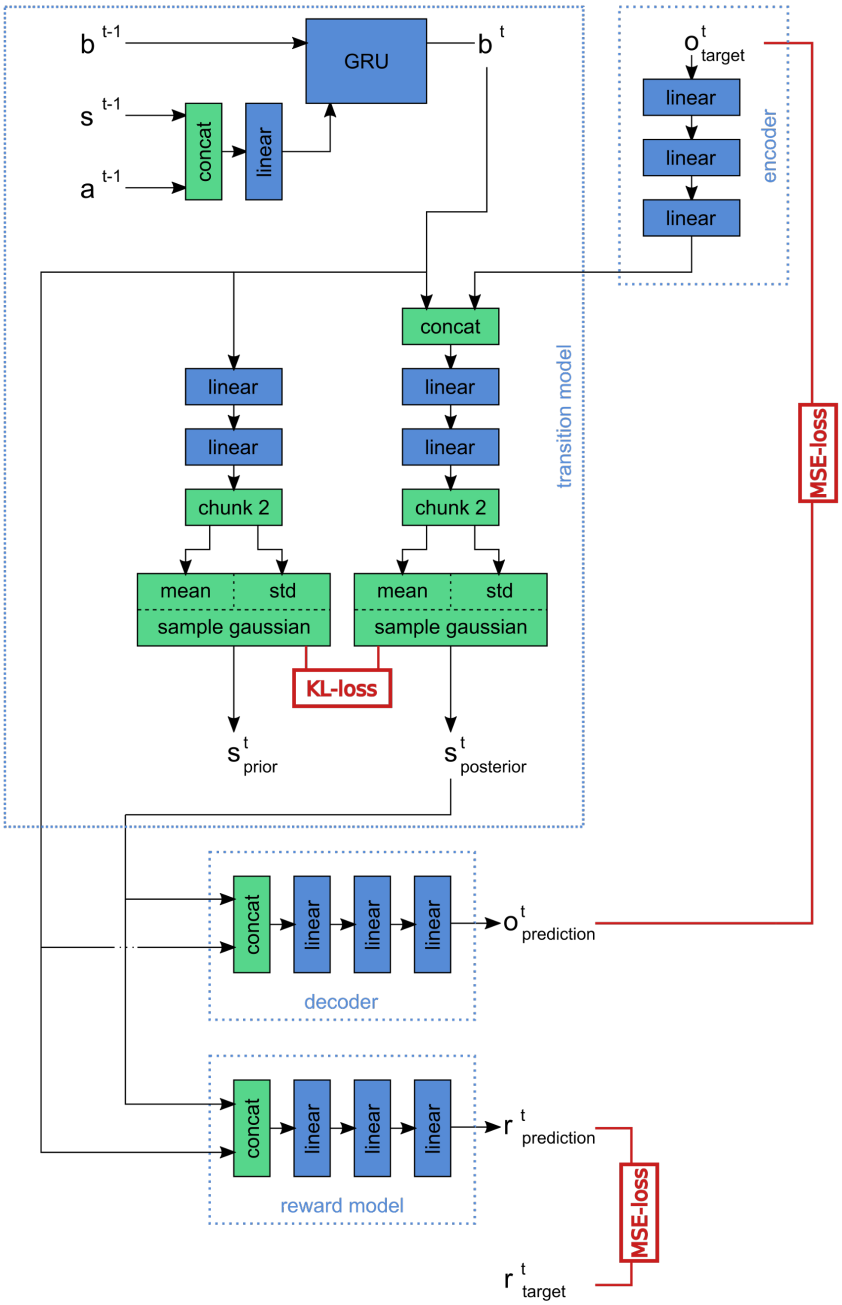


Figure 4: Neural network architecture for single-step prediction objective with PlaNet. Blue nodes denote parametric operations, green nodes denote non-parametric operations, and red nodes denote training objectives. Dense layers (linear) are followed by ReLU non-linearities [37], except for those leading into output variables and into the inputs for the gaussian sampling modules. These sampling modules use the reparameterization trick [30]; since standard deviations can not be negative, the 'std' inputs for these modules are first passed through a softplus method [18].

function, describing the relationship between latent states and observations. $\gamma \in [0, 1]$ is the discount factor, weighing the respective importance of long- and short-term rewards.

The world model, implemented as a deep neural network, functions as a learned simulation model of the environment. The purpose of the world model is to estimate fitness scores to optimize sequences of actions. Without loss of generality, we assume discrete-timestep, finite-length episodes so that $t \in [1, ..., T]$. At time t, given observations $o^{1:t}$ and actions $a^{1:t-1}$, the transition model predicts a posterior distribution for the latent state $(b, s_{post})^t$ (see Figure 2). Further given a sequence of actions $a^{t:T}$, the model predicts prior distributions over future latent states $(b, s_{prior})^{t+1:T}$, without receiving additional observations. In principle, the latent state of the environment is not measurable and can not be used as a training signal. So, given a latent state estimate $(b, s)^t$, the world model predicts the reward r^t yielded by the environment at that time. In this way, the model can predict the cumulative reward $r^{t+1:T}$ expected to be gained for any action sequence $a^{t:T}$ (see Figure 3).

In addition to predicting future rewards, the world model is trained to predict future observations. Although predictions of observations are not used at inference, training on observations forces the latent states to capture a richer representation of the environment and overall induces a richer training signal than the rewards alone. As shown in Figure 4, when given a latent state $(b, s)^{t-1}$ and an action a^{t-1} the world model produces the deterministic component b^t and a prior distribution over the stochastic component s^t_{prior} of the next latent state. When also provided with the observation o^t, the world model produces a posterior distribution over the stochastic component s^t_{post} of the next latent state informed on this observation. The posterior latent state $(b, s_{post})^t$ is fed into a multi-layer perceptron functioning as a decoder that outputs a reconstruction of the observation o^t. The squared error between the predicted reward $r^t_{predict}$ and the ground truth r^t_{target} (Reward Loss) is used as a training loss to optimize model weights. Additionally, the mean-squared error between the reconstructed observation $o^t_{predict}$ and the ground truth o^t_{target} (Observation Loss), as well as the KL-divergence between the prior s^t_{prior} and posterior s^t_{post} distributions of the stochastic state component (KL-divergence Loss) are used as training losses:

$$\mathcal{L}_{reward} = \frac{1}{T} \sum_{t=1}^{T} (r^t_{predict} - r^t_{target})^2 \tag{1}$$

$$\mathcal{L}_{observation} = \frac{1}{nT} \sum_{t=1}^{T} ||o^t_{predict} - o^t_{target}||_2^2, \; n = \textbf{\textit{dim}}\ o \tag{2}$$

$$\mathcal{L}_{KL} = \frac{1}{T} \sum_{t=1}^{T} KL(T(s^t_{prior}|b^{t-1}, s^{t-1}, a^{t-1}), T(s^t_{post}|b^{t-1}, s^{t-1}, a^{t-1}, o^t_{target})) \tag{3}$$

Assuming perfect one-step predictions, perfect multi-step predictions can be made with the recurrent world model to predict the cumulative reward for an action sequence. In practice, however, predictions are not perfect, and cumulative errors cause latent state prediction errors to grow exponentially with the length of rollouts. In order to train the world model to retain information and stimulate consistency over longer rollouts, an additional latent overshooting objective is used to optimize model weights. Given an action sequence $a^{0:T-1}$ and a sequence of observations $o^{1:T}$, posterior latent state distributions $s^{1:T}_{post}$ are generated following the procedure in Figure 4. These one-step predictions of the posterior distributions of latent states are used as training targets for the multi-step predictions of the prior distributions over those latent states (see Figure 5). The gradients for these posterior distributions are stopped to ensure that the

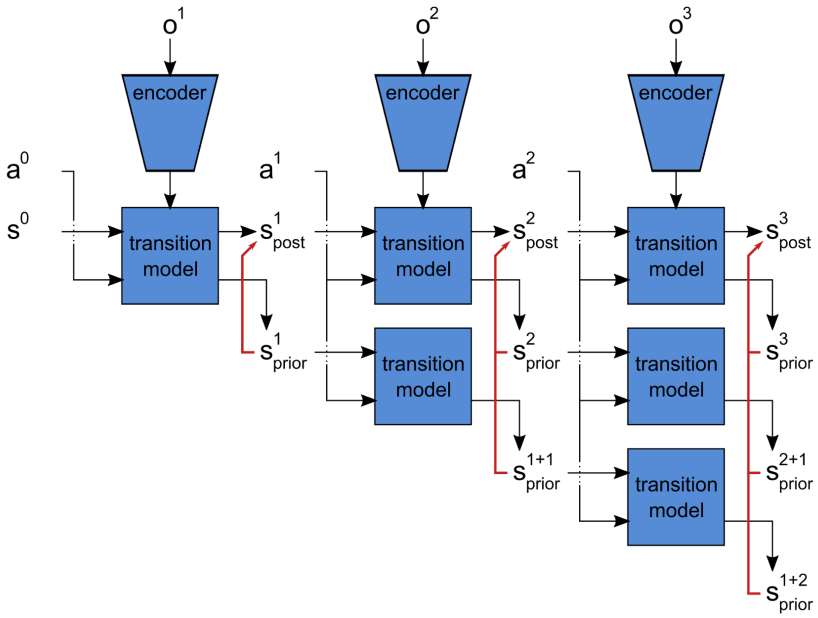


Figure 5: Latent overshooting - the deterministic component of the latent state vector is omitted for simplicity. Black arrows depict forward passes through the world model, red arrows depict KL-loss scores.

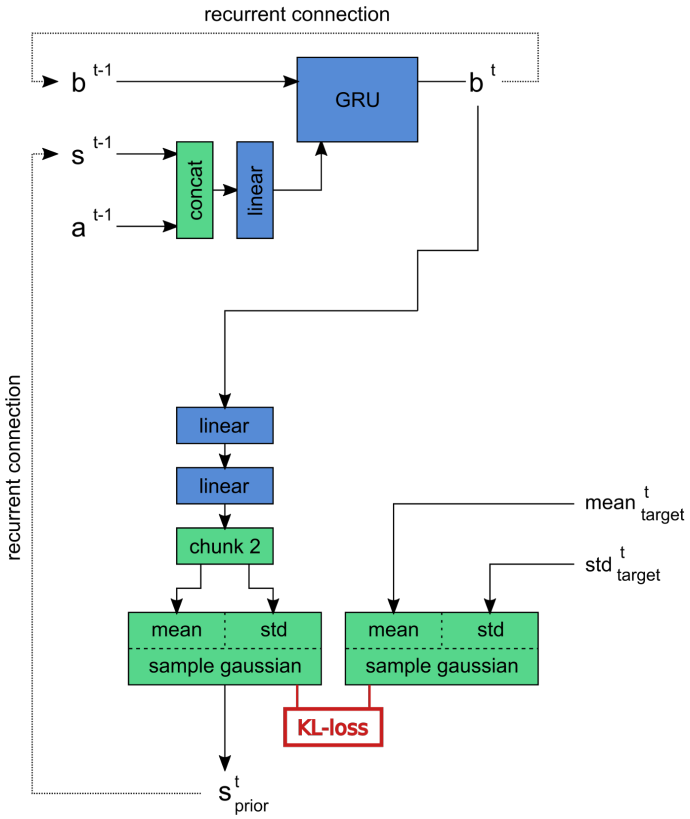


Figure 6: Rolling out the transition model for multiple steps over the prior loop without observations.

prior distributions converge to the posteriors, but not vice versa. For each $t \in [1, ..., T]$, given a chunk size D, b^t, s^t_{post}, and $a^{t:t+D}$; multi-step predictions of the latent state prior $s^{t+1:t+D+1}_{prior}$ are made by rolling out the recurrent prior loop of the transition model for D steps (see Figure 6). The overshooting loss is the KL-divergence between the predicted prior distributions s^{t+d}_{prior} and the previously computed target posterior distributions s^{t+d}_{post}.

$$\mathcal{L}_{OV} = \frac{1}{TD} \sum_{t=1}^{T} \sum_{d=1}^{D} KL(T(s^{t+d}_{prior} | b^t, s^t_{post}, a^{t:t+d-1}), T(s^{t+d}_{post} | b^{t+d}, o^{t+d})) \tag{4}$$

The training objective used for optimizing the world model is a simple sum of the reward, observation, KL-divergence, and overshooting losses:

$$\mathcal{L} = \mathcal{L}_{reward} + \mathcal{L}_{observation} + \mathcal{L}_{KL} + \mathcal{L}_{OV} \tag{5}$$

4 Policy Distillation

In the previous section, we have discussed how a deep neural network can be trained to function as a world model of an environment. In this section, we explore various techniques to distill effective policies from such a world model.

4.1 CEM Planner

The standard PlaNet architecture [21] deploys a CEM planner [35] to optimize an action sequence using the world model as a fitness function. At each timestep during rollouts, a population of action sequences $a^{t:t+d}$ of a fixed length up to a planning horizon d are sampled from a normal distribution, initially with mean 0 and standard deviation 1. The fitness of each of these action sequences is estimated by rolling out the transition model for d timesteps with that action sequence and summing over the predicted rewards $r^{t+1:t+d+1}$ resulting from the action sequence. After each optimization iteration, the mean and the standard deviation of the population are updated to the mean and the standard deviation of the top-k evaluated action sequences. After the last optimization iteration, the mean of the top-k evaluated action sequences is returned as an approximation of the optimal action sequence. Adopting its methodology for action selection from the model predictive control framework [6], only the first action of this optimal action sequence is executed. At the next timestep, the entire procedure is repeated.

A limitation of this method is the limited planning horizon. Conventional RL methods implicitly incorporate an infinite time horizon in their action-value predictions [49]. The influence of predicted future rewards slowly approaches zero for distant timesteps through the discounting factor. Since the CEM planner explicitly uses the predicted rewards for a fixed-length action sequence, the discounting factor is, in effect, 1 for future timesteps up to the planning horizon and 0 for timesteps after that.

A second limitation of the CEM planner is the computational intensity of on-policy optimization with this method. During each timestep, thousands of action sequences need to be rolled out and evaluated. On our hardware, an Nvidia Titan XP GPU (Nvidia, United States)[2], we could not reach a control frequency above 15 Hz. We posit that this is too slow for many real-world robotic control tasks.

[2] www.nvidia.com/en-us/titan/titan-xp/

4.2 Back-Propagation through Physics

Back-propagating reward scores through a differentiable physics engine has been shown to be an effective method for optimizing a policy [12]. Since a world model functions as a learnable, differentiable simulation model of the environment, we experiment with using the gradients in the world model between actions and predicted rewards to optimize a policy network.

In this gradient-based approach, we derive initial latent states from samples of the experience replay and roll out the world model for a number of timesteps up to a chunk size D. At each timestep, the deterministic and stochastic state vectors b^{t+d-1} and s^{t+d-1} are fed into a policy network that outputs an action a^{t+d-1}. This policy network has been implemented as a simple feed-forward neural network with two hidden layers of 200 neurons each. Then, b^{t+d-1}, s^{t+d-1}, and a^{t+d-1} are input into the transition model to predict the latent state on the next timestep and the reward yielded by the environment on that timestep. During this rollout, all gradients are stored. The predicted rewards with reversed sign $-r^{t:t+D}$, derived in this manner, are used as the training loss to optimize the weights of the policy network. The parameters of the world model are frozen so that only the policy network is trained with this objective.

Mixed results have been achieved in the past utilizing the gradients of trainable models that approximate environment dynamics [15]. Often the gradients of these learned models prove to be poor approximations of the ground truth model underlying the real system [13]. The learned models may display the same behavior under normal circumstances but develop internal dynamics that differ from the real system. Hence, there is no guarantee that optimizing policies using the gradients of the learned models leads to optimal policies in the real environment.

4.3 Imitation Learning

Finally, imitation learning [25] may form a viable approach to distilling effective policies from the world model. With this approach, the CEM planner is treated as an expert to train a simple feed-forward policy network with behavioral cloning [2]. This feed-forward network has the same architecture as the policy network used with our gradient-based approach: two hidden layers with 200 neurons each.

Ordinarily, imitation learning is used when pre-recorded data of usually a human expert is available and a policy network needs to be trained to perform the same task as the expert. In our case, the low control frequency of the CEM planner makes the expert incapable of generating real rollouts. Instead, the CEM planner can only generate approximately optimal actions off-policy. Policy rollouts can only be generated with the policy network and the CEM planner is used as an interactive expert that criticizes the actions taken by the policy network.

We initialize a policy network at random and collect experience by rolling out the policy for several seed episodes. We train the world model on the collected data samples. After training the world model on the collected data, we use the trained world model and CEM planner to generate optimal actions for the data points in the experience replay. These approximately optimal actions are then used as targets to train the policy network in a supervised manner. In essence, the policy network is trained to instinctively produce the same action as the CEM planner after several optimization iterations. At runtime, the policy network should be able to replace the CEM planner while running at a sufficient control frequency to interact with physical environments.

A notable difference with common applications of imitation learning is the fact that the expert is non-stationary: as more experience is collected the world model becomes a better approximation of the environment and thus the CEM planner starts producing

better approximately optimal actions. Therefore, the target actions for data points from earlier rollouts change as the world model learns.

The question remains how to incorporate newly collected data. The policy network could be trained only on the newly collected data. As the policy network improves, it will discover new areas of the environment, so it will primarily need to be trained on these new observations to progress further. However, when training the policy model in this manner, it will eventually forget what it has learned in earlier rollouts. For example, as the network learns how to balance the pole it will forget how to perform the swing up. We consider two methods to counter this phenomenon: aggregating data and aggregating policies. The aggregated data approach involves training a new policy model on all the previously collected experiences during each episode [42]. A disadvantage of this approach is the sheer amount of interaction with the CEM planner required. Since the model weights of the world model are updated at the start of each episode, new estimates of optimal actions need to be made for each data point in the experience replay before training the policy model. So, although the inference of the policy model during environment rollouts may be fast, the amount of computationally taxing interaction with the CEM planner grows quadratically with the number of episodes. The aggregated policy approach involves training a policy model only on the data newly collected during the previous episode. To maintain knowledge gained in earlier episodes, a linear interpolation is made between the model weights of the newly trained policy model and a copy of the old policy model. Due to the black-box nature of neural networks, there is no guarantee that such a linear interpolation between model weights maintains all the key functionalities of both networks.

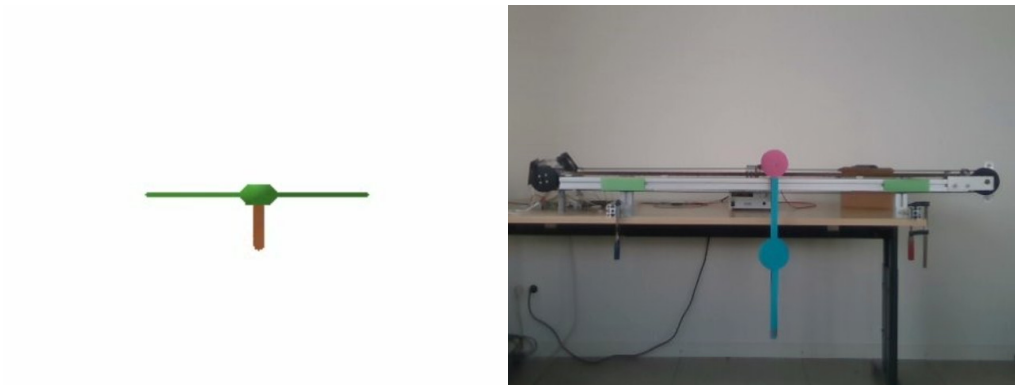
In conclusion, we propose to validate the standard PlaNet architecture, consisting of a world model that models transition dynamics of the environment and a CEM planner that optimizes action sequences with the world model at runtime. Additionally, we propose two adaptations to the PlaNet architecture to accommodate faster inference with the policy model. In both adaptations, the CEM planner is replaced with a simple feed-forward neural network; the methods differ in the manner in which the policy network is trained. Firstly, we propose to exploit the fact that the world model is fully differentiable to use the gradient between the actions generated by the policy model and the predicted reward scores as a loss score to optimize the model weights of the policy network. Secondly, we propose to train the policy model with imitation learning. To alleviate the common issue of forgetting in behavioral cloning, we examine two imitation learning frameworks: aggregated data and aggregated policy.

5 Experimental Setup

We aim to solve an inverted pendulum swing-up task on an in vivo robotic setup. In this task, a pendulum is suspended on a swivel from a cart, which can be slid back and forth along a rail (see Figure 7). The goal of the task is to manipulate the cart in such a manner that the pendulum is swung up to its vertical position and kept balanced in that unstable equilibrium for a period of time. The inverted pendulum swing-up task incorporates fundamentals for robotic balancing [47, 53]. We evaluate our models both on an in-vivo robotic setup as well as on a simulated version built on the PyBullet physics engine [10].

5.1 In-Vivo Setup

Our in-vivo setup consists of a cart that can slide along a rail. The cart is attached to a notched rubber belt, strung between two cogs. One of these cogs is powered by a Maxon



(a) (b)

Figure 7: (a) Simulated inverted pendulum setup and (b) In-vivo inverted pendulum setup.

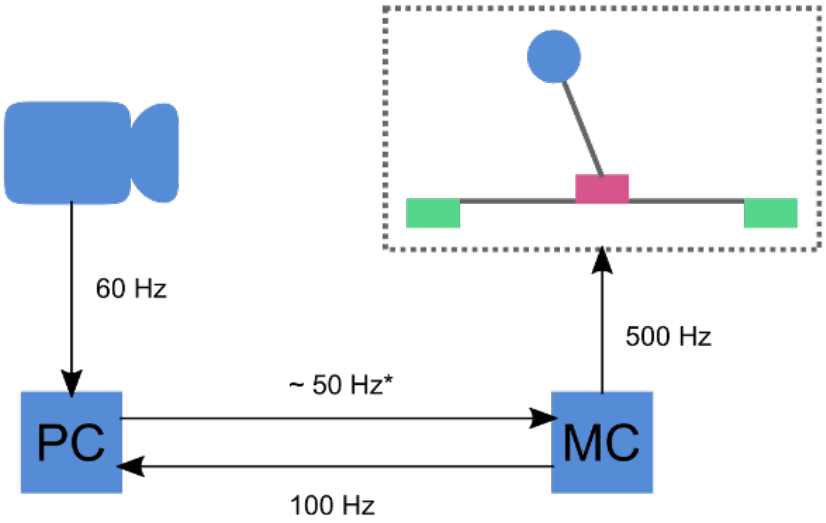


Figure 8: Schematic depiction of the communication loops in the setup. The server (PC) receives frames from the camera at 60 Hz and information on the speed of the motor from the microcontroller (MC) at 100 Hz. The PC sends actions to the microcontroller at approximately 50 Hz, depending on the inference speed of the policy model. The microcontroller accelerates or decelerates the motor to the action specified speed at 500 Hz.

motor (Maxon, Switzerland)[3] controlled by a Dimension Engineering sabertooth2x25 motor controller (Dimension Engineering, United States)[4]. Digital control commands are sent to the motor controller via a dwenguino microcontroller platform (Dwengo, Belgium)[5]. Two buttons have been placed at the ends of the rail, which are connected to a hard-kill switch on the motor controller. If the cart comes too close to the edge of the rail, one of these buttons is pressed, and the entire system shuts down to prevent any damage to the hardware.

At both ends of the rail, a green sheet of paper has been attached to mark the boundaries of the region where the card is allowed to move. On the cart and the pendulum respectively, a magenta piece of paper and a blue piece of paper has been attached to mark their positions. A realsense camera [29] has been used to record the setup during runtime. The location of the abovementioned keypoints is extracted with OpenCV [4]. With this setup, the observable state of the system is extracted at each time step as a 5-tuple: (1) the x-coordinate of the cart, ranging from -1 (all the way left) to 1 (all the way right); (2) the x-component of the angle of the pendulum, projected on the unit-circle; (3) the y-component of the angle of the pendulum, projected on the unit-circle - the angle is split into two components to prevent discontinuities. (4) the velocity of the cart, scaled between -1 and 1; (5) the angular velocity of the pendulum, scaled and clipped between -1 and 1.

In order to ensure minimal latency, a multi-threaded approach has been adopted for the processing of camera images. The camera framerate has been capped at 60 fps. The camera thread continuously receives frames from the camera and writes these to a shared memory buffer. An image processing thread continuously reads the most recently added frame from the shared memory and extracts the keypoints using HSV- and morphological filters. The observable state is calculated from these key points and stored in another shared memory array. During rollouts, at each timestep, the main thread reads the most recently computed observable state from the shared memory and uses this observation to choose an action with its policy model. This action is sent to the dwenguino controlling the motor controller via serial commands. Figure 8 shows a schematic depiction of the setup along with the rates of communication between its components.

5.2 Quality Metrics

Two metrics are of interest: performance score and control frequency. Firstly, we have measured how well the agent is able to solve the task by taking the cumulative rewards gathered over a trial of 1000 discrete timesteps. This measure has been applied both in the simulated environment and the in-vivo setup. In the real environment, the continuous time frame has been abstracted away, instead rewards have been measured in discrete time step samples of the environment. Note that this means that rollouts on the real environment with a model with a lower control frequency results in a longer rollout in time.

The reward at each timestep has been calculated by taking the y-component of the projection of the angle of the pendulum on the unit circle. Hence, the reward ranges from -1 when the pendulum is pointing vertically downwards to 1 when the pendulum is in its upright position. Over a full trial, the minimum achievable score is -1000. The maximum achievable score is 1000 minus the minimum amount of reward lost to swing up the pendulum from its start position.

In addition to overall performance, the control frequency of our policy models has

[3] www.maxongroup.com

[4] www.dimensionengineering.com/products/sabertooth2x25

[5] www.dwengo.org

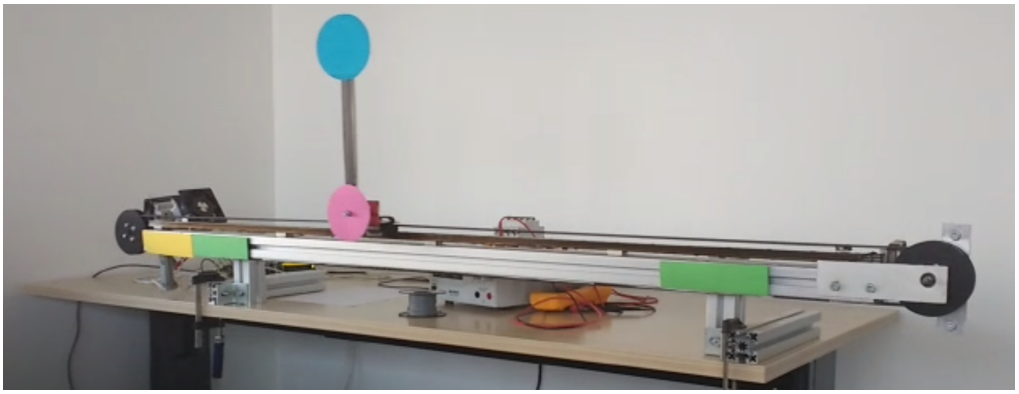


Figure 9: Our in-vivo setup in action with a classic controller. Note that the current setup has a longer pole than depicted in this figure. A full demonstration is available at https://youtu.be/W-pXJfK-4dw

been measured. The control frequency is the number of times that the agent can compute an action and send it to the setup. In order to successfully interact with a setup on a time-dependent task, a sufficiently high control frequency is required.

On both the in-vivo setup and the simulated domain, 5 initial seed episodes are collected and the model is trained for an additional 971 episodes. Performance is measured every 25 episodes.

6 Experimental Results & Discussion

In this section, we discuss our experimental results. In section 5.1 we start with a validation of the feasibility of our in-vivo setup by showing that it can be solved with a classical controller. In section 5.2 we validate our implementation of the standard PlaNet architecture on our simulated environment. In section 5.3 we test the standard PlaNet architecture on the in-vivo setup and find that inference is insufficiently fast to solve the balancing component of the task. In section 5.4 we discuss our alternate policy distillation methods: gradient-based optimization and imitation learning, evaluated on the simulated environment. In section 5.5 we evaluate the tolerance of the CEM planner and gradient-based policy models to two major contributors to the reality gap between our simulated and in-vivo environments. *Due to the corona pandemic, we have been unable to collect quantitative results on the in-vivo setup for many of our experiments.*

6.1 Validation of In-Vivo Setup

To validate the feasibility of our setup and to set a baseline for our models, we have implemented a classical controller for the robotic setup (see Figure 9). We have found that due to the latency induced by processing the camera feed it is not possible to keep the pole in its stable equilibrium with the straightforward approach of a bang-bang controller [46], so a more fine-tuned control signal is needed. The controller uses energy-based control for the swing-up and a fine-tuned control function when the pole is near its unstable equilibrium, designed to remove angular velocity from the pole and keep it balanced upright. In order to keep the cart away from the edges of the rail while balancing the pendulum, the target angle is dynamically adjusted slightly towards the center of the rail depending on the position of the cart, so that the cart is naturally steered back to the middle of the rail. *We would report a more quantitative analysis*

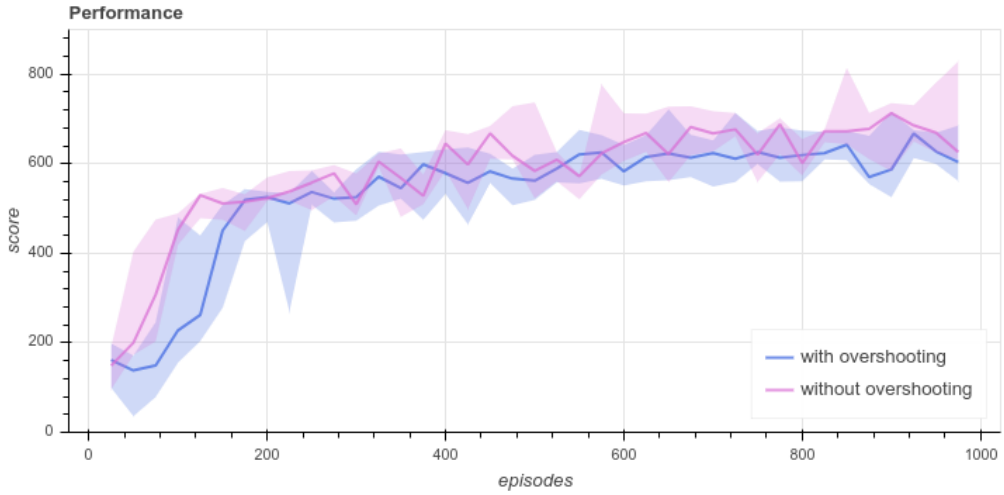


Figure 10: Performance of vanilla PlaNet, with and without latent overshooting in the training objective. The blue and magenta lines depict the median performance of the model, respectively with and without latent overshooting in the training objective. The semi-transparent areas of the same colors represent the first and third quartiles. Demonstrations are available at https://youtu.be/JWhUbW8bJUg and https://youtu.be/tsyT3bs-8_g

of the performance of our controller as well as a demonstration of the current setup in action, but due to the corona pandemic we do not have access to our setup

6.2 Validation of Standard PlaNet Architecture

In order to validate our re-implementation of PlaNet, we evaluate the standard PlaNet architecture in the simulated environment. The standard PlaNet architecture consists of the trainable world model and the CEM planner, which generates a policy through planning in latent space. The control frequency of 15 Hz of the CEM planner is not high enough for the fine-tuned control required for balancing on the in-vivo setup. This is not an issue for interaction with the simulated environment, which can be paused at each timestep to allow the CEM planner to compute an action. Figure 10 shows that our standard PlaNet implementation converges within 1000 training episodes. It also shows that the model trained without latent overshooting converges to a slightly better performance for this instantiation of the environment. It appears that one-step predictions are sufficiently accurate for multi-step rollouts so that the added complexity of the latent overshooting objective does not benefit training.

6.3 Standard PlaNet Architecture for In-Vivo Robotic Control

The computational complexity of the CEM planner leads to a relatively low control frequency of the pendulum. To evaluate whether this hinders balancing, we deploy the standard PlaNet architecture on the in-vivo setup. At most, we have managed to get a control frequency of approximately 15 Hz using a single Nvidia Titan XP GPU (Nvidia, United States). We have observed that the control frequency of the CEM planner is insufficient to learn a stable policy for the balancing task.

We have attempted to improve the control frequency by splitting the population of the CEM planner over multiple GPUs, but we have found that this only improves time-

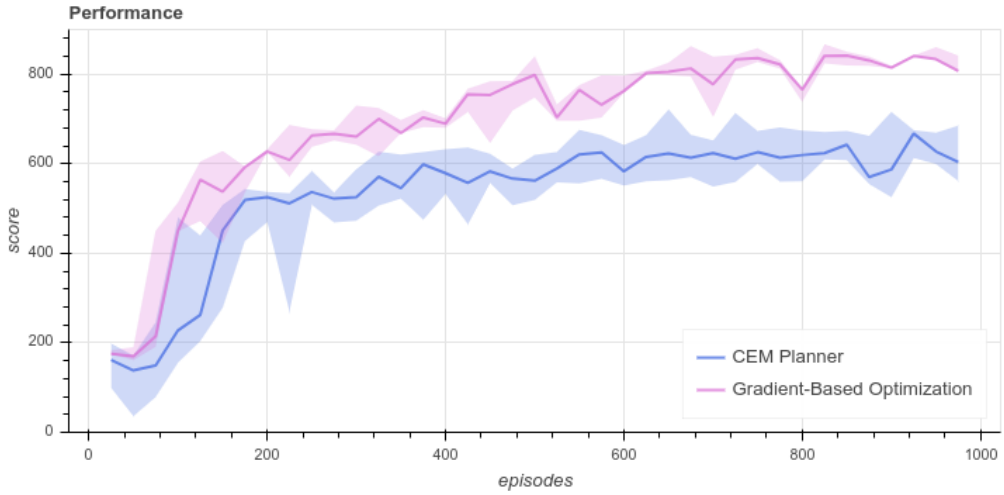


Figure 11: A comparison of the performance of the standard PlaNet architecture and our gradient-based approach. A demonstration of an agent trained with back-propagation through physics is available at https://youtu.be/4vouAX3Oc9Q

efficiency when working with much larger batch sizes. Since planning with the CEM planner is an inherently iterative process, we have not been able to gain performance improvements with data parallelism. In fact, splitting the population of the CEM planner over multiple GPUs causes slow-down due to the overhead of loading data to and from the CPU. So, although data parallelism can improve overall training time, it can not improve control frequency at inference on our setup. *Due to the corona pandemic, we are unable to gather quantitative results on our robotic setup.*

6.4 Gradient-Based Optimization & Imitation Learning

The policy network trained with the gradient-based technique reaches a control frequency of 40 Hz on our in-vivo setup. As we have demonstrated with our classic controller, this is above the minimum bound on control frequency required to solve the task. We have evaluated the performance of our gradient-based approach in the simulated environment. Hyperparameters for both methods have been optimized through trial-and-error. The only notable difference between optimal hyperparameter settings for the CEM planner and gradient-based approach is the action-repeat, which has been set to 2 for the CEM planner and to 4 for the gradient-based approach. Figure 11 shows that the gradient-based approach converges to a better performance than the standard PlaNet architecture. Apparently, the gradients in the learned world model are sufficiently similar to the real gradients of the physics of the simulator that gradient-based optimization of the policy network leads to good policies. Under some circumstances, it may be that back-propagating the rich gradient of the predicted reward score through the world model is a better approach than the CEM planner; not only for computational efficiency but also for performance.

We have observed an issue in both the standard PlaNet architecture and our gradient-based approach where the policy models can have trouble keeping the cart away from the edges of the rail while balancing the pendulum. We believe this to be a consequence of the finite planning horizon. Since the planning horizon is limited, the policy models have no intrinsic motivation to keep the cart in the middle of the rail or to slow the cart when it has a high velocity. The possibility of bumping into the end of the rail

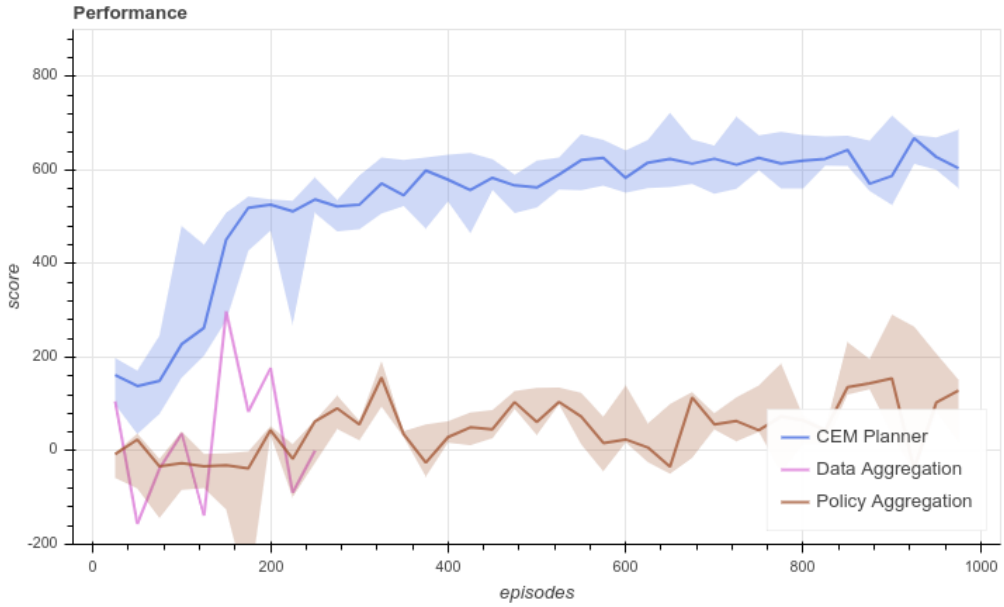


Figure 12: A comparison of the standard PlaNet architecture and our imitation learning approach. Due to the significantly higher time complexity of the data aggregation approach, a single trial with this method is reported over 250 episodes. This amount of training has required over a week on our hardware. A demonstration of the imitation learning approach with data aggregation is available at https://youtu.be/JkPNWcdwag8. Notice how the agent apparently makes no effort to balance the pendulum at 0:12. A demonstration of the imitation learning approach with policy aggregation is available at https://youtu.be/5KQUhw3z86E. Notice the lack of fine-tuned control, e.g. at 0:07.

only comes within the planning horizon once the cart is already near the edge. At that point, the agent will try to slow down the cart, but it may already be too late to do so effectively. Longer planning horizons, however, result in compounding errors and in insufficient planning due to the exponential growth of the number of candidates in deeper search spaces [57].

The policy network trained with imitation learning has an identical architecture to the policy network trained with the gradient-based approach. So, the policy networks trained with imitation learning also reach a control frequency of 40 Hz on our in-vivo setup. We have evaluated the performance of our imitation learning approaches in the simulation environment. Imitation learning with aggregated data has been found to be impractically slow due to the amount of computationally expensive interaction with the CEM planner growing quadratically with the number of episodes. For this reason, only one run with the data aggregation strategy has been reported. The performance of the imitation learning approach with data aggregation after 250 episodes is significantly worse than the performance of the standard PlaNet architecture and the backpropagation-through-physics approach. Although the time-efficiency of the policy aggregation method is much better than the data aggregation approach, neither approach has been capable of finding a successful balancing strategy in our experiments. Both approaches learn to swing the pendulum around, which is better than doing nothing, but neither are fit to learn the fine-tuned manipulations required for balancing (see Figure 12).

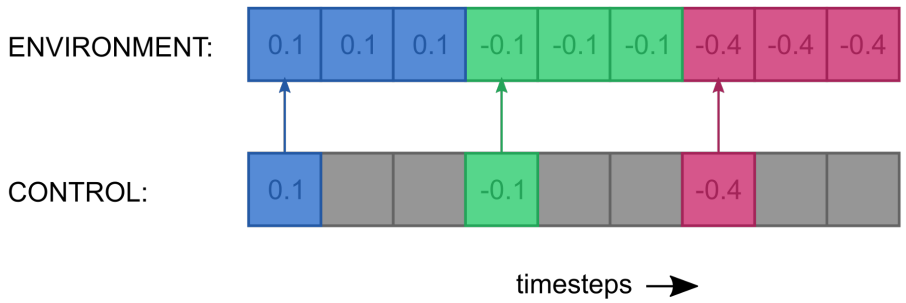


Figure 13: The timing between action selection and execution for a simulated environment with an action repeat of 3 timesteps.

A likely explanation why imitation learning performs poorly is due to the non-stationary nature of the expert. The CEM planner finds an optimal action given the world model. Since the world model is trained in parallel to the policy model, the actions that the CEM planner finds, are only approximations of optimal actions in the real environment, subject to the inaccuracies of the world model. In conjunction, the policy network makes an approximation of the actions that are be produced by the CEM planner, hence resulting in even less accurate approximations of optimal actions in the environment. So, due to the mutual dependence of the world model and the policy network, it is plausible that the world model never learns that the pendulum can be balanced and the policy network never learns to explore behaviors to that end.

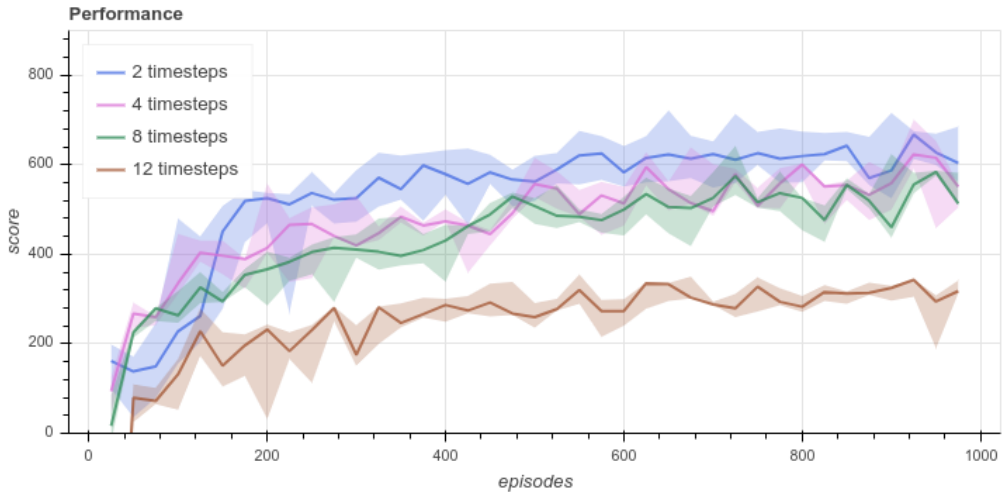
6.5 Control Frequency & Latency

Due to the corona pandemic we are currently unable to gather quantitative results on our robotic setup using our back-propagation through physics approach. Instead, we simulated several of the added difficulties interacting with a real environment in our simulation model.

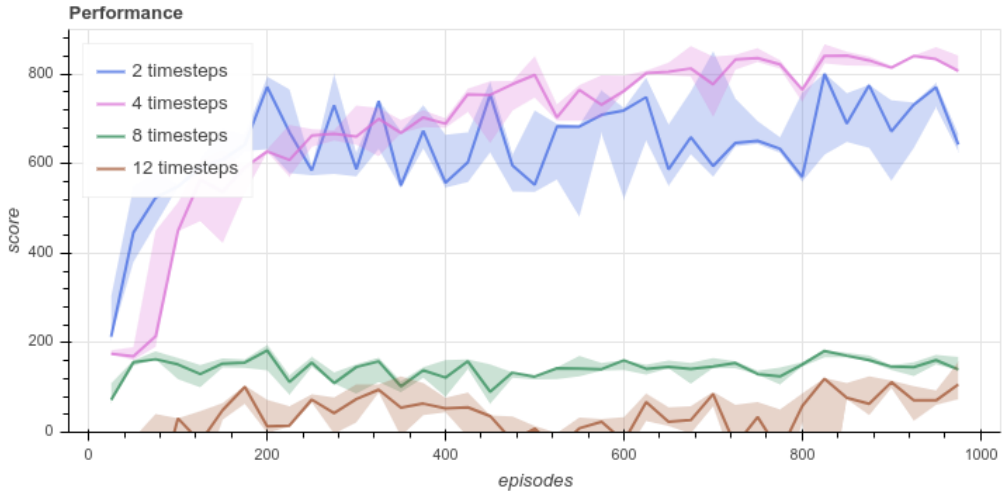
Our simulation environment is a distilled version of the physical domain, wherein many of the key challenges of the real world have been abstracted away. Two major differences between the simulated and physical domains are limited control frequency and latency. We evaluate the influence of these two factors on the performance of the CEM planner and our gradient-based approach to gain insight under which circumstances our proposed methods could be viable for the physical domain.

The control frequency is the number of times per second that the agent can compute an action and send it to the setup. In the simulation environment with discrete timesteps, this is analogous to action-repeat, the number of timesteps that an action is repeated before the agent is queried for a new action. In order to gain insight into the potential effect of a lower control frequency on model performance, we artificially raise the action-repeat in the simulation environment (see Figure 13).

Figure 14 shows that our gradient-based approach performs optimally at a lower control frequency than the CEM planner. However, as the control frequency is lowered further, the performance of our gradient-based approach declines faster than the performance of the CEM planner. Since it is easier to artificially lower the permitted control frequency on an in-vivo setup than to raise it, these results are promising in that they show a reasonable margin in the permitted control frequency within which the gradient-based approach should be able to achieve optimal performance in a physical environment. However, they also show that a definite lower bound on the permitted control frequency exists under which our gradient-based method does not perform. Ultimately, the viability of the proposed methods depends on the particular setup in question



(a)



(b)

Figure 14: A comparison of the influence of control frequency on performance: standard PlaNet (a) and our back-propagation through physics approach (b).

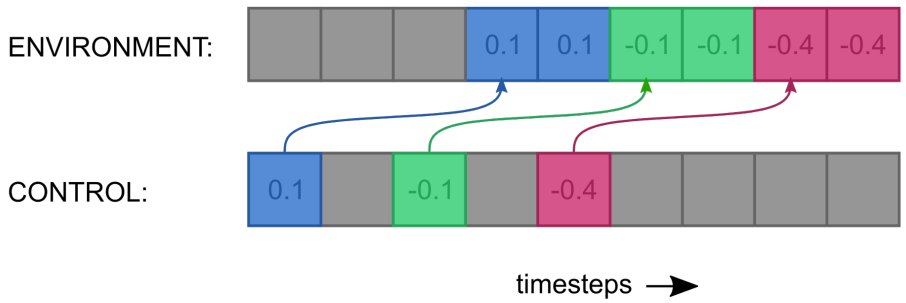


Figure 15: The timing between action selection and execution for a simulated environment with an action repeat of 2 timesteps and a latency of 3 timesteps.

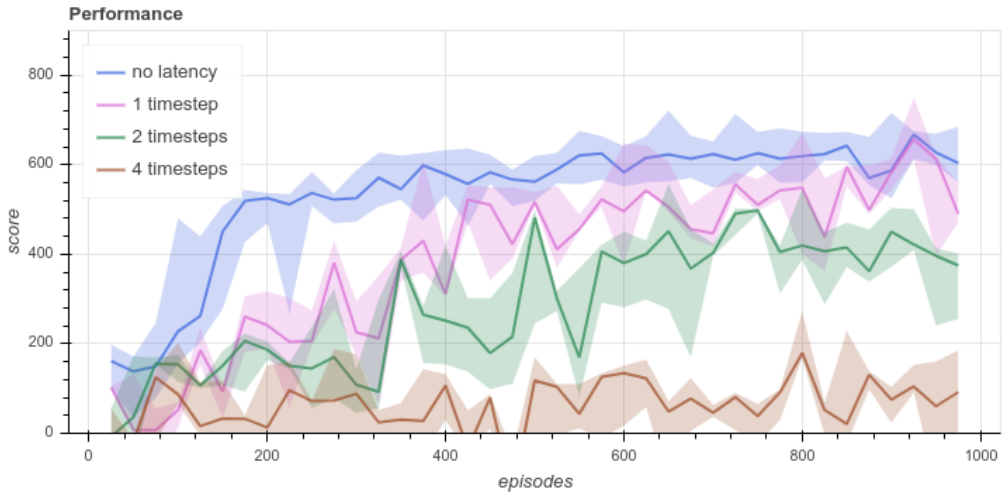
and will require extensive hyperparameter-tuning.

Latency is the time that passes after the camera captures a frame, until this frame is processed, an action is selected and this action is translated into a physical change in motor speed. While the image data is being processed and the agent computes an action with its policy model, the environment continues to change. So, by the time a selected action is executed on the setup, the environment has already progressed several milliseconds. Despite our efforts to make action selection as swift as possible, we have been unable to reduce latency below approximately 25 milliseconds with our hardware. The agent needs to learn to anticipate this time gap. In contrast, the simulation environment works in discrete time steps. After each timestep, the environment is paused until a new action is presented, allowing the agent to take an unrealistic amount of time to compute an action.

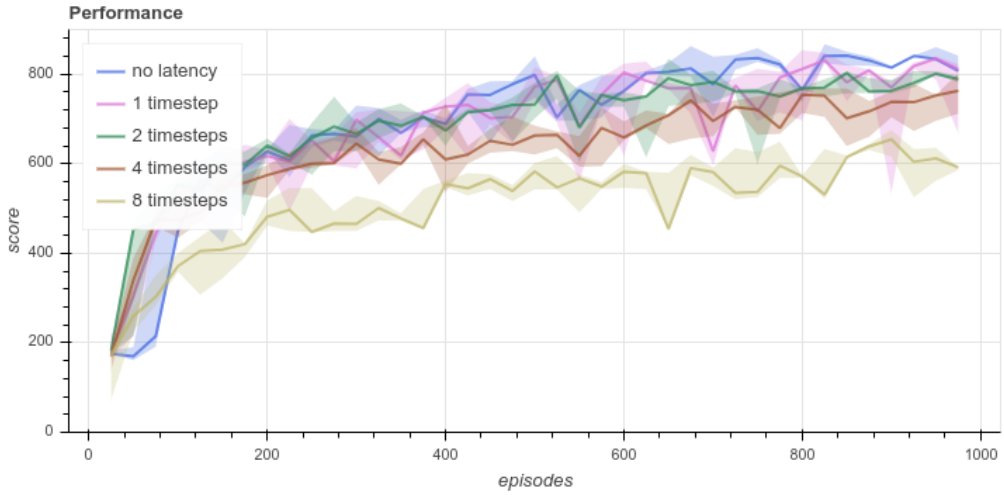
To gain insight into the potential effect of this latency on the performance of our models, we have added artificial latency to the simulation environment (see Figure 15). A buffer of actions is initialized with zeros. At each timestep, the agent sends an action to the environment. This action is appended to the buffer. Then the first action in the buffer is executed in the simulation model and removed from the buffer. For a buffer of length n, this means that the action chosen by the agent at time t, will be executed in the environment at time $t + n$. Hence, we simply manipulate the amount of latency by changing the length of the buffer; if we use a buffer of length 0, this is analogous to the environment without latency.

Figure 16 shows that for both the standard PlaNet architecture as well as our gradient-based approach performance decreases when the amount of latency is increased. However, our gradient-based approach is much more tolerant of latency than the CEM planner; even with 4 timesteps of latency, the gradient-based approach performs better than the CEM planner without latency. This suggests that our gradient-based approach is more fit for physical systems with latency.

A likely explanation for this difference in performance is the fact that the optimal control frequency for our gradient-based approach is lower than the optimal control frequency for the CEM planner. As an additional effect, this means that the total number of timesteps in the environment (i.e., not incorporating action-repeat) that fall within the reward horizon of the policy model is higher for the gradient-based approach than for the CEM planner. Specifically, with an action repeat of 2 timesteps and a planning horizon of 12, the CEM planner has a reward horizon of 24 timesteps; with an action repeat of 4 timesteps and a chunk size of 12, the gradient-based approach has a reward horizon of 48 timesteps. Hence, the same amount of latency proportionally reduces the reward horizon of the CEM planner by more than the reward horizon of the gradient-based approach.



(a)



(b)

Figure 16: A comparison of the influence of latency on the performance of the standard PlaNet architecture (a) and our gradient-based approach (b).

7 Conclusion & Future Work

In this paper, we have applied PlaNet on an in-vivo inverted pendulum swing-up task. We have found that the standard PlaNet architecture can solve the inverted pendulum swing-up task in simulation but not on the real setup. We have shown that this is due to the CEM planner, which after speed optimizations, is not able to reach a sufficiently high control frequency to control the pendulum in real-time.

To make PlaNet deployable on physical robotic tasks with real-time constraints, we have proposed two alternative parametric policy models using the PlaNet world model: gradient-based optimization and imitation learning. Both of these models can reach a control frequency of 40 Hz. As we have shown with our hand-crafted controller, this is high enough for solving the task. However, our experiments in the simulated environment have shown that only the gradient-based approach is a promising alternative, as the imitation learning approach is not capable of solving the task. The gradient-based approach achieves a higher performance than the CEM planner and behaves optimally at a lower control frequency. These results are promising for the application of the gradient-based approach on the real setup, since artificially lowering the control frequency is much easier than raising it.

Realizing that the simulated environment is an idealized representation of the real setup and that approaches that work in simulation are not guaranteed to work in the real world, we have analyzed the impact of two major challenges in the in-vivo setup that are by default not modeled in simulation: latency and control frequency. Our experiments have shown that our gradient-based approach is more tolerant of latency than the CEM planner and that the gradient-based approach performs best at a lower control frequency than the CEM planner. However, if the control frequency permitted by the system decreases further, the performance of our gradient-based approach declines more rapidly than the performance of the CEM planner. Future work will involve evaluating the performance of the gradient-based approach on our in-vivo setup.

Our findings with the gradient-based optimization approach are unexpected. Due to model-bias, optimizing a policy model with this method often leads to the policy model exploiting deficiencies in the world model rather than solving the task. The explicit distinction between deterministic and stochastic transition dynamics in the PlaNet world model likely alleviates this effect sufficiently or the task environment is sufficiently simple that the policy model does learn useful behaviors. Future research should be directed towards gaining a deeper understanding of the gap between the gradients of the physics behind real-world systems and the gradients in parametric world models, trained on those systems. Although it is impractical to compute the gradients of real-world systems, physics engines, such as MuJoCo [52], can compute the analytic gradient between actions and rewards. This can be used for a detailed analysis of the relationship between the architecture of the world model and model-bias.

Finally, future work should be directed towards evaluating how well our proposed gradient-based method generalizes to more complex domains, involving more degrees of freedom, rigid-body collision, and soft-body or fluid dynamics. Additionally, the potential for integration of trained and hand-crafted dynamics models should be evaluated. This could enable the addition of prior knowledge to world models, allowing model optimization to focus on components of the environment that are harder to define by hand. Simultaneously, this could enable the implementation of trainable modules in otherwise hand-crafted systems for differentiable physics engines, so that system components that are challenging to model in a differentiable manner can be replaced with inherently differentiable world models that approximate the dynamics of the component.

References

[1] Openai: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *Int. J. Rob. Res.*, 39(1):3–20, January 2020.

[2] M Bain and C Sammut. A framework for behavioural cloning. *Machine Intelligence 15*, 1995.

[3] Konstantinos Bousmalis, Nathan Silberman, David Dohan, Dumitru Erhan, and Dilip Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3722–3731. openaccess.thecvf.com, 2017.

[4] Gary Bradski and Adrian Kaehler. OpenCV. *Dr. Dobb's journal of software tools*, 3, 2000.

[5] Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. Sample-Efficient reinforcement learning with stochastic ensemble value expansion. In S Bengio, H Wallach, H Larochelle, K Grauman, N Cesa-Bianchi, and R Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 8224–8234. Curran Associates, Inc., 2018.

[6] Eduardo F Camacho and Carlos Bordons Alba. *Model Predictive Control*. Springer Science & Business Media, January 2013.

[7] Paul Christiano, Zain Shah, Igor Mordatch, Jonas Schneider, Trevor Blackwell, Joshua Tobin, Pieter Abbeel, and Wojciech Zaremba. Transfer from simulation to real world through learning deep inverse dynamics model. October 2016.

[8] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In S Bengio, H Wallach, H Larochelle, K Grauman, N Cesa-Bianchi, and R Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 4754–4765. Curran Associates, Inc., 2018.

[9] Ignasi Clavera, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, and Pieter Abbeel. Model-based reinforcement learning via meta-policy optimization. *arXiv preprint arXiv:1809. 05214*, 2018.

[10] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. *GitHub repository*, 2016.

[11] Filipe de Avila Belbute-Peres, Kevin Smith, Kelsey Allen, Josh Tenenbaum, and J Zico Kolter. End-to-End differentiable physics for learning and control. In S Bengio, H Wallach, H Larochelle, K Grauman, N Cesa-Bianchi, and R Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 7178–7189. Curran Associates, Inc., 2018.

[12] Jonas Degrave, Michiel Hermans, Joni Dambre, and Francis Wyffels. A differentiable physics engine for deep learning in robotics. *Front. Neurorobot.*, 13:6, March 2019.

[13] Marc Deisenroth and Carl E Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

[14] Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. Challenges of Real-World reinforcement learning. April 2019.

[15] Yarin Gal, Rowan McAllister, and Carl Edward Rasmussen. Improving pilco with bayesian neural network dynamics models. In *Data-Efficient Machine Learning Workshop, ICML*, volume 4, page 34, 2016.

[16] Yaroslav Ganin, Tejas Kulkarni, Igor Babuschkin, S M Ali Eslami, and Oriol Vinyals. Synthesizing programs for images using reinforced adversarial learning. April 2018.

[17] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *J. Mach. Learn. Res.*, 17(1):2096–2030, January 2016.

[18] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

[19] Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard L Lewis, and Xiaoshi Wang. Deep learning for Real-Time atari game play using offline Monte-Carlo tree search planning. In Z Ghahramani, M Welling, C Cortes, N D Lawrence, and K Q Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3338–3346. Curran Associates, Inc., 2014.

[20] David Ha and Jürgen Schmidhuber. World models. March 2018.

[21] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. November 2018.

[22] Mikael Henaff, William F Whitney, and Yann LeCun. Model-Based planning with discrete and continuous actions. May 2017.

[23] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[24] Zhewei Huang, Wen Heng, and Shuchang Zhou. Learning to paint with model-based deep reinforcement learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 8709–8718. openaccess.thecvf.com, 2019.

[25] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Comput. Surv.*, 50(2):1–35, April 2017.

[26] Stephen James, Andrew J Davison, and Edward Johns. Transferring End-to-End visuomotor control from simulation to real world for a Multi-Stage task. July 2017.

[27] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model-Based reinforcement learning for atari. March 2019.

[28] Ken Kansky, Tom Silver, David A Mély, Mohamed Eldawy, Miguel Lázaro-Gredilla, Xinghua Lou, Nimrod Dorfman, Szymon Sidor, Scott Phoenix, and Dileep George. Schema networks: Zero-shot transfer with a generative causal model of intuitive physics. June 2017.

[29] Leonid Keselman, John Iselin Woodfill, Anders Grunnet-Jepsen, and Achintya Bhowmik. Intel realsense stereoscopic depth cameras. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–10, 2017.

[30] Diederik P Kingma and Max Welling. Auto-Encoding variational bayes. December 2013.

[31] Scott Kuindersma, Robin Deits, Maurice Fallon, Andrés Valenzuela, Hongkai Dai, Frank Permenter, Twan Koolen, Pat Marion, and Russ Tedrake. Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot. *Auton. Robots*, 40(3):429–455, 2016.

[32] Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-Ensemble Trust-Region policy optimization. February 2018.

[33] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. January 2016.

[34] Yuping Luo, Huazhe Xu, Yuanzhi Li, Yuandong Tian, Trevor Darrell, and Tengyu Ma. Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees. July 2018.

[35] S Mannor, R Y Rubinstein, and Y Gat. The cross entropy method for fast policy search. *of the 20th International Conference on ...*, 2003.

[36] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[37] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[38] C Berner OpenAI, Greg Brockman, Brooke Chan, Vicki Cheung, P Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, and Others. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[39] Paolo Pagliuca and Stefano Nolfi. Robust optimization through neuroevolution. *PLoS One*, 14(3):e0213193, March 2019.

[40] Ivaylo Popov, Nicolas Heess, Timothy Lillicrap, Roland Hafner, Gabriel Barth-Maron, Matej Vecerik, Thomas Lampe, Yuval Tassa, Tom Erez, and Martin Riedmiller. Data-efficient deep reinforcement learning for dexterous manipulation. April 2017.

[41] Anil V Rao. A survey of numerical methods for optimal control. *Adv. Astronaut. Sci.*, 135(1):497–528, 2009.

[42] Stéphane Ross, Geoffrey J Gordon, and J Andrew Bagnell. No-Regret reductions for imitation learning and structured prediction. In *In AISTATS*. Citeseer, 2011.

[43] Fereshteh Sadeghi and Sergey Levine. CAD2RL: Real Single-Image flight without a single real image. November 2016.

[44] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, Multi-Agent, reinforcement learning for autonomous driving. October 2016.

[45] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Joshua Susskind, Wenda Wang, and Russell Webb. Learning from simulated and unsupervised images through adversarial training. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2107–2116. openaccess.thecvf.com, 2017.

[46] L M Sonneborn and F S Van Vleck. The Bang-Bang principle for linear control systems. *Journal of the Society for Industrial and Applied Mathematics Series A Control*, 2(2):151–159, January 1964.

[47] M Stilman, J Olson, and W Gloss. Golem krang: Dynamically stable humanoid robot for mobile manipulation. In *2010 IEEE International Conference on Robotics and Automation*, pages 3304–3309, May 2010.

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

[49] Richard S Sutton, Andrew G Barto, and Others. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[50] Y Tassa, T Erez, and E Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4906–4913, October 2012.

[51] J Tobin, R Fong, A Ray, J Schneider, W Zaremba, and P Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30. ieeexplore.ieee.org, September 2017.

[52] E Todorov, T Erez, and Y Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, October 2012.

[53] K M Tsui, M Desai, H A Yanco, and C Uhlik. Exploring use cases for telepresence robots. In *2011 6th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 11–18, March 2011.

[54] Eric Tzeng, Coline Devin, Judy Hoffman, Chelsea Finn, Pieter Abbeel, Sergey Levine, Kate Saenko, and Trevor Darrell. Adapting deep visuomotor representations with weak pairwise constraints. In Ken Goldberg, Pieter Abbeel, Kostas Bekris, and Lauren Miller, editors, *Algorithmic Foundations of Robotics XII: Proceedings of the Twelfth Workshop on the Algorithmic Foundations of Robotics*, pages 688–703. Springer International Publishing, Cham, 2020.

[55] Alexander Vandesompele, Gabriel Urbain, Hossain Mahmud, Francis Wyffels, and Joni Dambre. Body randomization reduces the Sim-to-Real gap for compliant quadruped locomotion. *Front. Neurorobot.*, 13:9, March 2019.

[56] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P Agapiou, Max Jaderberg, Alexander S Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, November 2019.

[57] Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. Benchmarking Model-Based reinforcement learning. July 2019.

[58] Hua Wei, Guanjie Zheng, Huaxiu Yao, and Zhenhui Li. IntelliLight: A reinforcement learning approach for intelligent traffic light control. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, pages 2496–2505, New York, NY, USA, July 2018. Association for Computing Machinery.