Title

Safety for Contact-Rich Manipulation via Learned Representations

Author

Sarda, Sanjit

Publication Date

2026-04-01

UNIVERSITY OF CALIFORNIA

Los Angeles

Safety for Contact-Rich Manipulation via Learned Representations

A thesis submitted in partial satisfaction

of the requirements for the degree Bachelor of Science

in Electrical Engineering

by

Sanjit Sarda

2026

ABSTRACT OF THE THESIS

Safety for Contact-Rich Manipulation via Learned Representations

by

Sanjit Sarda

Bachelor of Science in Electrical Engineering
University of California, Los Angeles, 2026

Assistant Professor Anushri Dixit, Co-Chair
Assistant Professor Jason J. Choi, Co-Chair

As robotic agents become more capable at performing a wide range of tasks and are increasingly deployed in situations where they are expected to be reliable, safety has become an increasingly important topic in robotics. Safety in robotic systems has seen substantial progress in both formal verification and learning-based methods. Formal safety methods including Hamilton-Jacobi (HJ) Reachability and Control Barrier Functions (CBFs) are powerful frameworks that can provide mathematical guarantees for physical safety constraints, however they scale poorly with problem complexity. Learning-based approaches to robotic safety, while broad in scope, often lack the mathematical guarantees of their formal counterparts.

This thesis studies semantic safety for manipulators in contact-rich scenes through the Latent Safety Filters (LSFs) paradigm. Contact-rich manipulation tasks require a deeper understanding how the objects in the environment interact with each other and the robot. Thus, a safety framework must be able to reason about how the robots interations do not just affect the objects it immediately contacts, but also how those objects interact with

other objects in the environment. In tasks such as removing blocks from a contact-rich environment like books placed on top of each other, a safety discriminator will have to be able to reason about dynamics on a high level (ex: pulling lower books at certain angles will prevent friction from dragging the top books) beyond trying to create an accurate contact dynamics model. Existing formal verification methods often struggle to capture the complexity of these interactions, since they are difficult to model analytically, and learning-based approaches are often limited by the lack of guarantees they provide. LSFs offer a strong alternative by combining the semantic representations of learning based methods with the formal guarantees of reachability-based safety. However, despite the generalization appeal, LSFs still require a classification model trained on examples of unsafe actions. Additionally, since LSFs use World Models, they may struggle to model contact-rich scenarios under partial observability and are subject to dataset bias in World Model training.

This thesis evaluates Latent Safety Filters as a framework for semantic safety in contact-rich manipulation. It first implements reachability-based SafeRL on simple simulation tasks to study how learned safety value functions filter unsafe actions and how their performance depends on training data coverage. It then develops data collection and world-model training pipelines for a 2D Dubins car environment and a manipulation task involving a Kinova Gen3 arm removing a lower book from a stack. A DreamerV3-style world model is shown to produce more stable predictions than a PlaNet-style model in the Dubins setting, and the learned world-model framework is extended to multi-camera contact-rich table manipulation observations. Finally, latent-space reachability analysis is applied to the manipulation task, where the resulting Latent Safety Filter prevents 78% of in-distribution failure trajectories without additional simulator or hardware interaction during safety training. These results demonstrate the promise of latent-space safety filtering for policy-agnostic robotic safety, while also highlighting the importance of diverse data collection, accurate latent dynamics, and improved sim-to-real transfer for reliable deployment.

The thesis of Sanjit Sarda is being written.

Danijela Cabric

Jason J. Choi, Committee Co-Chair

Anushri Dixit, Committee Co-Chair

University of California, Los Angeles

2026

Contents

List of Figures

Chapter 1

Introduction

1.1 Background and Motivation

As robots move from structured industrial settings into unstructured, human-centered environments, they must operate safely despite uncertainty, complex contact, and changing task conditions. This is especially challenging for manipulation tasks, where small errors in force, perception, or object interaction can lead to unintended collisions, dropped objects, or cascading failures. As a result, robotic safety requires methods that can reason not only about immediate physical constraints, but also about longer-horizon task outcomes and context-dependent failures.

The two main analytical safety methods include Hamilton-Jacobi (HJ) Reachability and Control Barrier Functions (CBFs). With an accurate dynamics and worst-case disturbance model, HJ reachability uses grid based Dynamic Programming to solve for a Backwards Reachable Set (BRS), which represents the set of all states, which rigorously guarantees that staying within the tube will keep the system within a safe operating region. Safety is enforced by filtering out actions that would cause the system to exit the safe set [1, 2, 3]. However, solving HJ reachability problems with a state dimension greater than 5 are considered intractable [3]. On the other hand, CBFs enforce safety by providing safe

controls that keep the system inside a pre-defined safe set, as well as certifying the invariance of the safe set [4, 5]. However, for safe set verification, CBFs rely on simple dynamics model and scale poorly with increasing coupling of constraints. Both HJ reachability and CBFs also suffer from a lack of generalization and do not offer a natural path to analyze semantic/contextual safety. Despite these flaws, their mathematical guarantees make them quite desirable, and many learning-based methods leverage this in their problem formulation [6, 1].

One such method is Latent Safety Filters [7], which learns a safety value function from pixels, by breaking down the problem into a learned world model, and a learned safety filter in the latent space of the world model, to enforce semantic safety for a robotics task. Latent Safety Filters combines the formal structure of reachability-based safety with representation learning in learned world models. Instead of operating on the high dimensional observation space, it learns a safety value function using the same techniques as SafeRL [8] in the latent space of a learned world models. This makes it possible to apply it to semantic manipulation tasks, since the latent space can capture essential features to represent the semantic aspects of the task.

1.2 Summary of Contributions

This thesis investigates Latent Safety Filters for semantic safety in contact-rich robotic manipulation. Building on Hamilton-Jacobi reachability-based safety analysis, safe reinforcement learning, and learned world models, this work studies whether safety reasoning can be performed in a compact latent representation rather than directly in the high-dimensional observation or simulator state space [8, 7, 9].

The main contributions of this thesis are as follows. First, this work implements and evaluates reachability-based safe reinforcement learning on benchmark control environments, including Half-Cheetah and Antball, to study how learned safety value functions can filter

unsafe actions from nominal policies. These experiments highlight a key limitation of safety filtering: the learned shield depends strongly on the coverage and diversity of the state-action distribution used during training.

Second, this work develops and evaluates world-model training pipelines for both a 2D Dubins car environment and then extends to a contact-rich table manipulation task in which a Kinova Gen3 arm must pick up a lower book without disturbing other stacked objects. A PlaNet-style recurrent state-space model and a DreamerV3-style world model are compared on the Dubins car task, showing that the DreamerV3-style model produces more stable short-horizon and open-loop predictions.

Third, this work develops a Reinforcement Learning (RL) framework for training manipulation policies using a robot arm. This includes the simulator environment, reward function design and a diverse policy training procedure. The same world-model framework is then extended to the manipulation task using multi-camera observations.

Fourth, this work applies latent-space reachability analysis to the manipulation task. A safety value function is trained entirely inside the learned world model, without additional simulator or hardware interaction during safety training. For rollouts that are in-distribution rollouts, the resulting latent safety filter prevents 78% of failure trajectories, demonstrating the promise of latent-space safety filtering for policy-agnostic robotic manipulation.

Finally, this thesis identifies practical limitations that must be addressed before deployment on physical hardware. These include world-model bias in contact-rich scenes, limited state-action coverage from nominal policies, sensitivity to the design of auxiliary state-estimation losses, and the need for improved Sim2Real transfer. Together, these findings suggest that Latent Safety Filters are a promising direction for semantic robotic safety, while also showing that diverse data collection and accurate latent dynamics remain essential for reliable safety filtering.

Chapter 2

Problem Formulation

2.1 Hamilton-Jacobi Reachability

Following work in Hamilton-Jacobi (HJ) reachability [2, 3], we consider a discrete-time controlled system

$$x_{t+1} = f(x_t, u_t), \qquad x_t \in \mathcal{X},\ u_t \in \mathcal{U}. \tag{2.1}$$

Here, x_t denotes the system state at time t, u_t denotes the control input, $\mathcal{X}$ is the state space, $\mathcal{U}$ is the control space, and $f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ is the dynamics map. We specify safety by a failure set

$$\mathcal{F} = \{x \in \mathcal{X} : \ell(x) \leq 0\}, \tag{2.2}$$

where $\ell : \mathcal{X} \to \mathbb{R}$ represents the safety margin function, with $\ell(x) > 0$ denoting safety and $\ell(x) \leq 0$ denoting failure.

The infinite-horizon reachability safety value function is

$$V(x_0) = \sup_{\pi} \inf_{t \geq 0} \ell(x_t), \qquad x_{t+1} = f(x_t, \pi(x_t)). \tag{2.3}$$

Here, $\pi : \mathcal{X} \to \mathcal{U}$ denotes a feedback control policy, and $V(x_0)$ is the largest worst-case safety margin that can be maintained over time starting from x_0. By the dynamic program-

ming principle, optimizing over feedback policies can be decomposed into choosing the first control action and then applying an optimal continuation policy from the successor state. Equivalently, V satisfies the fixed-point Bellman equation

$$V(x) = \min \left\{ \ell(x), \max_{u \in \mathcal{U}} V(f(x,u)) \right\}. \tag{2.4}$$

The backward reachable unsafe set is the zero level set of the safety value function

$$\mathcal{B} = \{x \in \mathcal{X} : V(x) \leq 0\}, \tag{2.5}$$

which contains states from which the system is guaranteed to eventually enter the failure set, $\mathcal{F}$ regardless of what control actions are used.

The associated safety-preserving controller can be found by maximizing the value function over the policy

$$\pi_{\text{safe}}(x) \in \arg\max_{u \in \mathcal{U}} V(f(x,u)). \tag{2.6}$$

Given a nominal policy π_{nom}, a safety filter can be used to intervene where the nominal policy fails to keep the system safe.

$$\pi_{\text{filter}}(x) = \begin{cases} \pi_{\text{nom}}(x), & V(f(x, \pi_{\text{nom}}(x))) > \epsilon, \\ \pi_{\text{safe}}(x), & \text{otherwise}, \end{cases} \tag{2.7}$$

where $\epsilon \geq 0$ is a safety margin.

2.1.1 Safe RL

As discussed before, using a grid based solver to solve for the exact HJ reachability value function is intractable at dimensions larger than 5. To mitigate this, we solve for the approximate HJ Reachability Value function using reinforcement learning.

Standard reinforcement learning maximizes a discounted sum of rewards,

$$V(x) = \max_{u \in \mathcal{U}} \left[r(x,u) + \gamma V(f(x,u)) \right], \tag{2.8}$$

where $\gamma \in [0,1)$ is the discount factor. This formulation works well for standard performance objectives, since it maximizes the expected reward. However, safety is not additive the way performance is. The average safety margin can be maximized without the system always staying within the safe set.

Following [8], safety can instead be written as a worst case payoff objective. Let $\ell(x)$ be a safety margin, with $\ell(x) \geq 0$ indicating that x is safe. The discounted safety value is defined by the Bellman-style recursion

$$V_\gamma(x) = (1-\gamma)\ell(x) + \gamma \min \left\{ \ell(x), \max_{u \in \mathcal{U}} V_\gamma(f(x,u)) \right\}. \tag{2.9}$$

The corresponding state-action safety value is

$$Q_\gamma(x,u) = (1-\gamma)\ell(x) + \gamma \min \left\{ \ell(x), \max_{u' \in \mathcal{U}} Q_\gamma(f(x,u),u') \right\}. \tag{2.10}$$

Here, $Q_\gamma(x,u)$ denotes the discounted safety value obtained by first applying action u at state x and then acting optimally with respect to safety. A safety-preserving policy is then obtained by choosing the action with the largest safety value:

$$\pi_{\text{safe}}(x) = \arg\max_{u \in \mathcal{U}} Q_\gamma(x,u). \tag{2.11}$$

The learned safe set is the set of states from which the best safety value is nonnegative,

$$\mathcal{S}_{\text{safe}} = \left\{ x \in \mathcal{X} : \max_{u \in \mathcal{U}} Q_\gamma(x,u) \geq 0 \right\}. \tag{2.12}$$

As $\gamma \to 1$, this discounted formulation approaches the undiscounted Hamilton-Jacobi

safety objective, and can still readily be solved using temporal-difference learning methods such as Q-learning or SAC as shown in the paper.

2.2 World Models

A world model learns a compact latent dynamics model from observation-action trajectories. World Models are typically formulated like this:

We are looking to model transitions between image observations under actions. Since images do not reveal full state information we model them as Partially Observable Markov Decision Processes (POMDPs).

We model the privileged state as s_t, and the partially observable observations as $o_t = f_o(s_t)$, where f_o maps states into a partial observation (such as an image). This can be thought of as taking a picture of the system. System dynamics are modeled as $s_{t+1} = f(s_t, a_t)$. For observations (typically images) o_t, actions a_t, and latent states z_t, the model is

$$z_t \sim q_\phi(z_t \mid z_{t-1}, a_{t-1}, o_t), \qquad z_{t+1} \sim p_\theta(z_{t+1} \mid z_t, a_t), \tag{2.13}$$

with decoders

$$\hat{o}_t \sim p_\theta(o_t \mid z_t), \qquad \hat{r}_t \sim p_\theta(r_t \mid z_t). \tag{2.14}$$

Here, q_ϕ is the encoder, p_θ is the learned latent dynamics model, $\hat{o}_t$ denotes the reconstructed observation and $\hat{r}_t$ denotes the predicted reward, and ϕ and θ are learned parameters. We ignore the reward prediction, since there is no planning objective that depends on it. The standard variational training objective is

$$\begin{aligned} \mathcal{L}_{\mathrm{WM}}(\theta, \phi) = \sum_t \Big(& \mathbb{E}_{q_\phi} \left[\log p_\theta(o_t \mid z_t) + \log p_\theta(r_t \mid z_t) \right] \\ & -\beta D_{\mathrm{KL}} \left(q_\phi(z_t \mid z_{t-1}, a_{t-1}, o_t) \,\|\, p_\theta(z_t \mid z_{t-1}, a_{t-1}) \right) \Big). \end{aligned} \tag{2.15}$$

Here, $\mathbb{E}_{q_\phi}$ denotes expectation under the approximate posterior, D_{KL} denotes the Kullback-

Leibler divergence, and β controls the strength of the KL regularization term. For latent-space reachability, the failure margin is represented directly in latent space by a learned classifier

$$\ell_\psi : \mathcal{Z} \to \mathbb{R}, \qquad \mathcal{F}_z = \{z \in \mathcal{Z} : \ell_\psi(z) \leq 0\}. \tag{2.16}$$

The latent HJ Bellman equation becomes

$$V_z(z) = \min \left\{ \ell_\psi(z), \max_{a \in \mathcal{A}} \mathbb{E}_{z' \sim p_\theta(\cdot|z,a)} \left[V_z(z') \right] \right\}. \tag{2.17}$$

Here, V_z denotes the latent safety value function, and the expectation is taken over the next latent state z' sampled from the learned transition model $p_\theta(\cdot \mid z, a)$. A discounted approximation is

$$V_z(z) = \min \left\{ \ell_\psi(z), \gamma \max_{a \in \mathcal{A}} \mathbb{E}_{z' \sim p_\theta(\cdot|z,a)} \left[V_z(z') \right] \right\}. \tag{2.18}$$

2.2.1 PlaNet

PlaNet [10] uses a recurrent state-space model with deterministic hidden state h_t and stochastic latent state s_t. The deterministic recurrent state evolves as

$$h_t = f_\theta(h_{t-1}, s_{t-1}, a_{t-1}). \tag{2.19}$$

Where, h_t summarizes the deterministic history-dependent component of the latent state, while s_t captures stochastic latent information. The stochastic prior, posterior, observation model, and reward model are

$$\text{prior:} \qquad p_\theta(s_t \mid h_t), \tag{2.20}$$

$$\text{posterior:} \qquad q_\phi(s_t \mid h_t, o_t), \tag{2.21}$$

$$\text{observation model:} \qquad p_\theta(o_t \mid h_t, s_t), \tag{2.22}$$

$$\text{reward model:} \qquad p_\theta(r_t \mid h_t, s_t). \tag{2.23}$$

Here, $p_\theta(s_t \mid h_t)$ is the prior over the stochastic latent state before observing o_t, while $q_\phi(s_t \mid h_t, o_t)$ is the posterior after incorporating the observation. Thus the latent state used for planning is

$$z_t = (h_t, s_t). \tag{2.24}$$

The PlaNet objective maximizes a variational lower bound:

$$\begin{aligned}\mathcal{L}_{\text{PlaNet}} = \sum_{t=1}^{T} \Big(& \mathbb{E}_{q_\phi} \left[\log p_\theta(o_t \mid h_t, s_t) + \log p_\theta(r_t \mid h_t, s_t) \right] \\ & - D_{\text{KL}} \left(q_\phi(s_t \mid h_t, o_t) \,\|\, p_\theta(s_t \mid h_t) \right) \Big).\end{aligned} \tag{2.25}$$

Here, $\mathbb{E}_{q_\phi}$ denotes expectation under the approximate posterior, and D_{KL} denotes the Kullback-Leibler divergence.

PlaNet plans by solving a finite-horizon latent model-predictive control problem:

$$a^\star_{t:t+H-1} = \arg\max_{a_{t:t+H-1}} \mathbb{E}_{p_\theta} \left[\sum_{\tau=t}^{t+H-1} r_\tau \,\middle|\, z_t,\ a_{t:t+H-1} \right]. \tag{2.26}$$

Only the first action is executed,

$$a_t = a^\star_t, \tag{2.27}$$

and the optimization is repeated after receiving the next observation, giving an MPC-style controller in latent space.

PlaNet also introduces latent overshooting. Define the d-step predictive prior as

$$p^{(d)}_\theta(s_{t+d} \mid z_t, a_{t:t+d-1}) = \int \prod_{k=1}^{d} p_\theta(s_{t+k} \mid h_{t+k})\, ds_{t+1:t+d-1}. \tag{2.28}$$

The overshooting regularizer encourages multi-step latent predictions to match future posteriors:

$$\mathcal{L}_{\text{over}} = \sum_{d=1}^{D} \beta_d \sum_{t} D_{\text{KL}} \left(\text{sg}\left[q_\phi(s_{t+d} \mid h_{t+d}, o_{t+d}) \right] \,\middle\|\, p^{(d)}_\theta(s_{t+d} \mid z_t, a_{t:t+d-1}) \right), \tag{2.29}$$

, where sg[·] denotes a stop-gradient operator.

Chapter 3

Methods and Results

3.1 Project Components

Latent Safety Filters primarily consist of two main components. The World Model to reduce the observations to a smaller latent representation, and the Safe Reinforcement Learning component to learn the Reachability Value function. Training both components is essential for the effective operation of the Latent Safety Filters [7].

3.1.1 Safe Reinforcement Learning

Safe Reinforcement Learning, otherwise known as HJ-RL uses reinforcement learning techniques to solve a Hamilton Jacobi reachability problem. This enables finding an approximate solution to the dynamic programming equation that defines the reachability of a system.

To experiment with Safe Reinforcement Learning before applying it to the main task, two simpler tasks were tested to validate the approach.

Half Cheetah

The `Half-Cheetah-v5` is a standard benchmark in reinforcement learning, where the goal is to train the cheetah to run as fast as possible while maintaining balance. Using a simple

DDPG (Deep Deterministic Policy Gradient) based algorithm, Half Cheetah can successfully learn to run at high speeds while maintaining stability. However, to test Safe Reinforcement Learning, a safety margin was defined using the torso height and pitch angle, such that below zero would indicate a failure.

The safety value function was trained using an off-policy actor-critic structure. The safety actor-critic used two hidden layers of size `256` for both the actor and critic, with `ReLU` activations and Adam optimizers using a learning rate of `3e-4`. A replay buffer of `500,000` transitions was used with minibatches of `256`. Training was run for `100,000` environment steps, with the first 100 steps collected using random actions. During evaluation, the safety shield sampled 256 candidate actions over 8 rounds around the learned safety actor's nominal action and selected a candidate with predicted safety value above zero when available.

After training, the trained value function was used to filter a uniform distribution of the action space to ensure that only safe actions were taken. Since the nominal policy is random, it does not move the cheetah forward or keep it safe. However, applying the safety filter to the random policy shows the cheetah moving randomly without falling.

3.1.2 Antball

Antball is a more complex environment developed for testing a safety filtering tool known as EigenSafe. The environment is an offshoot of the simple quadruped Ant environment, which similar to the Half-Cheetah, is a standard benchmark in reinforcement learning and the goal is to train the ant to run as fast as possible while maintaining balance. The Antball environment adds an additional layer of complexity by adding a plate on top of the ant's torso and a ball the ant must balance to remain safe. This environment is great for testing the effectiveness of a safety filter, since a nominal policy can be trained to move the ant forward regardless of the ball's safety, and the safety filter can filter nominal actions that cause the ball to fall. The training process for the Antball environments and the hyperparameters used were the same as those used for the Half-Cheetah.

While in theory this is a highly effective approach, it demonstrates a significant challenge in applying this approach to real-world scenarios: state space coverage. In theory, a safety filter can take a nominal action and replace it with a safer one. For example, if the nominal policy chooses an action (u_{nom}), the filter tries to find a nearby action (u_{safe}) such that the next state remains safe:

$$\begin{aligned} u_{\text{safe}} = \arg\min_{u} \quad & |u - u_{\text{nom}}|^2 \\ \text{s.t.} \quad & V(x_{t+1}) \geq 0. \end{aligned} \tag{3.1}$$

If the nominal policy is only trained to make the ant run forward quickly, then it may mostly explore actions that are good for speed but bad for balancing the ball. In this case, the safe alternative action may be overly conservative and prevent the ant from making progress. This was exactly the issue encountered in the Antball environment.

3.1.3 World Models

As discussed in the earlier sections, the World Model is a critical component of the Latent Safety Filters. It is primarily responsible for reducing the high-dimensional observations from the environment to a more manageable latent representation that can be used by the safety filters. Research on World Models has primarily used two approaches: variational autoencoders and joint embedding predictions. While Latent Safety Filters have used both approaches, the focus of this work is on the variational autoencoder approach discussed in problem formulation. Additionally, while autoencoder based world models are primarily trained in a self-supervised manner using reinforcement learning to collect data and explore the environment, since a policy is not required for latent space reachability, the world model can be trained without training a policy.

To experiment with world models, a toy task was created simulating a 2D Dubins car navigating through a cluttered environment.

The Dubins car environment is modeled as a nonlinear dynamical system with a state

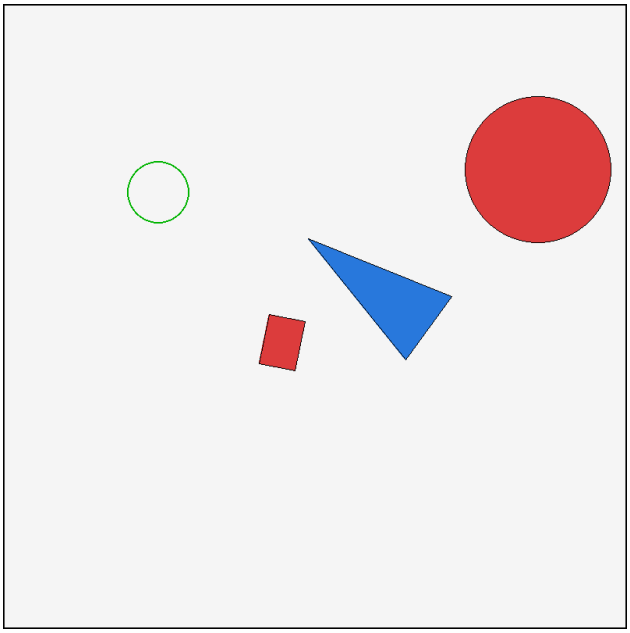

Figure 3.1: 2D Dubins Car Task.

vector that includes position, heading, and speed. The action space consists of steering and throttle inputs that affect the car's motion. The dynamics are governed by this set of discretized differential equations:

$$x_{t+1} = x_t + dt \cdot v_{t+1} \cdot \cos(\theta_t) \tag{3.2}$$

$$y_{t+1} = y_t + dt \cdot v_{t+1} \cdot \sin(\theta_t) \tag{3.3}$$

$$\theta_{t+1} = \theta_t + dt \cdot a_t, \quad a_t \in [-1.25, +1.25] \tag{3.4}$$

$$v_{t+1} = \text{clip}(v_t + dt \cdot a_t \cdot th_t - 0.15 \cdot v_t, [0, 3.0]) \tag{3.5}$$

Where,

- x_t, y_t: Position of the car at time t
- θ_t: Heading of the car at time t
- v_t: Speed of the car at time t
- a_t: Steering input at time t

- th_t: Throttle input at time t
- u_t: Control input at time t is defined as $u_t = [a_t, th_t]$
- dt: Time step

Data is collected by running the simulator under a heuristic controller that chooses a short horizon goal and follows it until it reaches the end of the episode, collides with an obstacle or is given a new goal. At the end of each episode, if the recording is long enough, the data is saved for later use and a new map is generated. The dataset for each episode contains a sequence of states, images, and actions. Once recorded, the rollouts are loaded as fixed-length training sequences, where they are broken into smaller contiguous windows of length `64`. The dataset consists of `38,242` such sequences, which corresponds to roughly `2.4 Million` action state observation pairs.

A number of dataset parameters needed to be tuned to ensure the model's effectiveness. This included the size of the objects in the environment (car, obstacles), the timestep and dataset diversity (ex: using random maps). Without tuning these parameters, the model either suffered from posterior collapse (would reconstruct a learned map that was not representative of the true map), or fail to learn the underlying dynamics.

The final model was trained using the `DreamerV3` [9] architecture, with the following hyperparameters:

- **RSSM latent state:** Deterministic state size 512; Stochastic state with 32 categorical variables and 32 classes per variable.
- **Encoder/decoder:** convolutional base depth 32, 3 convolutional stages, kernel size 4, `SiLU` activations, and `MSE` image reconstruction.
- **MLP components:** hidden size 512 with 2 layers for observation, image, and dynamics networks.

- **Training setup:** Adam optimizer, learning rate 10^{-4}, batch size 8, rollout clip length 64, and 125 training epochs.
- **Regularisation and losses:** KL free-nats threshold 1.0, unimix ratio 0.01, gradient clipping at norm 20, and open-loop rollout horizon of 16 steps.

PlaNet Style Implementation

The original implementation of a World Model was implemented in a similar style to PlaNet [10]. It uses a Convolutional Neural Network (CNN) encoder/decoder to compress each image into a latent embedding and decode it back to the original image. The backbone of the latent space is a recurrent neural network that maintains a hidden deterministic memory state over time, and models transitions using the deterministic component and a stochastic component.

The training objectives includes three components: reconstruction, latent consistency through KL regularization, and multi-step open-loop consistency through overshooting. The reconstruction term ensures that the model can accurately reconstruct the input images from the latent representation. The KL objective encourages the prior distribution, which represents the expected distribution of the latent variable after an action is taken, to be close to the posterior distribution, which represents the actual distribution of the latent variable given the observation, enabling the model to learn transition dynamics. Finally, the multi-step open-loop consistency loss extended the KL divergence to a short open loop horizon to ensure that the model can make accurate predictions over longer horizons without posterior reconstruction. In both, the one step predictions and multi-step predictions, the KL divergence terms were clipped to stabilize training.

Figure 3.2 compares the original Dubins car trajectory with the one-step predictions and open-loop predictions produced by the PlaNet-style world model. The first column shows the original trajectory, the second column shows the 1-step predictions, and the third column shows the open-loop predictions. Each row is separated by 50 frames. From this, it is

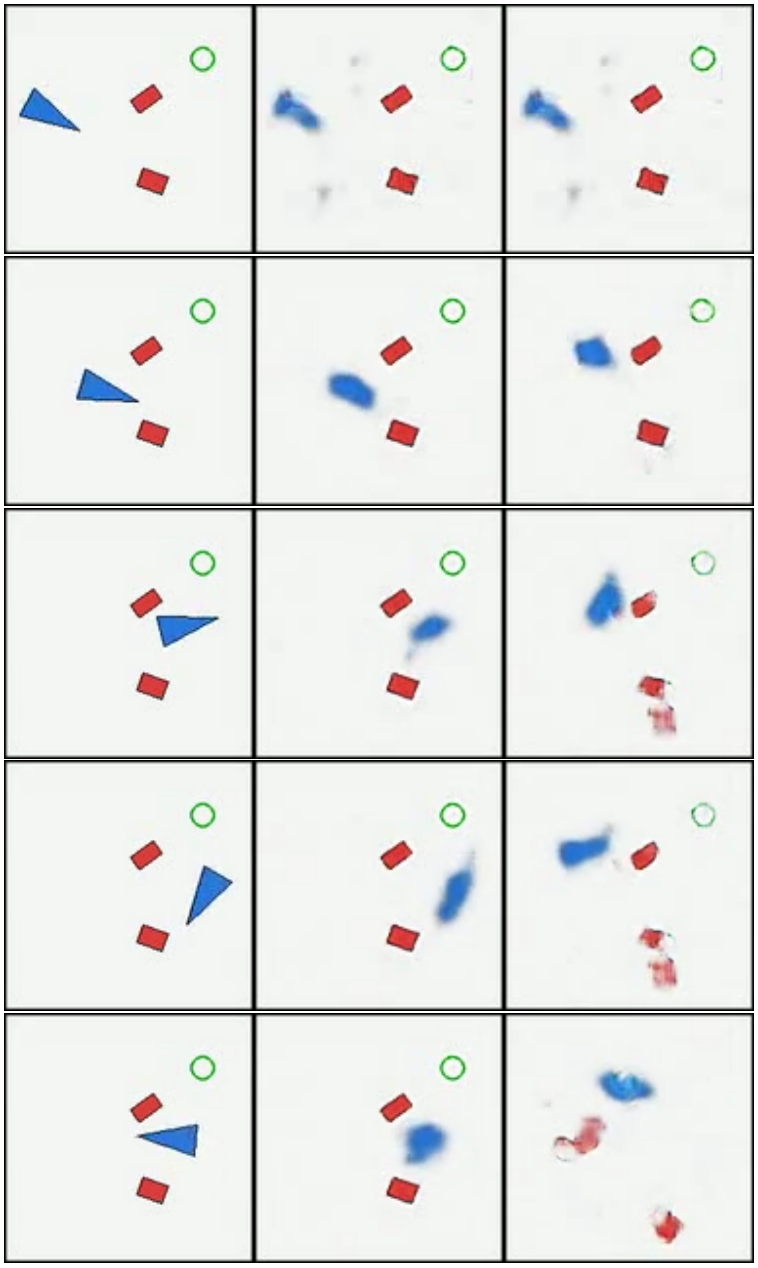

Figure 3.2: Original Dubins Car Trajectories; 1 Step Predictions; Open Loop Predictions

clear that one step predictions perform reasonably well, however, the open-loop predictions quickly diverge from the original trajectory. A video representation of this shows the open loop predictions having dynamics behavior that is quite different from the original trajectory however remains somewhat meaningful.

DreamerV3 Implementation

DreamerV3 is a more recent and advanced version of the PlaNet architecture that introduces several improvements key to improving the model's ability to learn and generalize from the data. These improvements include a number of tricks, including the use of a categorical latent space, a block wise recurrent update, and a split prior consistency loss that isolated the gradients of the KL divergence loss into a dynamics and a reconstruction component. An implementation of its architecture developed by [11, 12] was used in this work.

PlaNet and DreamerV3's problem formulations do not train a state reconstruction head, since full state information is rarely available for the tasks shown in their experiments. However if full state information is available, as it is for the Dubins car task as well as the arm task, adding a state reconstruction head to the model in the form of a simple MLP and training it on the full state information improved its performance.

Figure 3.3 shows the DreamerV3-style model's short-horizon and open-loop predictions on the Dubins car task. The first column shows the original trajectory, the second, third, fourth, and fifth columns each show 1, 4, 8 and 16 step predictions respectively, and the sixth column shows the open-loop predictions. The reconstructions are shown to perform better than the PlaNet reconstructions, and the short horizon predictions remain within a reasonable range of the original trajectory. This shows the Dreamer implementation significantly outperformed PlaNet in terms of open-loop generation and overall performance.

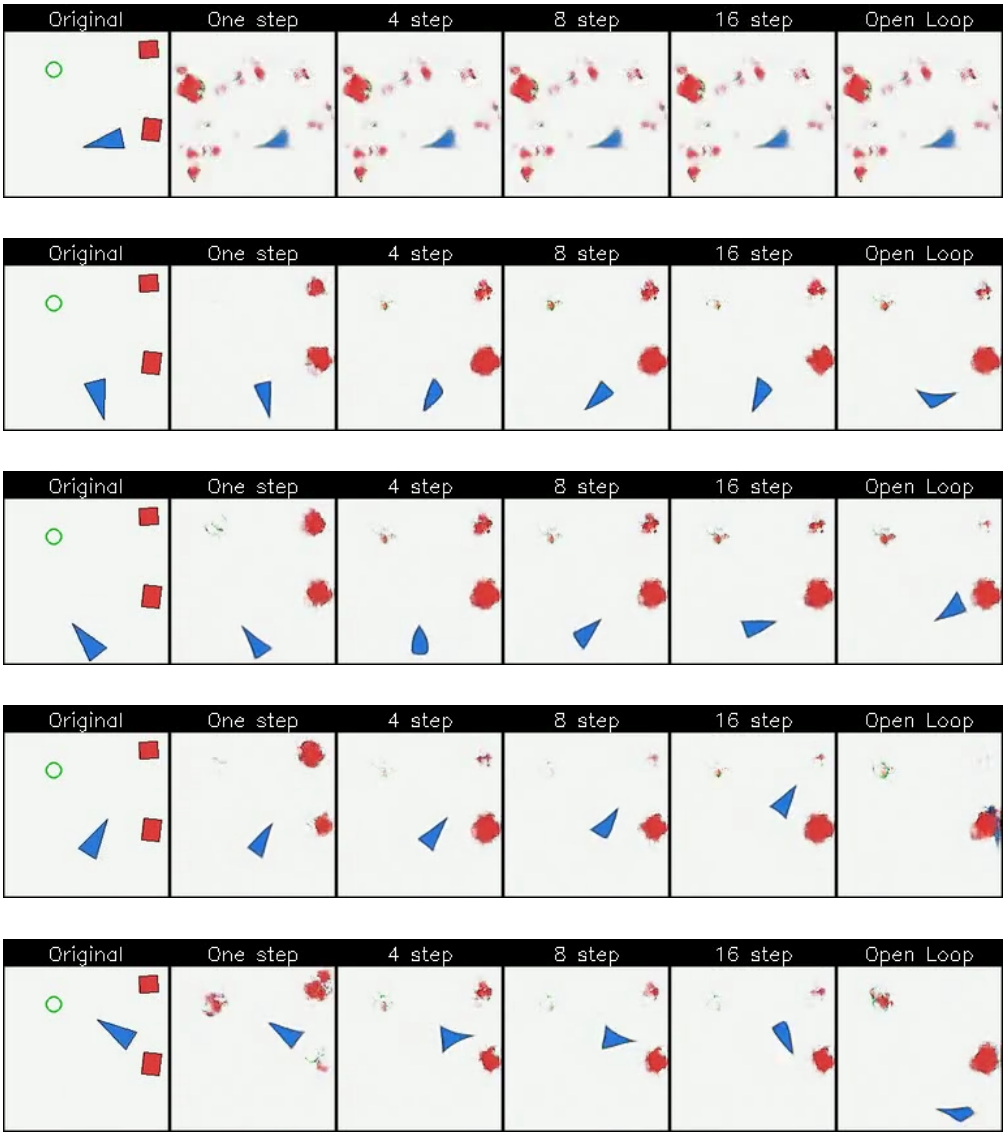

Figure 3.3: Dubins Car Trajectories; 1 Step Predictions; Open Loop Predictions

3.2 Table Task Setup

Latent Safety Filters were benchmarked on a manipulation task, where the robot arm needs to pick up the bottom book without disturbing the other books. The environment, seen in Figure 3.4, is a MuJoCo manipulation scene built around a fixed Kinova Gen3 arm equipped with a Robotiq gripper. The robot is mounted on a table with a smaller table on top of the main one. Three books are stacked on top of each other on the mini table, each modeled as a box with random position and orientation initializations that keep the books stacked on top of each other. Actions are provided to the robot arm in terms of delta end-effector position and orientation controls as well as a gripper command. The primary task in this environment is to pick up the bottom book without disturbing the other books. Various reward functions are available, including contact based, movement-based, and grasping-based rewards. The privileged observations include the robot state, object positions, object corner locations and grip related signals. An additional compact RSSM State is also recorded to capture the essential information needed to train a margin function for safety filtering.

The environment is trained using StableBaseline3's Soft Actor-Critic (SAC) with 32 parallel MuJoCo table environments wrapped by `VecMonitor`. SAC collects transitions from these environments into a replay buffer and, after `learning_starts = 10000`, updates its actor and critic networks using minibatches of size `256`, a replay buffer of size `1,000,000`, `train_freq = 1`, `gradient_steps = 1`, discount factor $\gamma = 0.99$, and learning rate $3e-4$. The main task rewards actions that lead to a successful notebook pickup, using a combination of dense rewards such as position and orientation errors, grip pose errors, rewards and penalties for the correct gripper grasp and sparse rewards like successful contacts and successful pickups. The end effector is first encouraged to align with the notebook rectangle, minimize vertical and orientation error, avoid closing the gripper too early, make one- and then two-finger contact, and finally lift the notebook above a height threshold of 0.25 m. The reward begins as a dense penalty, $-3z_{\text{err}} - 3q_{\text{err}} - xy_{\text{err}}$, so SAC receives continuous feedback for improving pose alignment before the sparse pickup success is reached. Once

the gripper is close enough in height and orientation, closing the gripper becomes rewarded; if both fingers contact the notebook, lift height contributes positively and successful pickup adds an additional bonus.

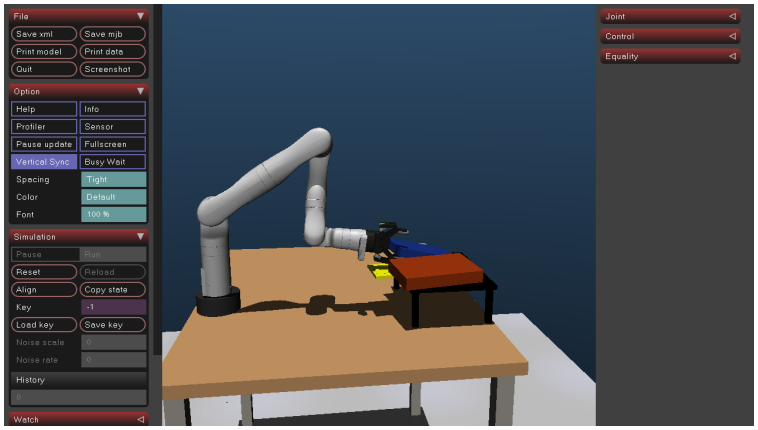

Figure 3.4: Digital Twin of Manipulation Task using Mujoco.

The MuJoCo environment is a digital twin of the physical manipulation task shown in Figure 3.5, allowing for safe and efficient testing of the safety filtering approach before deploying it on the real hardware.

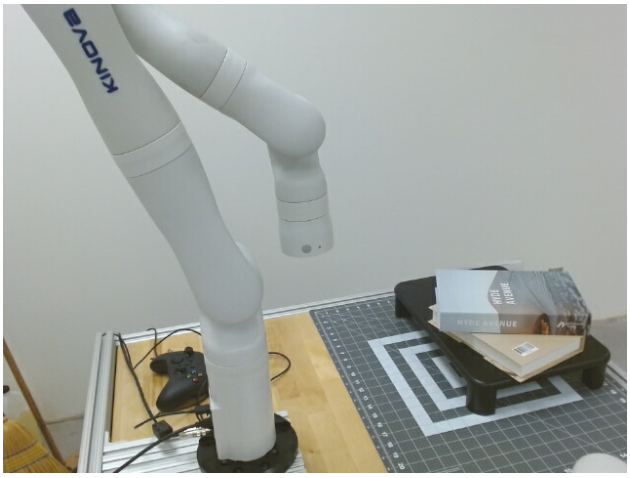

Figure 3.5: Manipulation task: Hardware arm setup.

The tasks in this environment are readily and safely solvable using Reinforcement Learning, but safety filters are intended to be a policy agnostic approach to ensuring safety in robotic systems. This means that a policy trained without reinforcement learning or even a

human operator can be shielded by the safety filters.

3.2.1 RSSM Data Collection using Reinforcement Learning

Similar to the Dubins car setup, a large dataset of trajectories needed to be collected through reinforcement learning.

To collect diverse data, multiple policies are trained using different reward functions rather than relying on one narrow task objective. These rewards include behaviors such as touching objects, reaching grasp-ready poses, picking up objects, disturbing objects, and dropping or avoiding dropping objects. Training across these reward variants encourages the policies to visit different parts of the state-action space, including successful grasps, partial contacts, object disturbances, and failure cases.

After Reinforcement Learning training, rollouts are generated from a list of saved Stable-Baselines3 SAC checkpoints. During collection, the script cycles through or randomly samples policies, resets the randomized environment, and runs the policy stochastically by default. At every timestep it records the compact RSSM state, action, full MuJoCo simulator state, and visibility of the end effector plus the notebook, book, and encyclopedia from two fixed RSSM camera views. The rollout is then cropped to the longest segment where all key entities (the end effector, the notebook, the book, and the encyclopedia) remain sufficiently visible, rendered from both cameras, and saved as an `.npz` file containing aligned arrays: state and camera observations with length $T + 1$, and action with length T. The dataset consists of `14816` such sequences, which corresponds to roughly `0.95 Million` action state observation pairs.

3.2.2 World Model Training

Training the World Model for this manipulation task is similar to the process used in the Dubins car setup, in fact most of the architecture is reused from the previous setup. The `DreamerV3` is used to train the world model on the collected dataset. Two key differences

are:

- The state estimator is used to reconstruct a custom state representation that includes the arm end effector position and orientation, and for each object (notebook, book, encyclopedia), its position, whether it is being gripped, if it is on the table, or if it is on the floor. These are binary flags, so the state estimator will output the logits representing the probability of each flag being true, and a BCEWithLogitsLoss (Binary Cross Entropy with Logits Loss) is used along with MSE as the training objective for the state estimator.

- The observations of the environment are stacked on two camera views to provide a more comprehensive representation of the scene. This is done in the channel dimension of the input tensor.

Figure 3.6 compares the original manipulation-task trajectories with one-step, short-horizon, and open-loop predictions from the trained world model. This approach results in a well performing world model without much more effort from the previous setup. One interesting result noticed in the videos of the trained model was the effect of using BCE (Binary Cross Entropy) loss in the state estimator. While BCE Loss is the correct way to handle the binary classification problem of predicting the presence or absence of each flag, BCE tends to compress the output distribution to either 0 or 1, leading to a loss of information in the latent space. This resulted in books teleporting directly to fall or disappear when the World Model predicts that they will fall in the near future, rather than smoothly transitioning to a falling state. This can be mitigated by weighing the BCE loss less heavily against the MSE loss.

3.2.3 Latent Space Reachability

Leveraging the trained world model, we learn the safety value as a function of the latent states. To ease the training process, the same DDPG algorithm is used. Here the safety

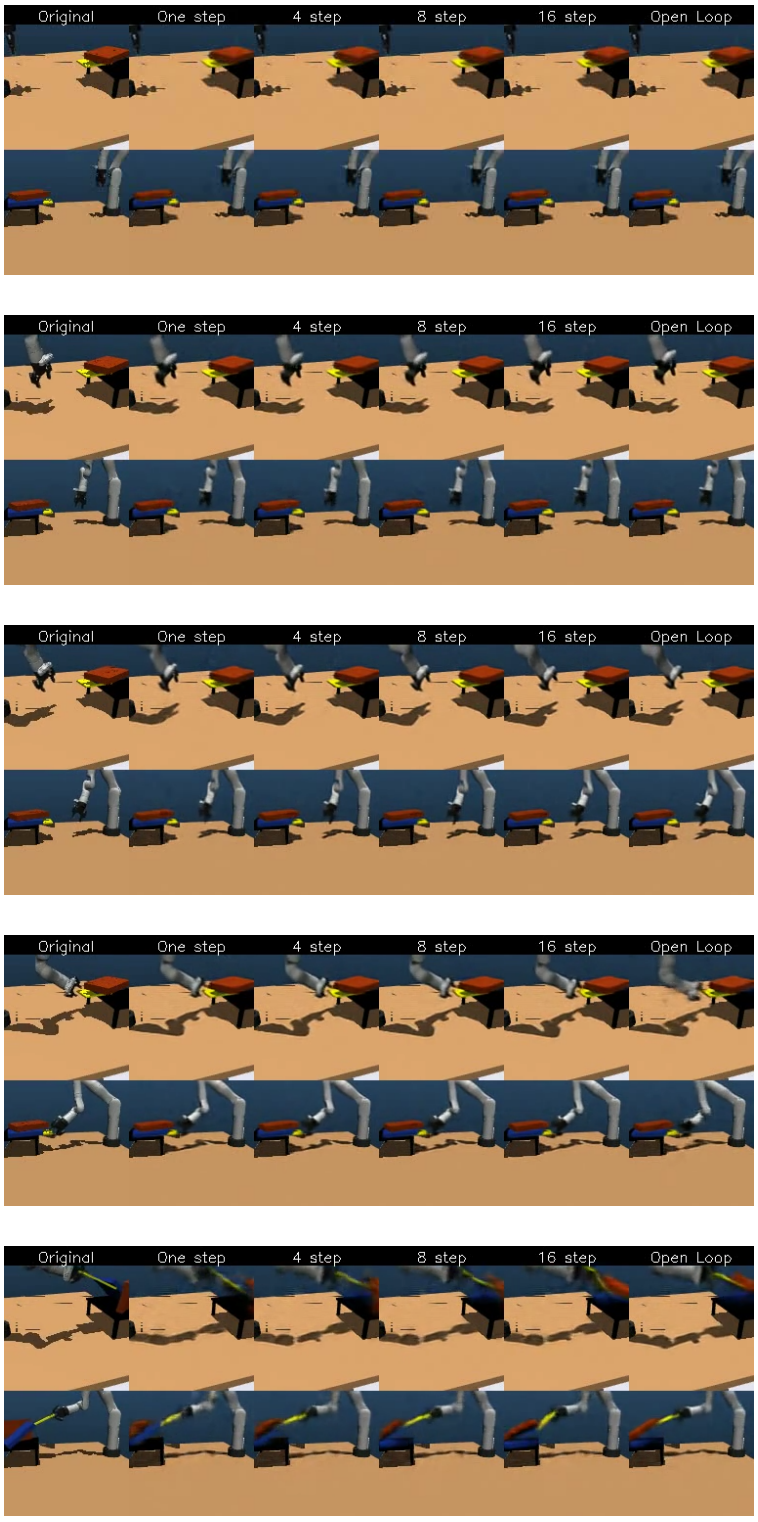

Figure 3.6: Arm Trajectories; 1 Step Predictions; Open Loop Predictions

margin maps the binary flags (from the state estimator) indicating that an object has fallen, to -1 or 1 depending on the outcome. This means that if any of the objects have fallen, the safety margin will be -1, otherwise it will be 1.

$$l(x) = 1 - 2 * \max(\text{book\_fallen}, \text{notebook\_fallen}, \text{encyclopedia\_fallen})$$

This algorithm was trained by using observation initializations from the collected dataset, and trained purely inside the world model without any simulation or real-world interaction. The hyperparameters were tuned to optimize the performance of the safety value function:

- **Actor, Critic model parameters:** Size: Linear (256 x 256); ReLU activations; learning rate: 3e-4.
- **Updates:** Timesteps: 100,000; Discount factor: Linear(0.98, 0.9995); Update every timestep

When rolled out, the learned safety shielded 78% of failure trajectories within in distribution rollouts, showing its viability for protecting against failures for the target task.

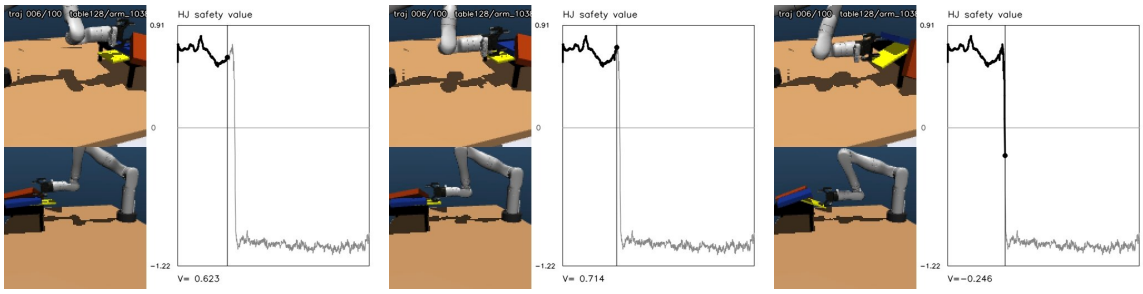

Figure 3.7: Value Function of the Arm in Different Time Steps

Figure 3.7 shows a case study of the safety value function over different time steps. Note, the frame separation is not uniform, as the focus is on the changes in the value function over time. The figure shows the safety value function predicting the safety outcome of the manipulation task at each time step.

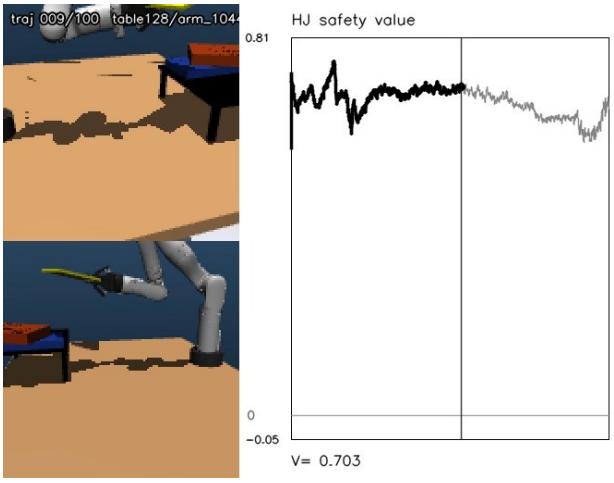

Figure 3.8: Value Function for the Pickup Task

Figure 3.8 shows the value function for the pickup task always predicting a positive safety value, when the object was successfully picked up.

However, there were a few instances where the value function did not correctly predict the safety outcome.

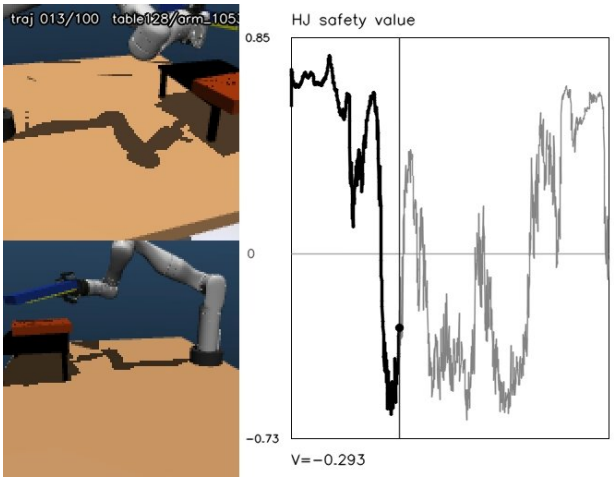

Figure 3.9: False Alarm in the Value Function for the Pickup Task

Figure 3.9 shows a false alarm in the value function for the pickup task, where the function predicted a negative safety value even though none of the objects were in danger of falling. This however is a highly out of distribution rollout, since along with the notebook, the blue book was also picked up by the manipulator, which is likely why the false alarm was triggered.

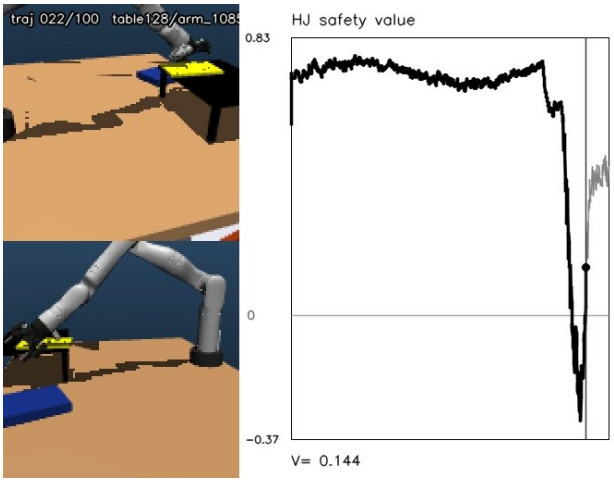

Figure 3.10: Forgetting the state of the fallen object

Figure 3.10 shows the value function recovering despite the fact that both the blue and red books have fallen in the rollout.

Chapter 4

Conclusion

4.1 Summary of Findings

This work investigated the use of Latent Safety Filters for improving safety in robotic control tasks by combining world models with Hamilton-Jacobi reachability-based reinforcement learning. Initial experiments on Half-Cheetah showed that a learned safety value function could successfully filter unsafe actions and prevent failures even when the nominal policy was random. However, the Antball experiments revealed an important limitation: the effectiveness of the safety filter depends strongly on state-action space coverage. When the nominal policy does not sufficiently explore safe recovery actions, the resulting filter may become overly conservative and miss task targets.

The world model experiments demonstrated that learning a compact latent representation is feasible for both a simple 2D navigation and more a complex manipulation setting. In the Dubins car task, the DreamerV3-based model produced more stable short-horizon and open-loop predictions than the earlier PlaNet-style implementation. Adding a state reconstruction head further improved the usefulness of the latent representation when some state information was available. To train the world model on manipulation tasks, a diverse set of training data was needed. This was collected by training a variety of nominal policies

on many permutations of reward functions with different objectives. These nominal policies were then rolled out stochastically to generate the required training data. After this, training the world model was easily accomplished using the framework created.

Finally, latent-space reachability was applied to the learned manipulation-task world model. The safety value function was trained entirely inside the world model, without additional simulator or real-world interaction. When evaluated on in-distribution rollouts, the learned safety shield prevented 78% of failure trajectories. These results suggest that Latent Safety Filters are a promising policy-agnostic safety mechanism for robotic manipulation, while also highlighting the importance of diverse data collection, accurate world models, and sufficient state-space coverage for reliable safety filtering.

4.2 Future Work

A number of improvements can be made to the current approach to Latent Safety Filters.

4.2.1 Hardware Deployment

This project is contained within a MuJoCo simulation environment, however the ultimate goal is to deploy the safety filters on real hardware. This will require some work on Sim2Real transfer as well as incorporating vision related algorithms to create a digital twin of the real-world setup (primarily the scene objects).

4.2.2 World Model Improvements

The state estimation component of the world model improved the overall performance of the world model, however it also introduced additional complexity in terms of training, and may increase the risk of overfitting, since it steers the training towards a specific type of state representation rather than a pixel-based representation. In fact, while pixel-based representations are more general and can capture a wider range of visual features, they also suffer

from overfitting to visual details. Joint Estimation and Prediction Architectures (JEPAs) are a promising approach to learn a latent state representation that encodes meaningful dynamics information [13].

4.2.3 Diverse Nominal Policy Training

One of the core issues with this thesis was the lack of diversity in the nominal policies used for training the safety filters. The dataset consisted of roughly equal distributions of trajectories from a limited set of nominal policies (¡10). These nominal policies also tended to be overly conservative, and displayed many success cases, causing the World Model to predict a limited range of possible outcomes. Improving this is an open research problem in Reinforcement Learning, however a promising approach is to fine tune these nominal policies with a diverse set of rewards to keep the wider range of policies that can be found when the policy function is continuously updated.

Chapter 5

Acknowledgment

This project is advised by Prof. Jason Choi (ECE) and Prof. Anushri Dixit (MAE). The Arm Software subgroup in Prof. Choi's lab consists of Jiandong Gu (MS ME), Jessie Shu (BS ME), Chelsea Chan (BS CE) and Sanjit Sarda (BS EE). The Arm Software subgroup contributed to the development of the MuJoCo simulation environments including reward design and implementation. Additionally, they supported the ongoing development and maintenance of the Kinova Gen3 7DOF arm in the lab, including Sim2Real transfer for the table top manipulation task.

Bibliography

[1] K.-C. Hsu, H. Hu, and J. Fern'andez Fisac, "The safety filter: A unified view of safety-critical control in autonomous systems," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 7, pp. 47–72, 2024.

[2] I. M. Mitchell, A. M. Bayen, and C. J. Tomlin, "A time-dependent hamilton–jacobi formulation of reachable sets for continuous dynamic games," *IEEE Transactions on Automatic Control*, vol. 50, no. 7, pp. 947–957, 2005.

[3] S. Bansal, M. Chen, S. L. Herbert, and C. J. Tomlin, "Hamilton–jacobi reachability: A brief overview and recent advances," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, 2017, pp. 2242–2253.

[4] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs for safety critical systems," *IEEE Transactions on Automatic Control*, vol. 62, no. 8, pp. 3861–3876, 2017.

[5] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control barrier functions: Theory and applications," in *2019 18th European Control Conference (ECC)*, 2019, pp. 3420–3431.

[6] K. P. Wabersich, A. J. Taylor, J. J. Choi, K. Sreenath, C. J. Tomlin, A. D. Ames, and M. N. Zeilinger, "Data-driven safety filters: Hamilton–jacobi reachability, control barrier functions, and predictive methods for uncertain systems," *IEEE Control Systems Magazine*, vol. 43, no. 5, pp. 137–177, 2023.

[7] K. Nakamura, L. Peters, and A. Bajcsy, "Generalizing safety beyond collision-avoidance via latent-space reachability analysis," in *Robotics: Science and Systems (RSS)*, 2025.

[8] J. F. Fisac, N. F. Lugovoy, V. Rubies-Royo, S. Ghosh, and C. J. Tomlin, "Bridging hamilton–jacobi safety analysis and reinforcement learning," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 8550–8556.

[9] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap, "Mastering diverse control tasks through world models," *Nature*, vol. 640, pp. 647–653, 2025.

[10] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, "Learning latent dynamics for planning from pixels," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 97. PMLR, 2019, pp. 2555–2565.

[11] N. Morihira, A. Nahar, K. Bharadwaj, Y. Kato, A. Hayashi, and T. Harada, "R2-dreamer: Redundancy-reduced world models without decoders or augmentation," in *The Fourteenth International Conference on Learning Representations*, 2026. [Online]. Available: https://openreview.net/forum?id=Je2QqXrcQq

[12] N. Morihira, "R2-dreamer: Implementation of r2-dreamer," https://github.com/NM512/r2dreamer, 2026, gitHub repository. Accessed: 2026-06-14.

[13] M. Assran, Q. Duval, I. Misra, P. Bojanowski, P. Vincent, M. Rabbat, Y. LeCun, and N. Ballas, "Self-supervised learning from images with a joint-embedding predictive architecture," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[14] S. Zhan, Y. Wang, Q. Wu, R. Jiao, C. Huang, and Q. Zhu, "State-wise safe reinforcement learning with pixel observations," in *Proceedings of the 6th Annual Learning for Dynamics and Control Conference*, ser. Proceedings of Machine Learning Research, vol. 242. PMLR, 2024, pp. 1187–1201.