Safe Reinforcement Learning for Robot Maze Escaping

Rocca Federico

Semester project

Under the supervision of:
Tingting Ni, Kai Ren and Prof. Maryam Kamgarpour

June 24, 2025

sycamore lab
SYSTEMS CONTROL AND MULTIAGENT OPTIMIZATION RESEARCH

École Polytechnique Fédérale de Lausanne (EPFL)
1015 Lausanne, Switzerland

Abstract

This project proposes a structured investigation of safe reinforcement learning (RL) in the scenario of autonomous maze traversal. This exploration is based on the implementation of a Lagrangian-PPO framework. The architecture was crafted and tested within the NVIDIA IsaacLab's simulation platform, and it was then deployed on a JetBot robot managed by ROS2. The primary objective of this research was to devise algorithms with the ability to not only reach predefined destinations, but to also consider the safety of the behavior, by staying away from obstacles. To address this task, we redefined the navigation challenge as a Constrained Markov Decision Process (CMDP). We modified the expected discounted rewards in order to ensure compliance with safety measures, which were represented through costs associated with obstacles or overall safety probabilities, and we implemented a dual formulation to consider the safety constraints. Our contributions include a Lagrangian model that merges per-environment costs to refine policies and a dual-updating mechanism based on global safety probability, on top of the implementation of a complete safe RL pipeline. Moreover, we carried out comprehensive empirical evaluations experimenting with different reward configurations, diverse strategies for balancing costs and probabilities, and methodologies to encourage exploration for navigation around local minima caused by obstacles.
Within IsaacLab, we leverage 4096 parallel JetBot environments on GPU to accelerate policy learning. We base the policies on observations that can potentially be obtained from onboard sensors. We find that using cost for policy updates converges faster but requires careful tuning of weight and constraint when used for the dual update, while safety probability provides as an intuitive global indicator of constraint satisfaction for updating the dual multiplier, while it is not informative enough for policy updates. A hybrid formulation, cost for the primal and safety probability for the dual, achieves both fast convergence and high safety ($\approx$ 99% of trajectories are collision-free).
For sim-to-real transfer, we export the trained policy to a ROS2 node and use it for inference by providing real observations. This phase proved more challenging than anticipated, as the discrepancies between simulated and real environment didn't allow a smooth transition. Qualitatively, the real JetBot has relatively low success rates and doesn't show safe behaviors, we believe because of differences in action space, control frequency and physical differences.
Finally, we discuss remaining challenges, such as action space alignment, and outline future work on multi-agent safe navigation.
Overall, this work demonstrates that a carefully designed Lagrangian-PPO algorithm, combined with IsaacLab's parallel simulation can yield collision-free,

goal-directed navigation policies, achieving the objective of safe-RL. Some improvement is necessary for allowing a smooth and effective transfer to physical robots.

Contents

Chapter 1

Introduction

1.1 Motivation and Real-World Scenarios

Autonomous mobile robots are increasingly deployed in environments for which no prior map is available, such as disaster zones, dynamic warehouses, and extraterrestrial terrains, yet must nonetheless navigate safely among obstacles. In these settings, collisions can cause costly damage to both the robot and the infrastructure, and manual mapping is infeasible or dangerous. A possible adaptable solution could be a learning-based navigation policy that can be trained in simulation and then transferred to real hardware. However, ensuring safety, meaning collision avoidance, throughout the deployment, without any preexisting map, presents a fundamental challenge in reinforcement learning (RL), and must be tackled in the training stage.

1.2 Project Objectives

This work develops and evaluates a constrained Markov Decision Process (CMDP) for robot navigation in environments with multiple obstacles, formulated in such a way that it can be applicable when no map of the environment exists. Our primary goals are:

1. Formulating the goal-directed navigation task as a CMDP in which collisions result in a cost and reaching the goal yields a reward.
2. Deriving a Lagrangian primal-dual algorithm, built on top of Proximal Policy Optimization (PPO), that jointly maximizes cumulative reward and enforces a user-specified upper bound on expected collision cost.
3. Implementing parallel training in NVIDIA IsaacLab using the SKRL library, then transfer the learned policy to a real JetBot through a ROS2 node.
4. Evaluating the trade-off between task performance and safety under various reward structures, constraint formulations and dual-update approaches.

1.3 Problem Statement

The problem is formulated as a Constrained Markov Decision Process (CMDP) $(\mathcal{S}, \mathcal{A}, P, r, c, \gamma, \rho)$, where we have continuous state $s \in \mathcal{S}$ and action space $a \in \mathcal{A}$, a reward function $r(s,a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ that mainly guides the robots towards the goal, and a cost function $c(s,a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ that quantifies collision cost when the robot infringes some safety margins around the obstacles. $P(s'|s,a)$ is the transition kernel, $\gamma \in [0,1]$ is the discount factor and $\rho : \mathcal{S} \to [0,1)$ is the initial distribution, defining the probability of spawning from each state.
The objective is to learn a policy $\pi_\theta(a|s)$, parametrized by θ, such that:

$$\max_{\theta} \quad J_R(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right] \tag{1.1}$$

$$\text{subject to} \quad J_C(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t c_t \right] \leq C_{max} \tag{1.2}$$

where C_{max} is an application specific cost threshold.
Introducing a nonnegative Lagrange multiplier λ we can reformulate the problem as a min-max optimization problem:

$$\min_{\lambda \geq 0} \max_{\theta} \mathcal{L}(\theta, \lambda) \quad \text{with} \quad \mathcal{L}(\theta, \lambda) = J_R(\theta) - \lambda (J_C(\theta) - C_{max}). \tag{1.3}$$

We solve this via primal-dual gradient ascent/descent:

$$\theta \longleftarrow \theta + \alpha_\theta \nabla_\theta \left[\hat{J}_R(\theta) - \lambda \hat{J}_C(\theta) \right], \tag{1.4}$$

$$\lambda \longleftarrow \max \left\{ 0, \lambda + \alpha_\lambda \left(J_C(\theta) - C_{max} \right) \right\} \tag{1.5}$$

where $\hat{J}_R(\theta)$ and $\hat{J}_C(\theta)$ are PPO's clipped stochastic estimates of the reward and cost objectives, and α_θ and α_λ define respectively the learning rate used in the policy parameters update and in the Lagrange multiplier update.

1.4 Background and Related Work

In safety-critical domains, a variety of constrained and safe reinforcement learning methods have been developed to enforce hard or probabilistic guarantees at training time. Tessler et al. [5] introduce RCPO, a multi-timescale primal approach that alternates between objective improvement and constraint satisfaction, providing convergence guarantees under nonconvex cost contraints. Lee et al. [2] experimentally compare five CMDP-based algorithms, including PPO-Lagrangian, IPO and CRPO, in quadrupedal locomotion, demonstrating that explicitly incorporating physical constraints leads to more robust sim-to-real transfers. Chen and Liu [1] propose a SSA+PPO framework in clustered dynamic environments, using a Safe Set Algorithm to monitor and correct nominal controls, ensuring safety and improving efficiency in obstacle-rich, reward-sparse tasks.
When it comes to bridging simulation and hardware, Salimpur et al. [4] present an end-to-end pipeline that trains local navigation policies in NVIDIA

IsaacSim and deploys them into Gazebo and real ROS2 robots. They demonstrate good performance in obstacle avoidance and local planning, highlighting how careful alignment of observation representation and action interfaces can yield reliable sim-to-real transfer without extensive domain randomization. This work informs our own deployment pipeline, with a PPO-Lagrangian policy trained in IsaacLab environments, exported as a PyTorch model and subjected to identical observation processing and action mapping on a real JetBot via a ROS 2 node.

1.5 Overview of the Approach

The proposed solution consists in four different components:

1. MDP design (Sections 2.1, 3.1): the observation vector contains information about the robot's position, it's pose relative to the nearest goal and the obstacles, while the action is a two-dimensional continuous vector of left and right wheel speeds. The reward has been shaped to reliably guide the robots to the goals.

2. Lagrangian-PPO implementation (Sections 2.2, 3.2): we extend PPO with a dual-variable update, that, after chunks of policy-gradient epochs, calculates the constraint cost across the multiple environments and modifies the value of the dual-variable λ via gradient descent, to adaptively give more or less importance to the constraint. We propose three different possibilities for implementing the constraint: a collision-cost constraint (1), a safety probability constraint (2), and a hybrid formulation (3).

3. Curriculum and ablations (Section 3.3): we consider the training in environments of different difficulties, different reward terms, and both cost-based and probability-based dual updates, evaluating the impact on convergence and safety.

4. Sim-to-real transfer (Chapter 4): the trained policy network is exported and integrated into a ROS2 node. Real-time observations, computed from the motion capture system, are fed through some preprocessing pipeline for collision and goal features extraction, enabling real world inference.

Together, these components enable us to train performing navigation policies that respect safety constraint during simulation learning, that can be deployed in real world, even if with some discrepancies in performance given by the sim-to-real gap.

Chapter 2

Problem Formulation and Approach

2.1 MDP Configuration

The robot navigation task is modeled as a (Constrained) Markov Decision Process, defined by the tuple $(\mathcal{S}, \mathcal{A}, P, r, c, \gamma, \rho)$, where:

- **State space** $\mathcal{S} \subset \mathbb{R}^{12}$: at each time step t, each agent observes a 12-dimensional vector

  $$s_t = (x_t,\, y_t,\, d_{G,t},\, \Delta\psi_{G,t},\, w_{L,t-1},\, w_{R,t-1},\, d_{1,t},\, d_{2,t},\, d_{3,t},\, \Delta\psi_{1,t},\, \Delta\psi_{2,t},\, \Delta\psi_{3,t}), \tag{2.1}$$

  with:

  - $(x_t,\, y_t)$: robot's position
  - $(d_{G,t},\, \Delta\psi_{G,t})$: Euclidean distance and heading error (as in yaw difference) from the nearest goal
  - $(w_{L,t-1},\, w_{R,t-1})$: previous left and right wheel angular velocities
  - $\{d_{i,t},\, \Delta\psi_{i,t}\}_{i=1}^{3}$: distances and angular offsets from the three closest obstacles

- **Action space** $\mathcal{A} \subset \mathbb{R}^2$: continuous commands for the robot's wheel angular velocities $(w_{L,t},\, w_{R,t})$. This allows to directly control the robots, without going through a differential drive mapping. The fact that the action space is continuous increases the complexity of the learning process, but will allow better control of the robot.

- **Transition kernel** $P(s'|s,a)$: represents the probability density of transitioning from state s to state s' under action a, determined by the physical model, with the rigid body and the wheel dynamics of the robot.

- **Reward function** $r(s_t, s_{t-1}, a_t) : \mathbb{R}^{14} \to \mathbb{R}$: a weighted sum of

  - Goal-reaching reward:

    $$r_{goal}(s_t) = R_{goal} \cdot \mathbb{1}(d_{G,t} \leq d_{reached}), \tag{2.2}$$

    with the weight R_{goal} and a small goal tolerance $d_{reached}$.

– Goal distance progress: a nonnegative measure of the progress towards the goal,

$$r_{\Delta d}(s_t, s_{t-1}) = R_{\Delta d} \cdot \max\{0, d_{t-1} - d_t\}, \tag{2.3}$$

with the weight $R_{\Delta d}$, to incentivize forward motion toward the target and to avoid a sparse reward if only the goal reward was left. The lower bound at 0 is set to allow the agents to backtrack in order to navigate around obstacles.

– Termination penalty:

$$r_{termination}(s_t) = P_{termination} \cdot \mathbb{1}(\|(x_t, y_t)\| \geq d_{max}), \tag{2.4}$$

with weight $P_{termination}$, that punishes the robots when they end up outside the boundary defined by the threshold d_{max}. Especially important for when the cost is introduced, as the robots might try to escape from the obstacles.

The total reward is then computed as:

$$r^{primal}(s_t, s_{t-1}, a_t) = r_{goal}(s_t) + r_{\Delta d}(s_t, s_{t-1}) + r_{termination}(s_t) \tag{2.5}$$

This formulation encourages moving towards the goal and reaching it, while avoiding the termination of the episode, but will also allow the exploration of the environment, which is especially helpful in overcoming local minima. To obtain the total reward term $\bar{r}(s_t, s_{t-1}, a_t)$, we combine $r^{primal}(s_t, s_{t-1}, a_t)$ with the cost term or the safety probability term, weighted by the Lagrange multiplier λ, to also consider the safety constraint in the reward term that the PPO algorithm uses to update the policy parameters.

- **Cost function**: a measure of the proximity to the obstacles, that, given the 3 closest obstacles at timestep t at distances $d_{1,t}$, $d_{2,t}$, $d_{3,t}$ and the minimum tolerated distance from the obstacles d_O, is computed as

$$c(s_t, a_t) = C_{obstacle} \cdot \sum_{i=1}^{3} \max\left\{0, 1 - \frac{d_{i,t}}{d_O}\right\} \tag{2.6}$$

- **Policy:** a diagonal-Gaussian policy

$$\pi_\theta(a \mid s) \;=\; \mathcal{N}\big(\mu_\theta(s), \operatorname{diag}(\sigma^2)\big), \tag{2.7}$$

where the mean $\mu_\theta(s)$ is produced by a feedforward network that takes as input the state vector $s \in \mathcal{S}$ and outputs the action vector $a \in \mathcal{A}$.

2.2 Safe RL Objective

The idea behind "Safe Reinforcement Learning" is to not only maximize a cumulative reward, as classical RL usually does, but also to enforce some safety constraints, here relative to the proximity to obstacles, in order to obtain policies less prone to dangerous behavior. This is achieved by defining the problem as a CMDP and solving it via Lagrangian relaxation.

2.2.1 Constraint Formulation

As expressed before (1.3), the problem is formalized by defining it as a constrained optimization. We consider the policy π_θ that acts in the CMDP, that collects the sequence of rewards $\{r_t^{primal}\}$, with $r_t^{primal}(s_t, s_{t-1}, a_t)$ defined in the MDP configuration (2.1). The expected return of the cumulative discounted reward over finite horizon H is defined as

$$J_R(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{H} \gamma^t r_t^{primal}\right]. \tag{2.8}$$

For the design of constraint, we consider it with two different approaches, (1) obstacle collision cost constraint and (2) safety probability constraint:

1. **Cost constraint**: as presented above (2.1), we defined a cost function that measures the proximity of the robots to the obstacles. Its value can be a good indicator of how safe the policy is. We have

$$J_C(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{H} \gamma^t c_t(s_t, a_t)\right]. \tag{2.9}$$

2. **Safety probability constraint**: we can also directly compute the probability of safety of the trajectories generated by the policy and constrain it to be higher than a desired value. The probabilistic constraint becomes

$$J_C(\theta) = \mathbb{E}_{\pi_\theta}\left[\prod_{t=0}^{H} \mathbb{1}_{\tau \in safe}\right]. \tag{2.10}$$

   To tractably reformulate (2.10) and enable Markov policy [3], we consider a binary collision indicator for each environment in the training setup, that for the $i-$th environments corresponds to

$$\mathbb{1}_{\tau \in safe}^{(i)} = \begin{cases} 0, \text{ if env. } i \text{ has collided in its trajectory } \tau, \\ 1, \text{ otherwise,} \end{cases} \tag{2.11}$$

   consistently maintained until the reset of the environment. At any time we can compute the global safety probability, across the N environments, as

$$P_{safe}(\pi) \approx \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}_{\tau \in safe}^{(i)}. \tag{2.12}$$

   It's important to notice a subtle but crucial difference in our safety probability approach. While constrained optimization typically aims to minimize a constraint to stay below an upper bound, we instead maximize a lower bound for the constraint, as the goal is to increase the safety probability. This seemingly inverse objective is mathematically sound, as our formulation can be derived by a simple change of variables from a standard minimization of an "un-safety" probability constraint, where $P_{unsafe}(\pi) = 1 - P_{safe}(\pi)$.

We then pose the primal constrained optimization as:

$$\max_{\theta} J_R(\theta) \;\text{ subject to }\; J_C(\theta) \leq C_{max}. \tag{2.13}$$

A key observation, that will influence how the strategies perform, is that while the cost constraint can be considered independently between each environment, since each one of them collects a different value for the cost function, the safety probability constraint can only be enforced globally, given that to estimate the value of probability (2.12) with high confidence level we require more collections of environments compared to the cost values that can be individual.

2.2.2 Dual Problem and Lagrangian Relaxation

To solve the primal constrained optimization (2.13), we form the Lagrangian

$$\mathcal{L}(\theta, \lambda) = J_R(\theta) - \lambda(J_C(\theta) - C_{max}), \tag{2.14}$$

by introducing the Lagrange (dual) multiplier λ. The safe-RL objective can then be defined as a min-max saddle point problem:

$$\min_{\lambda \geq 0} \max_{\theta} \mathcal{L}(\theta, \lambda) = \min_{\lambda \geq 0} \max_{\theta} \{J_R(\theta) - \lambda(J_C(\theta) - C_{max})\} \tag{2.15}$$

The min-max problem is solved by implementing the following routine:

Algorithm 1 Primal–Dual PPO-Lagrangian Update (Part 1)

Require: maximum steps *MaxSteps*, chunk steps H, rollout length T_{rollout}, dual learning rate α_λ, constraint limit $C_{\max}$

$\theta \leftarrow \theta_0$ ▷ initial policy parameters
$w \leftarrow w_0$ ▷ initial value-network parameters
$\lambda \leftarrow \lambda_0$ ▷ initial dual multiplier
Steps $\leftarrow 0$

while *Steps* $<$ *MaxSteps* **do**
  ChunkSteps $\leftarrow 0$
  while *ChunkSteps* $< H$ **do**
    $\mathcal{T} \leftarrow \varnothing$ ▷ rollout buffer
    for $t = 1$ to T_{rollout} **do** ▷ collecting rollouts
      $s_t \leftarrow$ Env.current_state()
      $a_t \sim \pi_\theta(\cdot \mid s_t)$
      $(s_{t+1}, r_t, c_t, \text{done}) \leftarrow \text{Env.step}(a_t)$
      Append $(s_t, a_t, r_t, c_t, done)$ to $\mathcal{T}$
      ChunkSteps $\leftarrow$ *ChunkSteps* $+ 1$
      if *done* **then**
        Env.reset()
      end if
    end for

Primal–Dual PPO-Lagrangian Update (Part 2, continued)

```
21:        θ ← PPO(θ, T)                                  ▷ Primal update
                                                  run PPO with rollout buffer T
                                                  to update old policy θ
22:    end while
23:    Steps ← Steps + ChunkSteps
24:    C ← Env.current_constraint_cost()
25:    λ ← max(0, λ + α_λ (C − C_max))                    ▷ Dual update
26: end while
```

We alternate:

1. **Primal policy update** via PPO, using the clipped surrogate

$$\hat{J}_R(\pi_\theta) - \lambda \hat{J}_C(\pi_\theta), \tag{2.16}$$

where θ are the policy parameters and $\hat{J}_R$ and $\hat{J}_C$ are empirical estimate over a rollout batch $\mathcal{T}$. We remove the constraint threshold C_{max} from the formulation of the Lagrangian (2.14) when we use it for policy update with PPO (2.16), as this variable does not depend on θ and is constant.

2. **Dual multiplier update** with gradient ascent on λ:

$$\lambda \longleftarrow \max\left\{0, \lambda + \alpha_\lambda \left(C - C_{max}\right)\right\}. \tag{2.17}$$

This step is considered in the two formulations, with cost and safety probability constraints:

- **Cost constraint**: in this case, at time H when the dual update is performed, we can consider the average cost between all the N environments at the current step, $C_H = \frac{1}{N}\sum_{i=1}^{N} c_H^{(i)}$, and compare this value with the desired constraint threshold. Due to the elevated number of environments that are considered, taking the cost function values only at the time of the dual update is informative enough about the safety of the policy to obtain good results. Considering instead the value of the cumulative cost would require more careful tuning of the constraint value, based on the length of the finite horizon H considered.
  We increase the value of the Lagrangian multiplier λ if the average cost is too high, giving more importance to the constraint in the next policy updates, and we decrease it when it is too low, to allow the policy updates to also focus on the other objectives.
- **Safety probability constraint**: we simply consider the value of the global safety probability P_{safe} (2.12), shared between all environments, at the time of dual update.
  We increase the dual variable λ when the value of this probability is not high enough, equivalently to increasing it when the "un-safety" probability is too high, and we decrease the value for the dual variable when the constraint is satisfied.

When updating the dual variable, we no longer face the distinction between the fact that the cost function can be evaluated for each individual environment, while the safety probability can only be a global indicator. Because the Lagrangian multiplier λ is updated based on how the constraint is satisfied across all the environments, and is considered as a global variable, we simply average the per-environment cost to produce an overall cost signal, whereas the safety probability already provides a global indicator by construction. In either case the dual update uses a global scalar value to drive λ toward enforcing the desired constraint.
We perform H PPO epochs per batch to update the policy parameters θ, and then one dual step to update the multiplier λ, leading to a stable convergence to a policy that both maximizes the reward and respects the constraint.

Chapter 3

Simulation-Based Learning

In this chapter, we describe how we built and leveraged a simulation pipeline to develop, test, and refine our safe navigation policies. All of the code and configuration files described here are available in our public repository at `https://github.com/federock02/EPFL-SemesterProject1`.

3.1 Simulation Environment Design

To validate our Lagrangian-PPO approach in a controlled setting, we first develop two progressively more challenging simulators: a simple goal-centered arena to teach basic navigation (3.1.1), first without and then with added obstacles, and the full maze environment matching our real-world testbed (3.1.2). We then describe the 12-dimensional observation vector used in both cases.

3.1.1 Simple Goal-Centered Environment

In the initial phase, agents are trained in a circular arena to isolate goal-seeking behavior. Each episode begins with the robots uniformly sampled within a spawning circle of radius $r_{spawn} = 5\,m$, and a single goal is fixed at the world origin $(0, 0)$. Episodes terminate either when the robot crosses an outer radius $r_{termination} = 7\,m$ or upon goal achievement. A penalty applies for excursions into the annulus $5\,m < \|(x, y)\| \leq 7\,m$: upon crossing $\|(x, y)\| > 5\,m$, the agent immediately receives a termination penalty, but gets reset only after surpassing the $7\,m$ radius. Initially, this environment contains no obstacles and serves to validate the base PPO implementation. We then augment it with four static walls around the goal to test our safe-RL framework in a setting still relatively simple (Figure 3.1). The reward and cost structure considered in this setup are the same as presented in the MDP (2.1).

3.1.2 Maze Environment

To bridge toward realistic navigation, we adopt the exact maze geometry used in hardware tests (Figure 3.2). The world is a rectangular arena ($3\,m \times 6\,m$) enclosed by boundary walls with five exits, plus internal partitions forming the

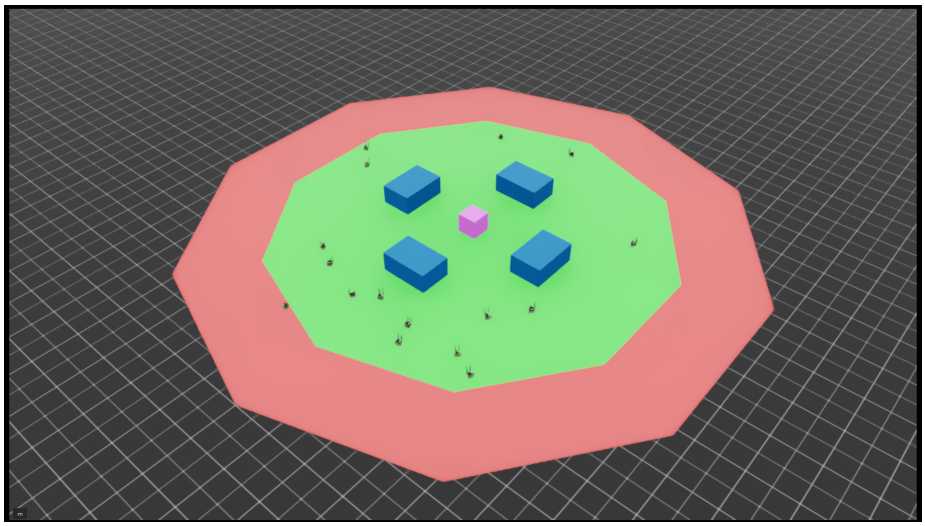


Figure 3.1: Simple goal-centered environment, with obstacles in blue, termination penalty ring in red and goal in pink

maze corridors. Wall thickness and exit gaps match the real robot's testbed. In this setup we consider the exits of the maze as the goal points.

- Spawn region: robots are initialized uniformly at random within the free floor area, subject to a minimum distance of $0.15\,m$ from any wall.
- Termination: exiting the arena beyond a circle of radius $3.5\,m$ around the center triggers episode end with a negative penalty.
- Reward structure remains as in the simple environment, encouraging both efficient goal approach, efficient navigation and constraint satisfaction.

3.1.3 Observation Engineering

At each time step, the agent receives a 12-dimensional observation vector designed to be realizable on a physical JetBot robot:

$$o_t = (x_t,\, y_t,\, d_{G,t},\, \Delta\psi_{G,t},\, w_{L,t-1},\, w_{R,t-1},\, d_{1,t},\, d_{2,t},\, d_{3,t},\, \Delta\psi_{1,t},\, \Delta\psi_{2,t},\, \Delta\psi_{3,t}) \in R^{12}, \tag{3.1}$$

where each component corresponds to a variable defined in the MDP state in (2.1). As explained before, the goals correspond to the maze's exits, so the closest goal is the closes exit. All features have been chosen to be directly computable from on-board odometry, range sensors (e.g. LiDAR or depth camera), and IMU. This ensures a smooth transition from simulation to real robot deployment, aligning the observation space with what is physically measurable on hardware and facilitating the policy transfer.

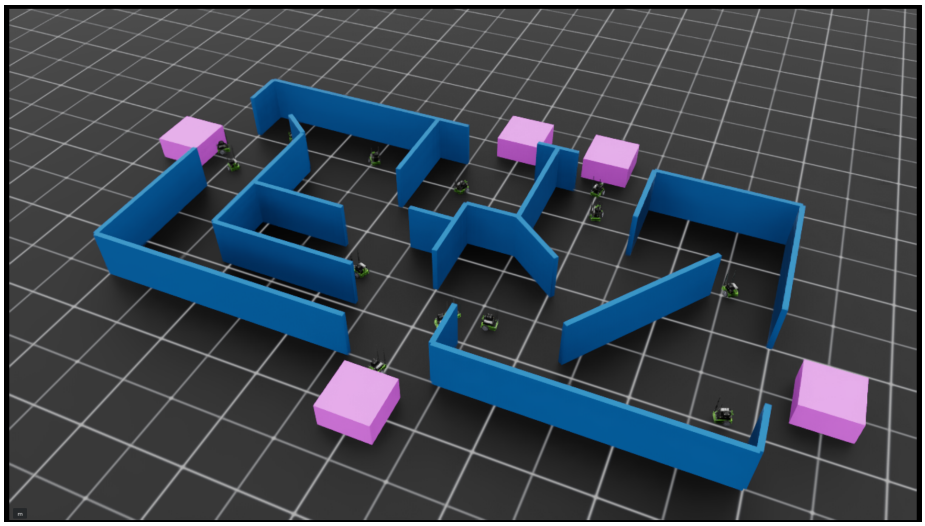


Figure 3.2: Maze environment, with maze walls in blue and pink boxes marking the goal positions at the exits

3.2 PPO-Lagrangian Algorithm Implementation

In this section we detail the practical implementation of the safe-RL framework, with the primal-dual extension of the Proximal Policy Optimization (PPO) algorithm within the IsaacLab-SKRL simulation environment. We first describe how the standard PPO pipeline was augmented to accommodate the dual variable updates, ensuring the coupling between policy learning and constraint satisfaction. Next, we present the strategy used for the dual update, illustrating how the dual variable λ is adjusted in practice, based on observed constraint violations, considering both the approaches of cost constraint and safety probability constraint. Finally we discuss key hyperparameters, like frequency of dual versus primal updates, learning rates for both policy and multiplier, constraint threshold, and analyze their influence on learning performance.

3.2.1 Adaptation in IsaacLab and SKRL

Before diving into the custom extensions, it is helpful to understand the underlying simulation and RL infrastructure:

1. IsaacLab's high throughput simulation: IsaacLab leverages GPU acceleration to run thousands of environment instances in parallel. In our setup, for instance, we spawned 4096 parallel JetBot environments on a single NVIDIA GeForce RTX 4060 GPU, each one executing its own physics steps, observation gathering and action application. This massive parallelism allows rapid data collection by generating millions of state, action and reward samples, critical for on-policy methods like PPO.

2. SKRL's PPO implementation: the SKRL library provides a highly op-

timized, PyTorch based PPO agent that supports vectorized environments. At each policy iteration the agent collects a rollout of fixed length across the 4096 simulations, storing observations, actions and rewards in a single buffer. This buffer is then split into large mini-batches that are used for computing clipped surrogate updates in parallel, using GPU tensor operations for forward pass and gradient computations.

Together, IsaacLab's simulation power and SKRL's GPU-native PPO implementation offer a strong foundation upon which we can implement the Lagrangian primal-dual extensions. We extended both the environment interface and the PPO agent implementation:

- **IsaacLab's environment**: we implemented our `JetBotEnv` subclass to compute and store per step constraint cost, wether penalties for distance to obstacles or binary safety indicators, alongside the standard reward, and expose APIs for setting the updated Lagrange multiplier λ and for querying the current constraint values. At each reset, the environment zeroes all the necessary buffers; at each PPO step, the environment computes observations, applies actions and returns the calculated rewards; at each dual update it provides the value for the constraint.

- **SKRL runner integration**: instead of SKRL's out-of-the-box single phase training loop, we wrap its `Runner` in a custom `PPO_LagrangianWrapper`. This wrapper alternates between (i) running SKRL's usual PPO "learn for H timesteps" call and (ii) performing a dual update on λ. We pass the current λ into the environment at the start of each chunk so that the policy's reward signal is effectively $r_t^{\lambda} = r_t^{primal} - \lambda c_t$, where c_t is either the per-step cost or the global safety probability, as explained in (2.2.2). By setting `reset_num_timesteps=False`, we allow the PPO optimizer's internal schedule (learning rate decay, advantage normalization) to continue smoothly across chunks.

- **Policy and value networks**: leveraging the SKRL's implementation, we configured a single feed forward neural network with two shared hidden layers ($192 \longrightarrow 128$ units, ReLU activations). From the final shared representation we split into a `GaussianMixin` policy head (2.1), that produces a 2-dimensional mean vector and a learned log-std vector, defining a diagonal Gaussian over the left and right wheel commands, and a `DeterministicMixin` value head that outputs a single scalar $V(s)$, estimating the state value. The implementation with shared layers allows both heads to share the same feature extractor.

3.2.2 Dual Update Strategy and Hyperparameters

At the core of the Lagrangian approach is the update rule for the dual variable λ. After every chunk of H policy updates, we perform the dual update. The correct balance between the number of policy updates and multiplier updates is fundamental for obtaining an effective policy, in a reasonable training time, while satisfying the safety constraints. The magnitude of the change applied to the dual variable at update time, defined by the Lagrange learning rate α_λ, is also an important parameter, and influences how strictly the constraints will

Hyperparameters	Role	Typical Value
H	Chunk size, number of policy updates per dual update	200
α_λ	Lagrange multiplier learning rate	0.05-0.5
α_θ	Policy learning rate	0.0003
$C_{limit,cost}$	Constraint threshold (upper bound) for cost formulation	0.02-0.1
$C_{limit,safety}$	Constraint threshold (lower bound) for safety probability constraint	0.98-0.995
γ	Discount factor PPO	0.99
entropy loss scale	Measures the amount of stochasticity and consequently exploration	0.04-0.08
rollout steps	Number of steps collected for each environment before policy update	32
minibatch size	Size of minibatch used for policy update	256

Table 3.1: Table of hyperparameters utilized in the training process

be enforced. Typically a value too high will not allow the policy to optimize the navigation goals as it will be too focused on constraint satisfaction, and will cause oscillations in λ, while a value too low corresponds to a really slow convergence time. This value also needs to be selected by taking into consideration the scale of the constraint: the obstacle violation cost usually has values in the magnitude of $10e-2$, while the safety probability is in the $[0,1]$ range, so a careful tuning is necessary for ensuring a successful training. For the same reason, the choice of constraint threshold is extremely important; as the value of the safety probability is not tied to other variables, it has proven more intuitive to set, while the cost constraint must depend on the weight applied in the cost function, and is set by trial and error.
The Lagrange multiplier λ was initialized with a value of 0 at every training experiment; that is to allow the network to quickly learn an efficient goal-tracking policy before having to consider too much the constraint. Starting with a high value for λ doesn't allow the agents to learn basic navigation to the goals because the focus is on safety from the beginning.
Balancing the entropy scale was also necessary, as a value too low will lead to premature convergence, poor exploration around obstacles and often local minima, while a value to high impedes cost minimization.
PPO is relatively robust to modest hyperparameter variations, which significantly eased tuning in our case. In particular, we found that key settings like learning rate, clipping range, and batch size could each vary within a reasonable window without dramatically altering learning stability or final performance. Our key hyperparameters are presented in Table 3.1.
Together, these design choices and empirically tuned hyperparameters enable

a stable PPO–Lagrangian training process that balances the trade-off between reward maximization and safety constraint satisfaction.

3.3 Training Experiments

In this section we present an in-depth empirical evaluation of our Lagrangian-PPO framework, exploring how different reward components and strategies for handling the safety constraint interact to shape both safety and learning efficiency. We begin by showing the effects of different reward terms that we experimented with, and the final formulation that allowed us to obtain the best results, demonstrating how they mitigate local minima and guide the agents through the obstacle-filled environment. We then compare three distinct formulations used for enforcing the safety constraint: using the environment obstacle violation costs, using the aggregated safety probability, or using a hybrid formulation that leverages cost gradients in the policy update while driving the dual update with the global safety probability. Finally we synthesize the findings into some guidelines for implementing safe-RL.

3.3.1 Reward Structure Ablations

We compared several candidate reward formulations both in the obstacle-free arena and the more challenging maze to identify a formulation that balances goal approach and robust obstacle avoidance. The experiment aimed to accelerate navigation to the goal, improve overall navigation, and overcome the frequent local minima, particularly near obstacles. Here are the most significant experiments that were performed:

- Exponential distance reward: it was believed to help incentivize a fast navigation to the goal, as the reward increases exponentially with the decrease of distance, but it introduced a pathological behavior: as the agents moved closer to the goal, they begun spiraling around it to maximize the collection of the high reward terms.
- Delta-distance reward: despite the simple linear structure, it proved efficient in encouraging movement towards the goal, and was selected in the final formulation. In order to allow the agents to temporarily increase the distance from the goal to navigate around the obstacles, this term was clipped to 0 when its value was negative, so when the distance was increasing.
- Time penalty: it was introduce to discourage the agents from getting stuck and not moving to the goal as efficiently as possible. However the agents learned to exit the episode prematurely by crossing the termination radius to avoid cumulating the penalty, so it was discarded.
- Movement reward: it was implemented to solve the local minima, by rewarding a delta in the position between successive timesteps. This proved useless, as the agents learned to simply oscillate around the local minima, and does not appear in the final formulation.

Reward Term	Weight
Goal-reaching reward (R_{goal})	60
Goal distance progress reward ($R_{\Delta d}$)	10
Termination penalty ($P_{termination}$)	-50
Obstacles collision cost ($C_{obstacle}$)	6
Minimum distance (d_O)	$0.15\,m$

Table 3.2: Table of weights utilized in the reward and cost function

- Exploration reward and penalty: the introduction of this component helped successfully address the local minima problem. It enabled agents to discover policies that efficiently navigated around obstacles when encountered on the path to the goal. It was not included in the final reward function because, thanks to careful tuning, the simple formulation with goal, delta-distance, and termination rewards (2.1), already allowed us to train policies with good exploration and no local minima.

After multiple trials, the reward structure that was adopted is the one presented in the MDP formulation (2.1)

$$r^{primal}(s_t, s_{t-1}, a_t) = r_{goal}(s_t) + r_{\Delta d}(s_t, s_{t-1}) + r_{termination}(s_t), \tag{3.2}$$

with the weights presented in Table 3.2.
It should be noticed that the values proposed are specific for this implementation and this environment layout, and may need to be tuned in the case of different configurations. We provide the plots for the average values of goal reward r_{goal}, delta-distance reward $r_{\Delta d}$, termination penalty $r_{termination}$ and total primal reward r^{primal} in an example training run (Figure 3.3).

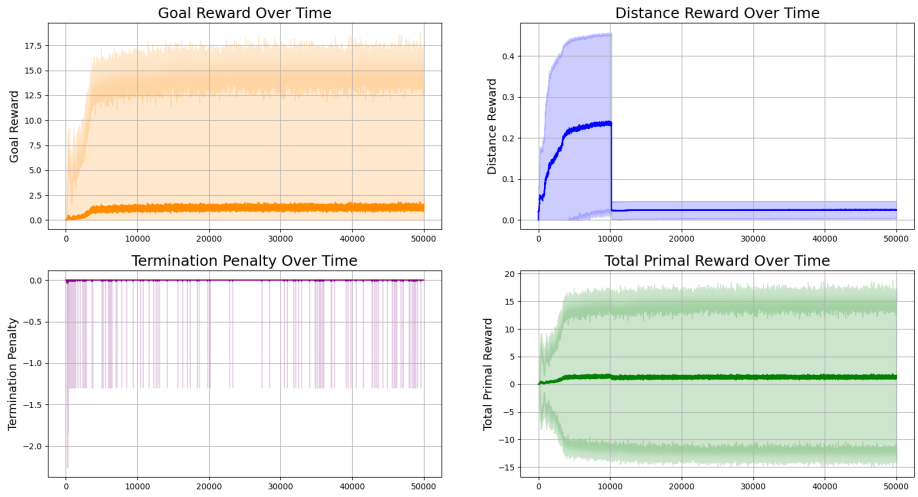


Figure 3.3: Evolution of reward components during training

3.3.2 Cost vs Safety Probability

The Lagrangian approach enforces safety by penalizing collisions with obstacles, but the choice of constraint signal greatly impacts the learning process and the sensitivity to the parameters. We compared three formulations:

1. **Cost only**: we compute the per-environment cost at each timestep, depending on the distance of each agent from the obstacles. In the IsaacLab implementation, this cost evaluation is stored in a tensor that is combined, after applying the Lagrange multiplier λ, with the primal reward tensor for performing policy update steps. At the time of dual update, the cost tensor is collected at the current step and averaged between the environments, to obtain a global constraint satisfaction signal that is used for changing the value of λ. From the plots (Figure 3.4) we can notice the evolution of the Lagrange multiplier λ, the mean cost and the safety probability during a training run with cost formulation. The convergence, indicated by the evolution of λ, is really quick as the safety constraint is soon satisfied.

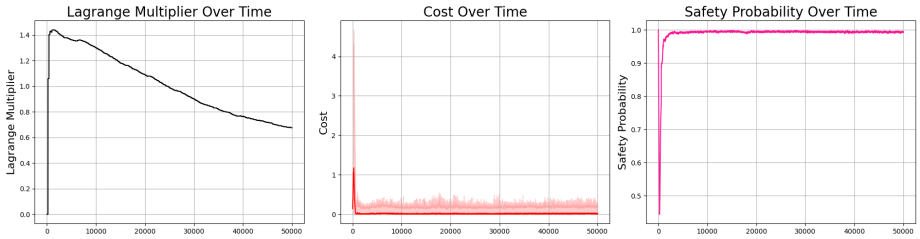


Figure 3.4: Evolution of λ, mean cost and safety probability for cost constraint formulation

Pros: the individual cost sample collected for each environment allows to define a fine-grained gradient that improves the performance of the policy optimization. The convergence to a good policy with safe behavior occurs quickly.
Cons: selecting the cost limit and the Lagrange learning rate used for the dual update requires tuning relative to other hyperparameters, like obstacle width and count, rollout size and obstacle violation weight, as they influence the values of the cost samples collected at each timestep. This can prove challenging and requires multiple trials.

2. **Safety probability only**: using the per-environment safety indicators we can compute at any time the overall safety probability. The policy update is performed by considering the total reward obtained by combining primal reward and the global "un-safety" probability, chosen for maintaining the correct dual formulation, scaled by the multiplier λ. The dual update depends again on the value of the global safety probability compared to the constraint threshold. From the plots (Figure 3.5) we notice how the Lagrange multiplier keeps increasing because the safety constraint is never satisfied, signifying that the policy is not converging to a safe behavior. That is because, when combining a per-environment

signal, the primal reward , with a global signal, the ”un-safety” probability, for performing policy updates, the gradient signals get degraded and loose part of the information. This leads to lower learning capabilities.

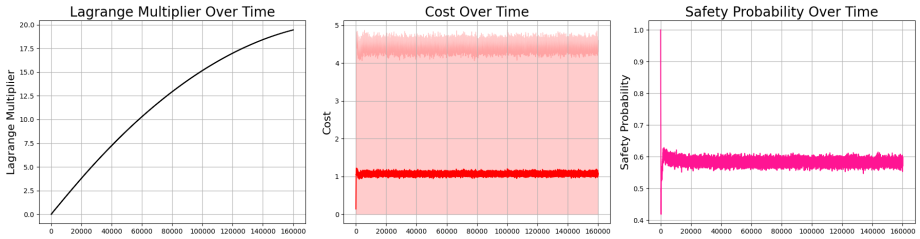


Figure 3.5: Evolution of λ, mean cost and safety probability for probability constraint formulation

Pros: The constraint limit is directly interpretable (e.g. require 97% of the trajectories to be collision free), and the Lagrange learning rate generalizes across different tasks without retuning.
Cons: The gradient signal used for policy updates doesn't include any per-environment cost information, only a scalar global signal, leading to unstable and slow learning, with difficulties in converging to a safe behavior.

3. **Hybrid**: the approach combines the use of per-environment cost in the policy update, providing informative individual gradients, and the implementation of the dual update driven by the constraint on safety probability. By pairing a precise, environment-specific cost signal for learning, with a stable, intuitive global safety measure for λ-updates, the hybrid method capitalizes on the benefits of each.
The choice of combining the safety constraints is justified by the fact that both metrics reflect how agents interact with obstacles. We computed Pearson's correlation coefficient r over multiple training episodes of between 20000 and 80000 PPO updates and found $r = -0.8937$ between the values logged at each step for mean cost and the corresponding safety probability (Figures 3.7). This strong negative correlation confirms that either signal could be used to measure constraint violation, but because safety probability is already a single, global metric, independent of environmental factors, it is more natural for the dual update. Meanwhile, cost remains a more informative per-environment feedback for the policy update. From the plots (Figure 3.6) we notice the fast convergence of λ, as the safety constraint is quickly satisfied.
Pros: the policy update is performed using informative gradients, with tailored cost signals for each environment; the dual update remains robust since the global safety probability is intuitive and stable under environmental changes, and the meaning of the safety constraint from which the λ-update depends is easy to understand. The convergence to a performing and safe policy occurs in a short time.

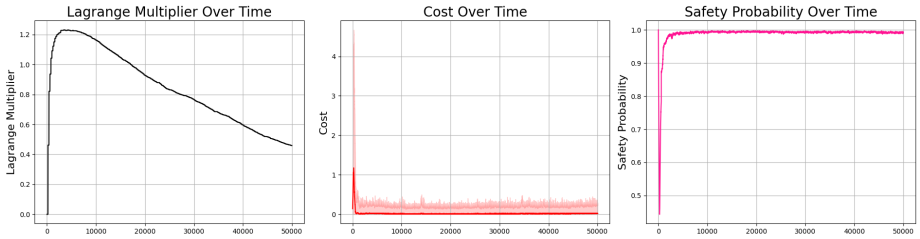
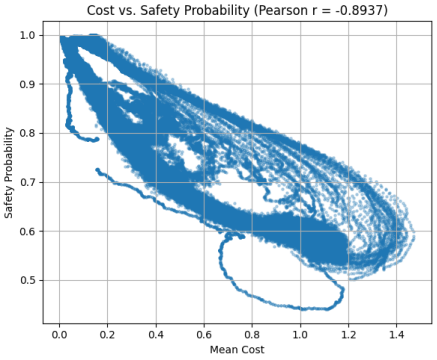
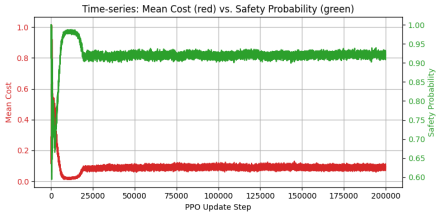


Figure 3.6: Evolution of λ, mean cost and safety probability for hybrid constraint formulation



(a) Cost vs. safety probability with Pearson correlation



(b) Time series of cost and safety probability for one training example

Figure 3.7: Images illustrating the correlation between cost and safety probability

3.4 Summary of Simulation Findings

Our extensive experiments yield three concrete takeaways:

- Careful tuning of the reward function, or an exploration term, is indispensable for overcoming local minima introduced by obstacles in the trajectory to the goal.
- Per-environment cost signal delivers an informative, localized feedback that drives precise avoidance behaviors and allows to obtain performing policies, but they demand careful hyperparameter tuning, based on the choice of other parameters, if used for the dual update.
- Global safety probability provides an intuitive, environment-independent metric for evaluating the safety of the overall policy and performing dual updates, but is not as meaningful as the cost function when it comes to finding a good signal used for policy update; if paired with the cost function, it can achieve both rapid convergence and robust safety.

These insights guided the final implementation of our Lagrangian-PPO agent, resulting in a success rate of 100%, as every agent manages to escape the maze, and $<$ 1% obstacle violation rates in the simulated maze environment. The

diagram (Figure 3.8) shows the trajectories implemented at test time by our best trained policy.

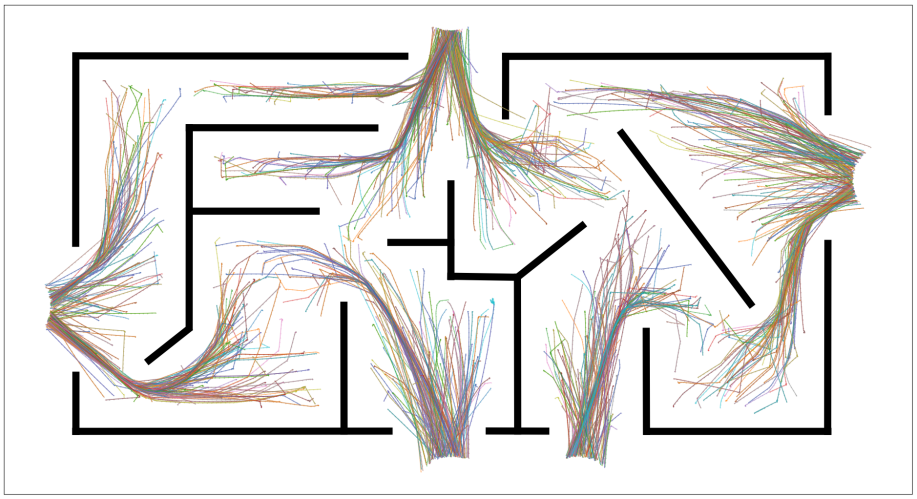


Figure 3.8: Trajectories at test time

Chapter 4

Sim-to-Real Transfer

In this chapter, we describe how the policy learned in simulation is exported, deployed, and executed on a real JetBot robot. We begin by outlining our deployment pipeline, covering policy export, the inference engine, real-time observation reconstruction, action mapping, and the runtime constraints imposed by the physical robot. We then present our transfer results, including empirical safety and success rate metrics, as well as qualitative observations of the JetBot's behavior in the real maze. Finally, we highlight key discrepancies between simulation and reality.

4.1 Deployment Pipeline

4.1.1 Policy Export and Inference Engine

After training in IsaacLab with SKRL and the Lagrangian-PPO framework, we save the final network parameters in a PyTorch ".pt" checkpoint file. In our ROS2 node, that will control the real robot, we reconstruct and exactly matching network architecture, a `GaussianMixin` based policy with a $12 \to 192 \to 128 \to 2$ linear architecture, and then load the stored parameters, mapping weights and biases to the correct layers and units. The model is then placed in evaluation mode and can be queried for inference by providing observation tensors. No further quantization or pruning is applied, as the network is already small enough to run in real time.
During deployment, the ROS2 node invokes the `policy.compute("states": observation_tensor)` method to feed the observation vector as an input to the network and produce a 2-dimensional mean wheel speed output and the learned standard deviation parameters. At inference time we simply take the mean value as action outputs. All inference is performed on the lab's central workstation; from there the ROS2 node publishes the commands on the JetBot's `/cmd_vel` topic, that ultimately drives the onboard motor controllers.

4.1.2 Real-Time Observation Reconstruction

In simulation, IsaacLab provides perfect state information; in the implementation on hardware in the lab we rely on the OptiTrack motion-capture system

and stored information about the JetBot's wheel speeds. Every 0.2 seconds the ROS2 timer callback will perform the following steps to reconstruct the 12-dimensional observation vector:

- **Pose and yaw from mocap:** the OptiTrack system publishes the JetBot's polition (x, y) and orientation quaternion (q_x, q_y, q_z, q_w) at a $5\,Hz$ rate. The position is taken as is, while the quaternion is used to compute the yaw angle, needed for determining the angle difference relative to goal and obstacles.
- **Wheel speed history:** for each new action output from the network, wheel speed values are processed and stored to serve as observations for the subsequent timestep. This method assumes that the commanded and effectively measured wheel speed values will be identical.
- **Distance and angle to goal:** the Euclidean distance from the nearest goal, and the angle difference between robot's heading and line between robot and goal, are computed based on the pose obtained from the motion capture system and the known goal positions.
- **Distance and angle to obstacles:** with the same logic as in the simulated environment, the three closest obstacles are selected and the distance and angle difference are computed based on robot's pose and obstacle's position and shape.

All computation occurs in the ROS2 node, matching the simulation implementation. Even thought this specific implementation relies on the motion capture system, it could potentially be realized with onboard sensing only, from odometry, range sensors and IMUs.

4.1.3 Action Mapping

In simulation, IsaacLab's physics engine accepts raw, unbounded angular velocities from the PPO policy and automatically enforces the actuator limits specified in the robot's configuration file. These limits are handled internally by clamping or scaling the motor commands in a smooth and idealized way, abstracting away the complexities of real-world motor dynamics.
On the real JetBot platform, however, no such implicit actuator handling is available. The PPO policy produces unbounded real-valued outputs representing the desired angular velocities of the left and right wheels. These values must be transformed into bounded motor commands compatible with the JetBot's low-level motor driver, which requires normalized control inputs. In the deployed ROS2 inference node, this transformation consists of optional clipping and rescaling, followed by a conversion into motor commands published to the `/cmd_vel` topic.
Despite efforts to match the behavior between simulation and reality, a significant sim-to-real gap remains. A major contributing factor is the presence of actuator deadzones in the real robot, which are not modeled in simulation. Real motors exhibit static friction that must be overcome before rotation begins, leading to a range of control values where no motion occurs. This phenomenon is modeled through piecewise affine functions for the wheel input-to-velocity relationship, with deadzone thresholds, and gain coefficients

estimated through system identification.
Since IsaacLab's simulation environment does not natively simulate motor deadzones or non-linear control curves, policies trained in simulation are unaware of the effort required to overcome static friction in reality. As a partial remedy, we experimented with manually applying output clipping to the policy before the physics step during training, so that the trained network would learn to operate within bounded command ranges. During inference, the same clipping or scaling was applied before passing the commands to the JetBot. Unfortunately, this approximation still leaves discrepancies, both in fine-grained control near zero velocity and in large and fast movements, and has not yet fully bridged the gap between simulated and real behavior.
Addressing this gap would require incorporating a deadzone-aware dynamics model either during training or through a post-processing module that explicitly accounts for the identified piecewise actuator models.

4.2 Transfer Results and Observations

4.2.1 Safety and Goal Success Rate

To evaluate how well the policy transfers from simulation to the real maze, we performed multiple trials on the physical JetBot. In each trial we place the robot in a random position within the maze, and launch the ROS2 node to control it. The network controllers chosen for the trials were the ones that showed the best performance in simulation, with efficient navigation and near-perfect safety rates. We evaluate:

- Goal success rate: a trial is considered successful if the JetBot reaches within $0.2\,m$ of a goal. The success rate is lower than in simulation and is around 75%, meaning that 1 in 4 times the robot fails to navigate to an exit.
- Safety rate: we define safety as the absence of violation of obstacles. As the maze in the lab is made by marking the positions of the walls with tape on the ground, we can empirically see if the robot goes through an obstacle in its navigation. The safety rate in the real environment is virtually 0, as the robot violates the obstacles almost every time, apart from some fortunate initial positions. The discrepancy with the implementation of the same controller in the simulated environment is enormous, as in IsaacSim the safety rate was around 99%, and we believe this is due to the sim-to-real gap in action space and control frequency.
- Failure modes: when the JetBot fails, it typically falls into two categories: (i) the robot gets stuck inside the maze in some local minima that sometimes appear in the real implementation, even though they were solved in the simulated environment, or (ii) the robot completely fails to track the goal and exits in random directions.

4.2.2 Qualitative Behavior, Discrepancies and Insights

Across nearly all real world trials in which the robot successfully reached the goal, it failed to satisfy the safety constraint by traversing over one or more

obstacles. Visually, the robot's trajectory approximates the correct strategy to go around the obstacles, as the path loosely resembles the ideal safe path that would be chosen in simulation, but this is not enough for guaranteeing a safe behavior (Figure 4.1). Moreover, the control is really jerky and not efficient, with frequent wheel slippage and unnecessary movements. This, again, is a consequence of the sim-to-real gap that arises from the following discrepancies between simulated and real environment:

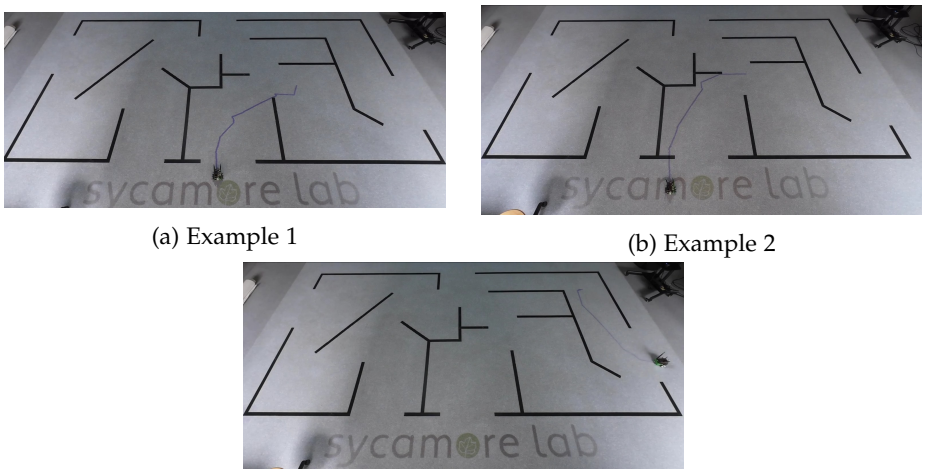
- Action space mismatch: in simulation, the raw wheel speed commands, that are produced as output from the network, are managed and processed by IsaacLab's actuator model. On the real robot we found difficult to extrapolate the correct mapping from policy output to motor command, and, as a result, the policy's intended velocities cannot be faithfully executed in hardware. Commands that appear safe in simulation translate into wheel slip or over-rotation on the actual robot, causing it to have unsafe behavior.

- Control frequency discrepancy: the ROS2 node runs at $5\,Hz$, so the policy was trained by having IsaacLab iterate over the steps at the same frequency. But, because of the internal implementation in the simulation and the delays that occur in the real implementation, and because of inference, communication and observation computations, the JetBot receives delayed commands. When the policy attempts rapid course adjustments, those adjustment may arrive too late, resulting in jerky movements and obstacle collisions.

- Wheel slip and friction differences: in simulation, the floor friction is ideal. In the real maze, as the robot attempts sharp turns or accelerations, the wheel frequently slip, causing the robot to drift into obstacles, despite the policy's correct commands. This continuous slipping forces the JetBot to make unnecessary back-and-forth movements that do not occur in simulation.

- Sensor noise and estimation error: IsaacLab provides perfect, noise-free measurements at each simulation step. In contrast, our OptiTrack produces poses that can be noisy. Those discrepancies lead to jerky feedback loops and errors in the navigation, that further reduce safety.

In summary, although the learned policy has a general understanding of how to move around obstacles, these real world mismatches conspire to produce faulty navigation and unsafe trajectories. Even when the JetBot reaches the goal, its path is far less smooth and optimal than in simulation.



(a) Example 1

(b) Example 2

(c) Example 3

Figure 4.1: Images of paths navigated by the robot in the real world environment; it can be seen how, even though a basic idea of the optimal path is present, the movement is jerky and not safe

Chapter 5

Conclusion and Future Work

5.1 Discussion of Results

In simulation, our Lagrangian-PPO framework consistently produced behaviors that were safe and efficient in navigating to the goal. When using only the per-environment cost in policy and Lagrangian updates, agents learned to avoid the obstacles but required careful tuning of cost weight, cost limit and dual update learning rate. Conversely, when using only safety probability as the constraint, the dual update was more intuitive, as the safety is a single, easily interpretable global signal, but policies trained using this strategy struggled to converge and presented unsafe behavior. Ultimately, the hybrid formulation, using per-environment cost for policy update and safety probability for dual update, proved most efficient, yielding the fastest convergence and the most robust safe-RL performance. In particular, we observed that cost and safety probability are highly negatively correlated (Pearson ≈ -0.89), justifying the combination of cost-based gradients, which provide informative per-environment feedback, with a global safety probability signal to drive the dual update, exploiting the best of both worlds. The agents trained with the hybrid strategy consistently converged to safe and efficient policies, achieving more than 99% safety rates while quickly achieving the goals.
However, in the real-world JetBot trials, the learned policy failed to satisfy the safety constraint. The robot often reached the goal but clipped or traversed the maze walls. We traced this sim-to-real gap to four principal factors: (i) an action space mismatch, (ii) control loop latency, (iii) wheel slip and friction differences from the simulation and lastly (iv) noisy motion capture measurements. Together, these discrepancies led to jerky control and drifts that drove the robot into the obstacles. In short, our Lagrangian-PPO approach excels in IsaacLab's parallel simulator but exhibits degraded performance in the implementation in real world, that will remain until the action mapping and control latency mismatches are addressed.

5.2 Summary of Contributions

Our work makes the following contributions:

- **Lagrangian–PPO in simulation:** we adapted the PPO algorithm within the SKRL framework to alternate between policy optimization and dual multiplier updates, in order to obtain policies that satisfy either cost or safety probability constraints. Leveraging IsaacLab's GPU-parallelized vectorized environments (4096 concurrent instances), we demonstrated rapid convergence to safe policies in both simple arena and maze scenarios.
- **Hybrid constraint formulation:** we introduced a novel hybrid approach that updates the policy with per-environment obstacle cost gradients while updating the Lagrangian multiplier using global safety probability feedback. This hybrid method converges fast and is easier to tune than cost-only or probability-only variants, as evidenced by extensive ablations.
- **Empirical sim-to-real analysis:** we deployed the learned policies on a physical JetBot in the maze in our lab and characterized the sim-to-real gap. We identified actuator deadzones, action space clipping, control loop latency, wheel slip friction and sensor noise as the probable failure points, providing clear targets for future improvement.
- **Open-source implementation.** all code, IsaacLab environment definitions, SKRL algorithm wrappers, logging utilities, and ROS2 inference nodes, is publicly available on GitHub, providing a reproducible platform for safe-RL research in robot navigation.

5.3 Limitations

While our experiments validate the Lagrangian–PPO approach in controlled settings, several limitations remain:

- **Simulated and real environment are identical:** we trained in a simulator whose geometry and obstacle placements were designed to match our physical maze. In practice, a truly map-free robot must generalize to unseen obstacle layouts. Without training on randomly generated mazes, our policies risk overfitting to the specific wall configuration.
- **Action-space and dynamics mismatch:** IsaacLab's internal actuator model differs substantially from the JetBot's real motors, which exhibit nonlinear torque saturation, deadzones, and slippage. Consequently, even a perfectly safe simulator policy may not transfer safely if the action mapping and control frequency are not carefully aligned.
- **Single agent focus:** all results consider only one JetBot navigating alone; multiagent interactions, like avoiding other robots, were not studied. In collaborative or competitive scenarios, the Lagrangian safety formulation would require extension to account for dynamic obstacle interactions.
- **No end-to-end learning from raw perception:** we relied on handcrafted, fully observable features (positions, distances, angles). While this accelerates algorithmic development, it limits generalization to more complex

visual or LiDAR inputs, which are necessary for deployment in unstructured environments.

5.4 Future Directions

Overall the approach seems promising, based on the findings, and calls for multiple advancements:

- **Random maze generation and domain randomization:** to ensure real generalization for map-free navigation, we propose to train policies on large ensembles of randomly generated mazes, with varying wall thickness, exit locations, and obstacle density, while using randomized friction and sensor noise. This curriculum will produce more robust policies capable of zero-shot transfer to unseen real-world layouts.
- **Action mapping and dynamics adaptation:** developing a systematic calibration pipeline that learns the mapping from raw policy outputs to actual wheel commands on hardware.
- **Multiagent safe navigation:** extending the single agent Lagrangian-PPO framework to multiagent settings, introducing additional safety constraints to avoid inter-robot collisions.
- **End-to-end perception to control:** replacing the handcrafted observation vector with a learned encoder that processes raw LiDAR scans or monocular camera inputs. The robot could then plan safe trajectories directly from sensor raw data, enabling navigation in dynamic, unstructured environments.

Bibliography

[1] Hongyi Chen and Changliu Liu. Safe and sample-efficient reinforcement learning for clustered dynamic environments. *IEEE Control Systems Letters*, 6:1928–1933, 2022.

[2] Joonho Lee, Lukas Schroth, Victor Klemm, Marko Bjelonic, Alexander Reske, and Marco Hutter. Exploring constrained reinforcement learning algorithms for quadrupedal locomotion. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11132–11138, 2024.

[3] Tingting Ni and Maryam Kamgarpour. A learning-based approach to stochastic optimal control under reach-avoid constraint. In *Proceedings of the 28th ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '25, New York, NY, USA, 2025. Association for Computing Machinery.

[4] Sahar Salimpour, Jorge Peña-Queralta, Diego Paez-Granados, Jukka Heikkonen, and Tomi Westerlund. Sim-to-real transfer for mobile robots with reinforcement learning: from nvidia isaac sim to gazebo and real ros 2 robots, 2025.

[5] Chen Tessler, Daniel J. Mankowitz, and Shie Mannor. Reward constrained policy optimization, 2018.

[6] Tengyu Xu, Yingbin Liang, and Guanghui Lan. Crpo: A new approach for safe reinforcement learning with convergence guarantee, 2021.