APPENDIX

A. Evaluation with Reward

In Sec. VI-B, we evaluated each method in terms of success rate. In Table IV we present an additional analysis in terms of the sum of rewards in each episode. Note that, to compute the success rate for Table I, we terminate each episode when the box reaches the goal; on the other hand, in this experiment, we continuously sample a new goal until it reaches 1000 steps. As a result, the cost in Table IV is slightly different than the cost in Table I since the episode termination criteria is defined differently. In the Ant agent environment, we also terminate the episode when the agent flips over and cannot recover.

B. Additional Training Curves

The training curves of our method and the baseline methods in PointPush2 and CarPush2 are shown in Fig. 5. These two tasks are harder than PointPush1 and CarPush1 since they have more obstacles in their environment.

C. Safe Exploration Experiments

Safe exploration, which is considered another line of work in safe RL, also focuses on satisfying safety constraints for reinforcement learning agents. Compared to the safe RL methods mentioned in the main text, additional information such as an explicit dynamics models used to predict the future states given the action sequence,is usually assumed to be given. Still, we compare our method with the Safety Layer method [13]. In the Safety Layer method, we first randomly sampled actions for 1e6 time steps to train the cost function which is used to set up the safety layer. Then the safety layer is combined with SAC to train a safe RL agent. We compare it with our method in the PointPush1 environment. The results are shown in Fig. 6, with our method shown in blue and the baseline shown in orange. This result shows that Safety Layer cannot outperform our method in terms of either reward or cost. It is generally hard to train an accurate cost function in practice, leading to poor performance by this method.

D. Adjusting the Trade-off between Reward and Cost

We can adjust the trade-off between the reward and the cost during test time in the proposed method. Specifically, we can adjust the threshold ϵ' defined in Equation 2. The results are summarized in Fig. 7. As a comparison, we also include three baselines: CPO, PPO-Lagrangian, and TRPO-Lagrangian.

From Fig. 7, smaller ϵ' might lead to a less conservative agent with higher reward and higher cost. Nonetheless, our method has the higher reward and lower cost compared to the baselines across different values of ϵ'.

E. How much of our improvement over the baselines is attributed to using a learned trajectory-following module

As an additional experiment to understand the effects of a trajectory-following module, we modify two of the baselines to incorporate a trained goal-reaching low-level agent. It is trained in the same way as the goal-reaching agent for our method except that the subgoal is randomly sampled to match the inference distribution. Note that this goal-reaching agent does not incorporate any safey constraints. Thus for this baseline, we use PPO Lagrangian to train a high-level policy that outputs a (hopefully safe) subgoal. The results of this experiment can be found in Table V, referred to as "PPO Lag + SAC." The results for the Mass agent are left blank since the Mass agent does not require learning a low-level goal-reaching policy. As can be seen, "PPO Lag + SAC" performs poorly, incurring a low reward and many safety violations. Though "PPO Lag + SAC" performs slightly better than the PPO Lagrangian method, there is still a huge performance gap compared to our method. This demonstrates that the benefits of our method do not come directly from the trajectory-following module; incorporating such a module into the baselines still leads to poor performance.

F. How much is our performance affected by perceptual errors?

As noted, some of the errors in our system come from perceptual errors, in which the location of the obstacles is perturbed by some noise in the Safety Gym. To measure the effect of these errors, we perform an experiment in which we allow our trajectory optimizer to have access to the ground-truth location of the obstacles, while the RL agent still takes in the noisy LiDAR observations as input. The results can be found in Table V and it is denoted as "Ours w/ gt". As expected, the performances for all the environments have increased. The MassPush1 task has the greatest improvement in reward by switching to ground-truth locations. This is because perception errors are the major source oferrors for the Mass agent, for which the agent dynamics are relatively simple.

G. Experiments with Safety Gym Goal Tasks

We evaluate our method with an additional task, the "Goal" task in SafetyGym [9]. In the Goal task, the robot itself needs to go to a specific goal and avoid obstacles instead of moving an object to the goal. The results are shown in Fig. 8. Our method still achieves the lowest cost and a relatively high reward during training.

H. Analyzing the Ablations

In Section VI-C, we discussed an ablation called "SAC + PPO Lag" in which we train a low-level policy to safely reach goals with PPO Lagrangian, and then we use SAC to output subgoals (see Section VI-C for further discussion). In this section, we analyze why this method appears to perform so poorly in Table I.

The training curves for the low-level safe goal-reaching policy trained with PPO Lagrangian are shown in Fig. 9. As can be seen, these policies do not learn to be safe and the cost during training is always above 0. Hence, integrating such a low-level policy with a high-level SAC agent leads to poor overall performance for "SAC + PPO Lag" baseline. This experiment highlights the difficulties of training even a safe short-horizon policy with PPO Lagrangian.

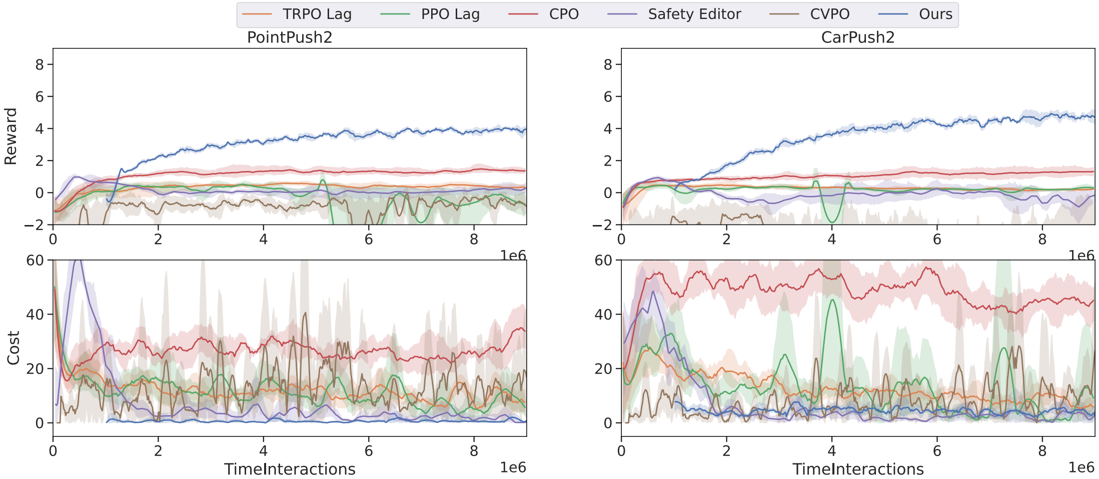


Fig. 5: Additional training curves of our method compared to the baseline methods. The shadow region denotes the standard error of different seeds. Our method starts from $1e6$ steps instead of 0 to denote the training of the goal-reaching policy. Our method achieves a lower cost than the baselines. It still incurs some cost during training because, during training time, we are using a fixed Lagrangian parameter for computation reasons and to encourage exploration.

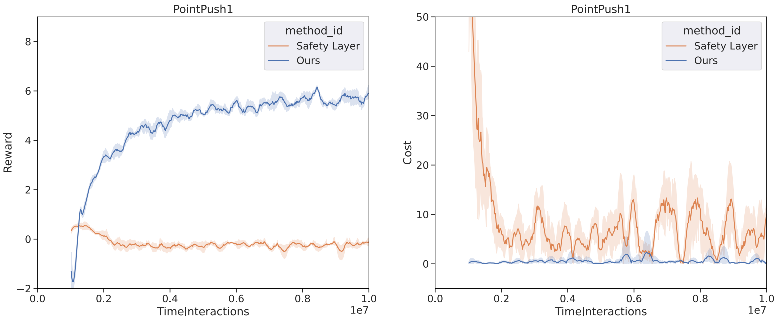


Fig. 6: Comparison between our method and the Safety Layer method. [13].

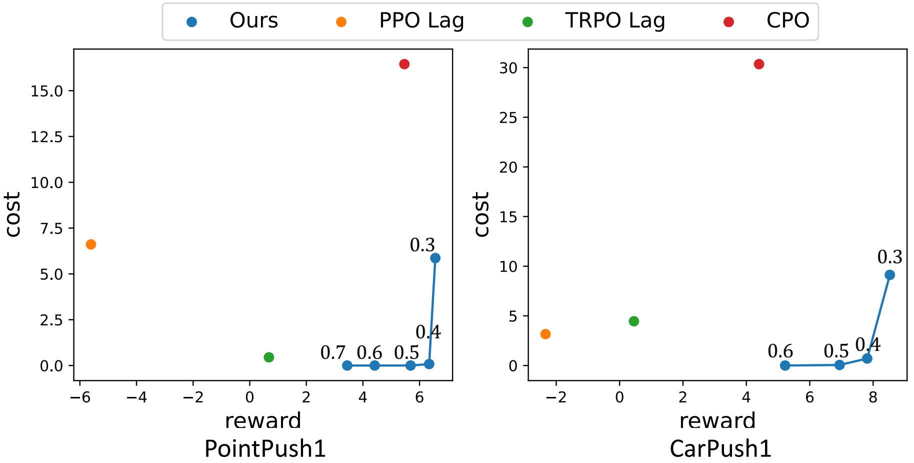


Fig. 7: Trade-off between the reward and the cost for different methods. The blue dots represent our method with different ϵ'. The numbers above the blue dots denote the value of ϵ'.

TABLE IV: Evaluation results of the final policy, using an adaptive Lagrangian parameter for our method. See text and Appendix. O for details.

		Ours	CPO [8]	PPO Lag [9]	TRPO Lag [9]	SE [41]	CVPO [11]
MassPush1	reward	**4.18**	0.78	-2.26	-0.39	0.36	-1.95
	cost	**0.00**	31.48	2.6	0.05	10.43	2.75
PointPush1	reward	**5.29**	5.47	-5.61	0.68	0.62	-0.12
	cost	**0.00**	16.45	6.61	0.45	5.16	**0.00**
CarPush1	reward	**6.47**	4.4	-2.34	0.45	0.58	5.34
	cost	**0.03**	30.35	3.17	1.46	0.92	22.00
AntPush1	reward	**6.55**	0.12	0.24	-0.02	0.01	0.05
	cost	0.48	0.01	**0.00**	**0.00**	0.85	**0.00**
PointPush2	reward	**3.47**	2.05	-7.62	0.18	0.22	0.85
	cost	0.08	32.63	11.95	4.20	**0.01**	11.25
CarPush2	reward	**4.22**	2.92	0.24	0.18	-0.55	-7.45
	cost	**0.29**	52.40	4.16	3.29	1.49	51.13

TABLE V: Evaluation results of our method and ablations. Each method was trained for $1e7$ environment interaction steps..

		Ours	PPO Lag + SAC	Ours w/ gt
MassPush1	reward	**4.31**		10.00
	cost	**0.00**		0.00
PointPush1	reward	**5.69**	0.16	5.70
	cost	**0.00**	1.41	0.00
CarPush1	reward	**4.57**	0.15	5.33
	cost	**0.00**	5.58	0.00

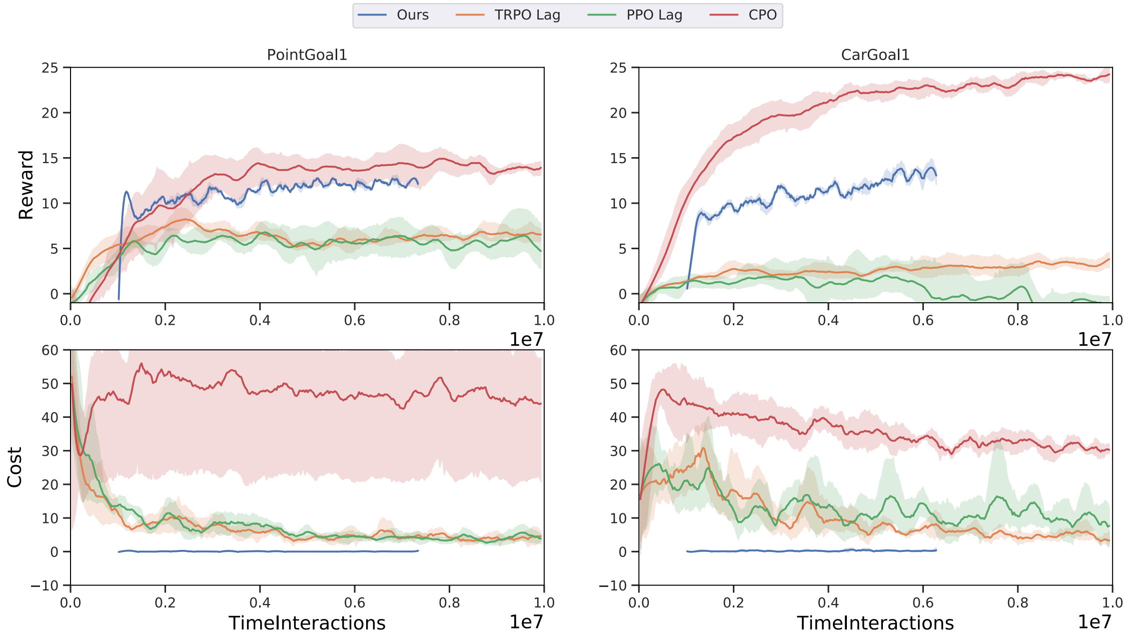


Fig. 8: Training curves of our methods compared to the baseline methods in the Goal environment. The curves have been smoothed for better visualization. Our method starts from $1e6$ steps instead of 0 to denote the training of the goal-reaching policy. Our method achieves the lowest cost among all the baseline methods.

I. Training Curves of the Ablations

The training curve of ablation methods are shown in Fig. 10 which corresponds to the results in Table I. The ablations have much higher cost than our method during training and significantly higher cost than our method during inference (in which our method obtains 0 cost) as shown in Table I.

J. Pseudocode

The pseudocode for our method can be found in Algorithm 1.

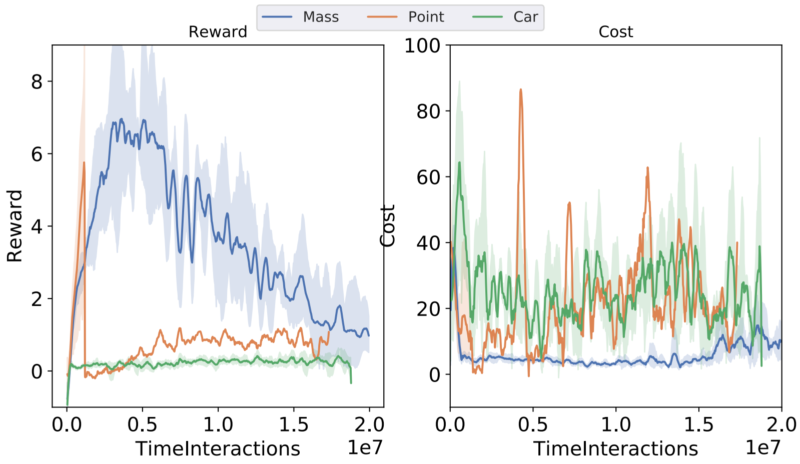


Fig. 9: Training curves of the goal-reaching policies used in "SAC + PPO Lag".

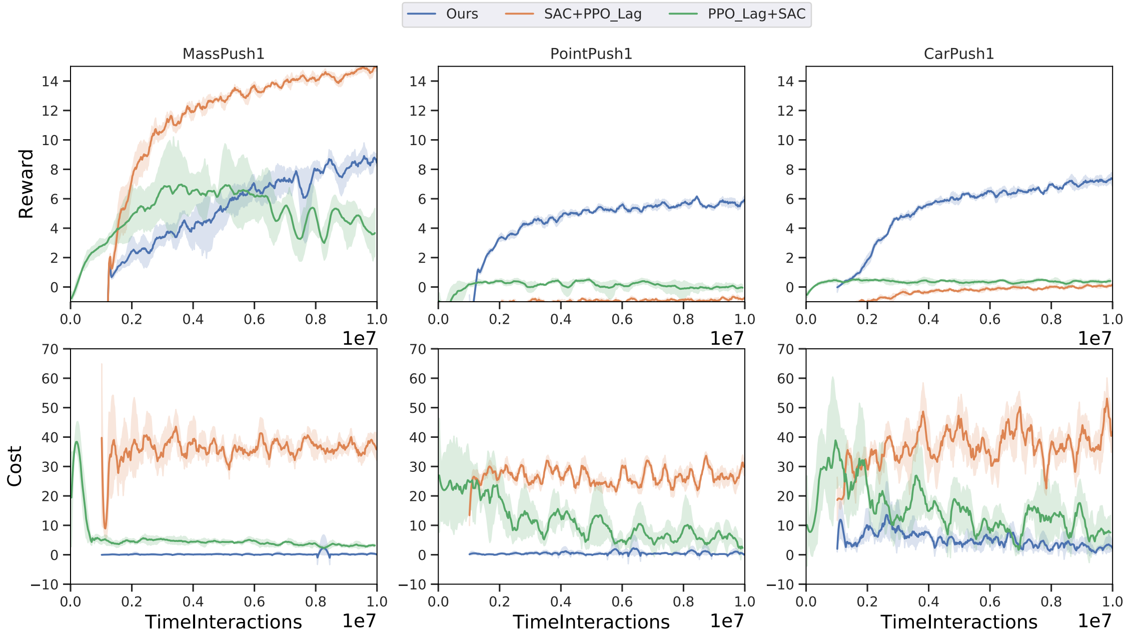


Fig. 10: Training curves of our method and 2 ablations (see Section VI-C for details on these methods).

K. *Definition of "Root" Node*

In MuJoCo, a robot is defined in a tree structure. Specifically, a robot is usually defined by mounting the adjacent link onto the previous link. The root node of the robot is the root body of the tree structure. In our experiments, it is usually the body of the robot. Thus the location of the root node can be interpreted as the center of the robot. The position of the root node is shown in Fig. 11. The small blue sphere inside the robot denotes the position of the root node.

L. *Implementation Details about the RL policy*

We use Soft Actor-Critic (SAC) [36] to train the high-level policy. The actor network and the critic network are three-layer neural networks with a hidden size of 256. The learning rate for training is set to be $3\text{e}{-4}$. The agent interacts with the environment for $1\text{e}7$ time steps to ensure it has fully converged.

M. *Additional Details about the Trajectory-following Module*

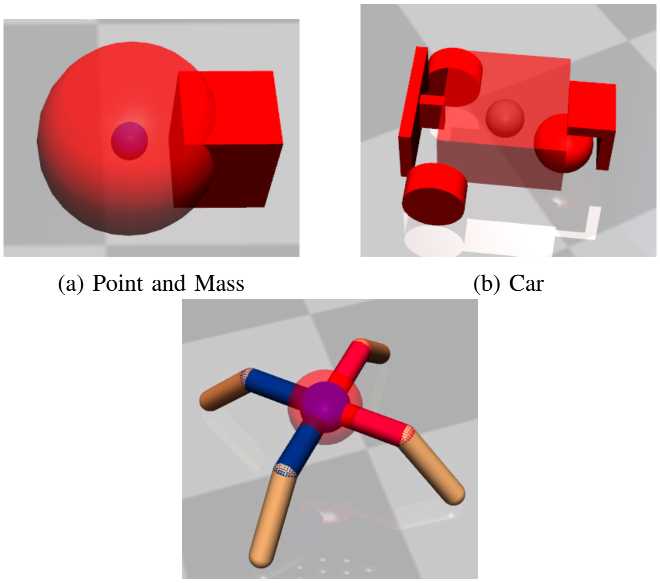
Training of the goal-following agent: The goal $\mathbf{x}_i$ is sampled uniformly from a range of d_{min} to $2d_{min}$ away from the robot to match the input distribution during inference, in which d_{min} is a distance threshold. The reward function used for training the goal-following agent is defined as the change in distance between the robot and the goal:

$$r_t^g := ||\mathbf{s}_{t-1,x} - \mathbf{g}|| - ||\mathbf{s}_{t,x} - \mathbf{g}||. \tag{6}$$

SAC is used to train the goal-following agent. The actor network and the critic network are three-layer neural networks

Algorithm 1 Reinforcement Learning in a Safety-Embedded MDP with Trajectory Optimization

```
Require: RL policy output interval $k$, goal-following agent $\pi_\phi$.
  Initialize the replay buffer $D$, high-level RL policy $\pi_\theta$.
  for each episode do
      for each environment step do
          Select subgoal: $\mathbf{a}'_t \sim \pi_\theta(\mathbf{s}_t)$.
          Input the subgoal $\mathbf{a}'_t$ to the trajectory optimizer to obtain a safe trajectory: $\mathbf{X} \leftarrow \text{TrajOpt}(\mathbf{s}_t, \mathbf{a}'_t)$.
          Initialize cumulative reward $r_t \leftarrow 0$.
          Initialize the initial state $\mathbf{s}_{t,i} \leftarrow \mathbf{s}_t$.
          for $i = 1$ to $k$ do
              Follow the trajectory:
              $\mathbf{a}_{t,i} \leftarrow \text{TrajFollow}(\mathbf{s}_{t,i}, \mathbf{X}, \pi_\phi)$.
              Execute $\mathbf{a}_{t,i}$ and observe reward $r_{t,i}$ and $\mathbf{s}_{t,i}$.
              Sum the reward $r_t \leftarrow r_t + r_{t,i}$.
          end for
          Save the final state: $\mathbf{s}_{t+1} \leftarrow \mathbf{s}_{t,k}$.
          Store the transition $(\mathbf{s}_t, \mathbf{a}'_t, \mathbf{s}_{t+1}, r_t)$ into $D$.
          Update the policy $\pi_\theta$ on data from $D$.
      end for
  end for
```



(a) Point and Mass

(b) Car

(c) Ant

Fig. 11: Root node position of different robots shown by the location of the small blue sphere.

with a hidden size of 256. d_{min} is set to be $0.2m$. The agent interacts with the environment for $1e6$ time steps to ensure it has fully converged.

Choosing a waypoint to follow: In the trajectory-following module, the trajectory is represented as a set of waypoints. However, the goal-following agent only takes in a single goal as input. Thus, we use the following procedure to select a waypoint from the set for the goal-following agent:

1) The agent keeps track of the waypoint that is input into the goal-following agent ("tracking point"). The tracking point is initialized as the first waypoint of the trajectory.
2) At each time step, the agent searches from the tracking point to the end of the trajectory until it finds a waypoint that is at least d_{min} away from the current robot root position. It sets the point as the tracking point for the current time step. It is possible that the tracking point is the same as the previous time step.

N. Additional Details about the Trajectory Optimizer

Hyperparameters: The distance threshold ϵ' is set to be $0.5m$ during training for all of the robots. The number of waypoints is set to be 30. The maximum time step for optimizing the trajectory is set to be 10 steps during training. If the robot is currently within ϵ' of some obstacle, we would manually set $\boldsymbol{\lambda}$ to be the maximum value $\boldsymbol{\lambda}_{max}$ to encourage the robot to escape the unsafe region as quickly as possible.

Accelerate the training: In practice, since the Lagrangian method adds computational overhead, we use a fixed $\boldsymbol{\lambda}$ during training, which speeds up training time and also encourages exploration (although this leads to some safety violations during training). At test time, we update $\boldsymbol{\lambda}$ as mentioned previously, which ensures safety at test time.

Implementation simplifications: In our experiments, instead of using three distinctive Lagrangian values for the three constraints mentioned in equation 3 respectively, we use a single Lagrangian value λ for all 3 constraints, to simplify our implementation.

To solve the dual problem mentioned in Equation 5, instead of using gradient descent to update λ, in practice, we follow a simpler procedure of increasing λ if the obstacle constraints are violated until the optimizer returns a feasible solution.

O. Training Details in the Evaluation Experiment

For each experiment, we train four different seeds to account for randomness.

For our method, we train it for $9e6$ steps in all the environments. For CPO, PPO Lagrangian, TRPO Lagrangian, we train them for $6e7$ in MassPush1, PointPush1, CarPush1, PointPush2, and CarPush2, and we train them for $1e7$ in AntPush1. For Safety Editor and CVPO, we train them for $1e7$ for all the environments. The baseline methods have more time steps to ensure convergence.

P. Implementation Details of Training the Goal-reaching Policy

For our method and the "PPO Lag + SAC" ablation, we use SAC to train the goal-reaching policy. In our method, we sample the goal from a range of d_{min} to $2d_{min}$. Specifically, d_{min} is set to be $0.2m$ in our experiment to match the average distance between waypoints in the trajectory. For the "PPO Lag + SAC" ablation, we sample the goal randomly from a $2m \times 2m$ area to match the distribution of subgoals output from the high-level PPO Lagrangian agent.

Q. Real-robot Experiment Details

In the real-robot experiment, the fingertip of the gripper moves in a plane to push the box towards the goal (the large green circle). As in the simulation setup. the robot also needs to avoid hazards (the small red circle) and try not to get stuck by the pillar (blue).

In order to match the observation distribution seen during training, we convert the location of the objects into LiDAR readings in a similar format to the Safety Gym Pseudo LiDAR.

In this experiment, we first train the policy in simulation. We modify the simulation environment to match the setup in the real world, e.g., we change the size of objects and the shape of the robot in the simulation. Then we directly transfer the policy to the real robot without any finetuning.

R. Safety Gym Objects

The following types of objects exist in the environments for the Push tasks:

1) **Goal** indicates where the robot needs to reach. As long as the robot is within a specific range of the goal, the task is considered successful.
2) **Hazard** denotes circular regions that the robot should not enter. It is a virtual region and does not interact with the robot. The robot entering such a region will incur the cost. It is defined as obstacles in our method.
3) **Pillar** is a fixed cylinder; contact with the pillar will not incur any cost. Though contacting with pillar might now incur cost, the robot might get stuck. Thus, it is also defined as an obstacle in our method.
4) **Box** is an object that the robot can (and must) interact with: the goal is for the agent to push the box towards the goal. If the box enters the hazard regions, it will not incur any cost.

S. Safety Gym agent

There are four different kinds of agents in our Safety Gym experiments: Point, Car, Mass, and Ant. The Point and Car are two default agents from the Safety Gym benchmark; the Point agent is a robot with one actuator for turning and another actuator for moving forward or backward. The Car agent is a robot with two independently driven wheels. We created the Mass agent, which is an omnidirectional agent; the action space of the Mass agent is defined as a delta movement of the agent position. The Ant agent is a quadrupedal agent with eight joints, similar to the Ant agent from MuJoCo [42].

T. Reward Function

The reward function for the "Push" task is defined as the change in the distance between the robot and the box and between the box and the goal:

$$r_t^p := ||\mathbf{x}_{t-1}^b - \mathbf{g}|| - ||\mathbf{x}_t^b - \mathbf{g}|| + ||\mathbf{s}_{t-1,x} - \mathbf{x}_{t-1}^b|| - ||\mathbf{s}_{t,x} - \mathbf{x}_t^b||, \tag{7}$$

in which $\mathbf{x}_t^b$ denotes the location of the center of the box at the t th step, and $\mathbf{g}$ denotes the location of the final goal that the box needs to reach.

U. Cost Function

The cost function is defined as:

$$c_t^p := \mathbb{1}(\min_j ||\mathbf{s}_{t,x} - \mathbf{x}_j^{obs}|| < \epsilon), \tag{8}$$

in which $\mathbb{1}$ is the indicator function, and ϵ is a distance threshold. The cost function will return 1 if the "root" of the robot is within ϵ of any obstacle.

V. Observation Space

The observation space of the Safety Gym environment consists of LiDAR observation and robot states. The LiDAR observation returns an approximate position of the objects, due to the limited LiDAR resolution. The robot states consist of robot velocity, acceleration, and orientation.

W. Reasons of Safety Violations

Though our trajectory optimizer is formulated to completely avoid obstacles, in practice, the executed trajectory might not be perfectly safe, due to perception errors or errors with the trajectory-following module. Specifically, there might be errors in practice due to a number of factors:

1) Localization errors in estimating the robot position $\mathbf{s}_{t,x}$
2) Perception errors in estimating the locations of the obstacles $\mathbf{x}_j^{obs}$
3) Errors in following the planned trajectory
4) Discretization errors: since the planned trajectory is a discrete set of waypoints $\mathbf{X} := \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$, it is possible that the agent will encounter an unsafe state in between the trajectory waypoints that was not accounted for in the trajectory optimization.
5) It is possible that no safe trajectory exists from the robot's current state.

Our code is available at https://github.com/safetyembedded/SafetyEmbeddedMDP.