Runtime-Safety-Guided Policy Repair

Weichao Zhou[1], Ruihan Gao[2], BaekGyu Kim[3],
Eunsuk Kang[4], and Wenchao Li[1]

[1] Boston University, Boston MA, USA
{zwc662,wenchao}@bu.edu
[2] Nanyang Technological University, Singapore
GAOR0007@e.ntu.edu.sg
[3] Toyota Motor North America R&D, Mountain View, CA, USA
baekgyu.kim@toyota.com
[4] Carnegie Mellon University, Pittsburgh, PA, USA
eskang@cmu.edu

Abstract. We study the problem of policy repair for learning-based control policies in safety-critical settings. We consider an architecture where a high-performance learning-based control policy (e.g. one trained as a neural network) is paired with a model-based safety controller. The safety controller is endowed with the abilities to predict whether the trained policy will lead the system to an unsafe state, and take over control when necessary. While this architecture can provide added safety assurances, intermittent and frequent switching between the trained policy and the safety controller can result in undesirable behaviors and reduced performance. We propose to reduce or even eliminate control switching by 'repairing' the trained policy based on runtime data produced by the safety controller in a way that deviates minimally from the original policy. The key idea behind our approach is the formulation of a trajectory optimization problem that allows the joint reasoning of policy update and safety constraints. Experimental results demonstrate that our approach is effective even when the system model in the safety controller is unknown and only approximated.

1 Introduction

Data-driven methods such as imitation learning have been successful in learning control policies for complex control tasks [4]. A major shortcoming that impedes their widespread usage in the field is that the learnt policies typically do not come with any safety guarantee. It has been observed that when encountering states not seen in training, the learnt policy can produce unsafe behaviors [3,27].

A common approach to mitigate the safety problem at runtime is to pair the learning-based controller[5] (LC) with a high-assurance safety controller (SC) that can take over control in safety-critical situations, such as the Simplex architecture first proposed in [32]. The safety controller is tasked with predicting

[5] We use the terms 'controller' and 'control policy' (or simply 'policy') interchangeably in this paper. The latter is more common in the machine learning literature.

an impending safety violation and taking over control when it deems necessary. Such controllers are often designed based on conservative models, has inferior performance compared to its learning-based counterpart, and may require significant computation resources if implemented online (e.g. model predictive control). Moreover, frequent and intermittent switching between the controllers can result in undesirable behaviors and further performance loss.

In this paper, we propose to *leverage the runtime interventions carried out by the safety controller to repair the learnt policy*. We do not assume access to the original training data of the LC but we assume that the policy is parameterized, differentiable and given as a white-box. This means that while fine-tuning the LC from scratch is not possible, it is still possible to improve the controller based on new data that is gathered during deployment. In particular, we introduce the concept of *policy repair* which uses the outputs of the safety controller to synthesize new training data to fine-tune the LC for improved safety. Furthermore, we formalize a notion of *minimal deviation* with respect to the original policy in order to mitigate the issue of performance degradation during policy repair. The main idea in *minimally deviating policy repair* is the formulation of a trajectory optimization problem that allows us to *simultaneously reason about policy optimization and safety constraints*. A key novelty of this approach is the synthesis of new safe 'demonstrations' that are the most likely to be produced by the original unsafe learnt policy. In short, we make the following contributions.

- We formalize the problems of *policy repair* and *minimally deviating policy repair* for improving the safety of learnt control policies.
- We develop a novel algorithm to solve the policy repair problem by iteratively synthesizing new training data from interventions by the safety controller to fine-tune the learnt policy.
- We demonstrate the effectiveness of our approach on case studies including a simulated driving scenario where the true dynamics of the system is unknown and is only approximated.

2 Related Work

Model-based control is a well-studied technique for controlling dynamical systems based on the modelling of the system dynamics. Algorithms such as iterative Linear Quadratic Regulator (iLQR) [33] have achieved good performance even in complex robotic control tasks. One important advantage of model-based control is its ability to cope with constraints on the dynamics, controls and states. Constrained Model Predictive Control [15] has been studied extensively and proven to be successful in solving collision avoidance problems [5,6] as well as meeting complex high-level specifications [10]. In this paper, we utilize model-based control techniques to verify the existence of safe control as well as synthesize new training data to guide the policy learning.

Imitation learning provides a way of transferring skills for a complex task from a (human) expert to a learning agent [20]. It has been shown that data-driven methods such as behavior cloning are effective in handling robotics and

autonomous driving tasks [25,28] when an expert policy is accessible at training time. Model-based control techniques have already been introduced to imitation learning to guide the policy learning process [22]. Our work shares similarity with [22] in using a model predictive controller to generate training examples. What distinguishes our work from theirs is that in [22] the model predictive controller operates based on a given cost function whereas in our work we do not assume we know any cost function. An outstanding challenge in the imitation learning area is the lack of safety assurance during both training and final deployment. Efforts on addressing this challenge include [36,17], where multiple machine learning models cooperate to achieve performance and safety goals. However, the learned models can not provide guarantees on runtime safety by themselves. In fact, even when the dynamical model is given, existing imitation learning algorithms lack the means to incorporate explicit safety requirements. In this paper, we use imitation learning to formulate the problem of minimally deviating policy repair such that a repaired policy can match the performance of the original learnt policy while being safe.

Safe Learning research has experienced rapid growth in recent years. Many approaches consider safety requirement as constraints in the learning process. For example, [1,9] encodes safety as auxiliary costs under the framework of Constrained Markov Decision Processes (CMDPs). However, the constraints can only be enforced approximately. [9] developed a Lyapunov-based approach to learn safe control policies in CMDPs but is not applicable to parameterized policy and continuous control actions. Formal methods have also been applied to certain learning algorithms for establishing formal safety guarantees. In [37], safety is explicitly defined in probabilistic computational tree logic and a probabilistic model checker is used to check whether any intermediately learned policy meets the specification. If the specification is violated, then a counterexample in the form of a set of traces is used to guide the learning process. Providing assurance for runtime safety of learning-based controller has also garnered attention recently. [12] combines offline verification of system models with runtime validation of system executions. In [2], a so-called shield is synthesized to filter out unsafe outputs from a reinforcement learning (RL) agent. It also promotes safe actions by modifying the rewards. A similar idea can be seen in [23] where a so-called neural simplex architecture is proposed and an online training scheme is used to improve the safety of RL agents by rewarding safe actions. However, in the context of RL, choosing the right reward is in general a difficult task, since incorrect choices often lead to sub-optimal or even incorrect solutions. In [8], a model predictive approach is proposed to solve for minimum perturbation to bend the outputs of an RL policy towards asymptotic safety enforced by a predefined control barrier certificate. A similar idea also appears in [34] where robust model predictive control is used to minimally perturb the trajectories of a learning-based controller towards an iteratively expanding safe target set. Our method differs from [8,34] as we improve the runtime safety of the learning-based control while preserving its performance from an imitation learning perspective.

3 Preliminaries

In this paper we consider a discrete-time control system (X, U, f, d_0) where X is the set of states of the system and U is the set of control actions. The function $f : X \times U \to X$ is the dynamical model describing how the state evolves when an control action is applied, and $d_0 : X \to \mathbb{R}$ is the distribution of the initial states. By applying control actions sequentially, a trajectory, or a trace, $\tau = \{(x_t, u_t)|t = 0, 1, \ldots\}$ can be obtained where x_t, u_t are the state and control action at time t. In typical optimal control problems, a cost function $c : X \times U \to \mathbb{R}$ is explicitly defined to specify the cost of performing control action $u \in U$ in state $x \in X$. The cumulative cost along a trajectory τ can be calculated as $\sum_{(x_t,u_t)\in\tau} c(x_t, u_t)$. An optimal control strategy is thus one that minimizes the cumulative cost.

Model Predictive Control (MPC) leverages a predictive model of the system to find a sequence of optimal control actions in a receding horizon fashion. It solves the optimal sequence of control actions for T steps as in (1) but only applies the first control action and propagates one step forward to the next state. Then it solves for a new sequence of optimal control actions in the next state.

$$\arg\min_{x_{0:T}, u_{0:T}} \sum_{t=0}^{T} c(x_t, u_t) \tag{1}$$

$$s.t.\ \ x_{t+1} = f(x_t, u_t) \qquad t = 0, 1, 2, \ldots, T \tag{2}$$

When the dynamics f in constraint (2) is nonlinear, the iterative Linear Quadratic Regulator (iLQR) algorithm [14] applies a local linearization of f along an existing trajectory which is called the nominal trajectory. It computes a feedback control law via LQR [13], which induces a locally optimal perturbation upon the nominal trajectory to reduce the cumulative cost. Formally, given a nominal trajectory $\{(x_0, u_0), ..., (x_T, u_T)\}$, perturbations can be added to each state and control action in this trajectory, i.e. $x_t \to x_t + \delta x_t, u_t \to u_t + \delta u_t$. The relationship between $\delta x_t, \delta u_t$ and δx_{t+1} is locally determined by the dynamics as well as the state and control actions in the nominal trajectory as in (4) where $\nabla_x f(x_t, u_t), \nabla_u f(x_t, u_t)$ are the partial derivatives of $f(x_t, u_t)$ w.r.t x, u. Meanwhile, based on the nominal trajectory, $\sum_{t=0}^{T} c(x_t, u_t)$ in the objective (1) is substituted by $\sum_{t=0}^{T} c(\delta x_t + x_t, \delta u_t + u_t) - c(x_t, u_t)$ while the decision variables become $\delta x_{0:T}, \delta u_{0:T}$. When adopting an online trajectory optimization strategy [33], the optimal control law has a closed form solution $\delta u_t = k_t + K_t \delta x_t$ in which k_t, K_t are determined by the dynamics and the cumulative cost along the nominal trajectory.

$$x_{t+1} = f(x_t, u_t) \qquad x_{t+1} + \delta x_{t+1} = f(x_t + \delta x_t, u_t + \delta u_t) \tag{3}$$

$$\delta x_{t+1}^T \approx \delta x_t^T \nabla_x f(x_t, u_t) + \delta u_t^T \nabla_u f(x_t, u_t) \tag{4}$$

A **control policy** in general is a function $\pi : X \to U$ that specifies the behavior of a controller in each state. Given a deterministic policy π, its trajectory can be obtained by sequentially applying control actions according to the

outputs of π. Specifically, for an LC such as a deep neural network, the policy is usually parameterized and can be written as π_θ where the parameter θ belongs to some parameter set Θ (e.g. weights of a neural network). We assume that $\pi_\theta(x)$ is differentiable both in x and θ.

Imitation learning assumes that an expert policy π_E (e.g. a human expert) can demonstrate on how to finish a desired task with high performance. The learning objective for an agent is to find a policy π that matches the performance of π_E in the same task. Traditional approaches such as behavioral cloning consider the 0-1 error $e(x_t, \pi_E; \pi) = \mathcal{I}\{\pi(x) \neq \pi_E(x)\}$ where $\mathcal{I}$ is an indicator function. In this setting, an optimally imitating policy minimizes $\mathbb{E}_{x\sim d_{\pi_E}}[e(x, \pi_E; \pi)]$ where d_{π_E} is state visitation distribution of π_E. From another perspective, the difference between π and π_E can be estimated based on their trajectory distributions. When the trajectory distribution $Prob(\tau|\pi_E)$ is known, one can empirically estimate and minimize the KL divergence $D_{KL}[\pi_E||\pi]$ by regarding $Prob(\tau|\pi)$ as the probability of π generating trajectory τ under an additional Gaussian noise, i.e. $u_t \sim \mathcal{N}(\pi(x_t), \Sigma), \forall(x_t, u_t) \in \tau$. On the other hand, one can estimate and minimize the KL divergence $D_{KL}[\pi||\pi_E]$ by treating $Prob(\tau|\pi)$ as being induced from a Dirac delta distribution $u_t \sim \delta(\pi(x_t))\ \forall(x_t, u_t) \in \tau$. Both KL-divergences are related to negative log-likelihoods.

4 Runtime Safety Assurance

In this section we discuss the runtime safety issues of LCs and introduce our basic strategy for safe control. We consider a runtime safety requirement Φ for finite horizon T, such as 'if the current state is safe at step t, do not reach any unsafe state within the next T steps'. Temporal logic can be used to formally capture this type of safety requirements [16,24]. Given an LC with a deterministic policy π_θ , if π_θ satisfies Φ globally, that is, at each time step along all its trajectories, we denote it as $\pi_\theta \models \Phi$; otherwise $\pi_\theta \not\models \Phi$.

We assume that for any satisfiable Φ, there exists an SC, which we represent as π^{safe}, that checks at runtime whether Φ is satisfiable if the output $\hat{u} = \pi_\theta(x)$ of the LC is directly applied. That is, whether there exists a sequence of control actions in the next $T-1$ steps such that Φ is not violated. If true, then the final output $\pi^{safe}(x, \pi_\theta(x)) = \hat{u}$. Otherwise it overrides the LC's output with $\pi^{safe}(x, \pi_\theta(x)) \neq \hat{u}$. We formally define the SC below.

Definition 1. *Given a safety requirement Φ, the corresponding SC is a mapping π^{safe} from $X \times U$ to U. In each state $x \in X$, $\pi^{safe}(x, \pi_\theta(x)) = \pi_\theta(x)$ iff Φ is satisfiable after applying the control action $\pi_\theta(x)$; otherwise, π^{safe} intervenes by providing a substitute $\pi^{safe}(x, \pi_\theta(x)) \neq \pi_\theta(x)$ to satisfy Φ.*

We use $\langle \pi_\theta, \pi^{safe} \rangle$ to represent the LC and SC pair. Obviously the trajectories generated by this pair satisfy Φ everywhere if π^{safe} exists. There are multiple options of implementing the SC such as having a backup human safety driver or using automated reasoning. Depending on the safety requirement and task environment, the difficulty of implementing safe control varies. In this paper, we

assume that a dynamical model of f is given, possibly constructed conservatively, and adopt a scheme known as Model Predictive Safe Control as detailed below.

4.1 Model Predictive Safe Control

This scheme exploits the dynamical model to predict safety in the future. Depending on the safety requirement Φ considered, a function $\varphi : X \to \mathbb{R}$ can be defined to quantify how safe any state x is, i.e. if $\varphi(x) \leq 0$, then x is safe; otherwise x is unsafe. Without loss of generality, we let the current step be $t = 0$. Then the safety requirement can be translated into the constraints $\forall t \in \{1, 2, \ldots, T\}, \varphi(x_t) \leq 0$. After the LC provides a candidate control output $u_0 = \pi_\theta(x_0)$, the SC first verifies the satisfiability of (7) by using an MPC-like formulation as (5) $\sim$ (8).

$$\min_{x_{0:T}, u_{0:T}} \quad 0 \tag{5}$$

$$s.t. \quad x_{t+1} = f(x_t, u_t) \qquad t = 0, 1, 2, \ldots, T-1 \tag{6}$$

$$\varphi(x_t) \leq 0 \qquad t = 1, 2, \ldots, T \tag{7}$$

$$u_0 = \pi_\theta(x_0) \tag{8}$$

The formula differs from MPC in that it solves a feasibility problem to check the existence of a sequence of control actions satisfying the constraints. It is easier to solve than optimal control since optimality is not required here. If this problem is feasible, that is, (6) $\sim$ (8) can be satisfied at the same time. Then $\pi_\theta(x_0)$ is deemed safe and the final output is $\pi^{safe}(x_0, \pi_\theta(x_0)) = \pi_\theta(x_0)$. Otherwise, the SC solves another feasibility problem which is the same as (5) $\sim$ (7) and has (8) removed because the unsafe candidate control action $\pi_\theta(x_0)$ is to be substituted. Note that it is possible that (7) is unsatisfiable, in which case there is no feasible solution. This means a safety violation is inevitable based on the given model, but the SC can predict such outcome T steps in advance and more drastic actions (e.g. physically changing the model) may be applied to prevent an accident from occurring. If a feasible solution to (5) $\sim$ (7) can be obtained, we let $\pi^{safe}(x_0, \pi_\theta(x_0)) = u_0$ and use this solved u_0 to evolve the system to the next state.

There have been works on model predictive control of cyber-physical systems subject to formal specifications in signal temporal logic (STL) and its probabilistic variant [26,29]. Techniques have been proposed to synthesize safety constraints from formal specifications to accommodate optimal control of continuous systems and to reason about safety under uncertainty. In the semantics of STL, φ can be viewed as the negation of the robustness satisfaction value.

In this paper, at the beginning of each time step, before solving the feasibility problem (5) $\sim$ (8), we forward simulate the policy π_θ for T steps. If the simulated trajectory satisfies the safety constraint (7) already, then there is no need to query the SC at all. Otherwise, we use the constrained iLQR approach from [7] to solve the feasibility problem. This approach treats the simulated trajectory as nominal trajectory and iteratively update the nominal trajectory. Also, this

approach turns the safety constraint (7) into a penalty $\sum_{t=0}^{T} exp(M_t\psi(x_t))$ with sufficiently large $\{M_t\}_{t=0}^{T}$. And the penalty is added to the objective. By using this approach, even if the feasibility problem cannot be solved, at least a low-penalty solution can be provided.

Monitoring overhead. Model Predictive Safe Control (MPSC) can provide assurance for a variety of runtime safety requirements. However, it can be more expensive to implement in practice compared to an LC due to the need to repeatedly solve a (nonlinear) optimization online as opposed to performing inference on a neural network [35]. Frequently using an SC to both verify safety and solve safe control at runtime can be computationally taxing for the entire control system. For instance, suppose the LC's inference time is t_{LC}, the time for solving (5) $\sim$ (8) is $t_{SC}^{(1)}$ and the time for solving (5) $\sim$ (7) is $t_{SC}^{(2)}$. Typically, t_{LC} is much smaller than $t_{SC}^{(1)}$ or $t_{SC}^{(2)}$. At each step, forward simulation of the LC for T steps takes at least $T * t_{LC}$ time. If (7) is violated in the forward simulation, the SC would need to be invoked and the total overhead will grow to $T * t_{LC} + t_{SC}^{(1)}$. If the problem based on the LC's candidate control output is infeasible and the SC is required to intervene with a substitute control value, then the SC will have to solve another MPC-like problem and the overhead will grow to $T*t_{LC}+t_{SC}^{(1)}+t_{SC}^{(2)}$. Thus, it would be more economical to have an inherently safe LC such that the SC is less triggered. Motivated by this, we propose to *repair* the LC so that it becomes safer and requires less intervention from the SC. In the next section, we formally introduce the policy repair problem and describe our solution in detail.

5 Policy Repair

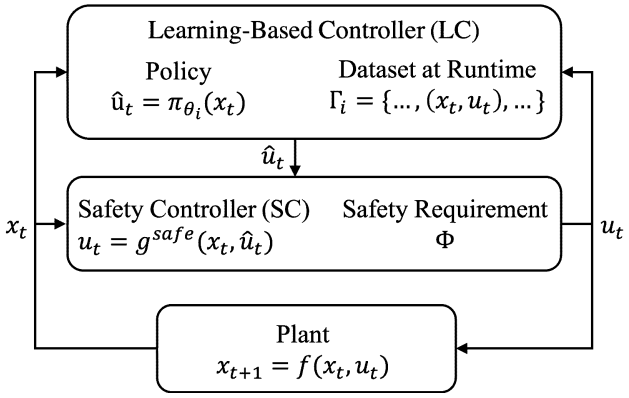

Fig. 1. Architecture of pairing LC's policy π_θ with an SC π^{safe}.

We first give a formal definition of the policy repair problem below.

Definition 2. *Given a deterministic policy π_θ paired with an SC π^{safe} as defined in Definition 1,* ***policy repair*** *is the problem of finding a new policy π_{θ^*} such that $\theta^* = \arg\min_{\theta\in\Theta} \mathbb{E}_{x\in X}[\mathcal{I}\{\pi^{safe}(x, \pi_\theta(x)) = \pi_\theta(x)\}]$ where $\mathcal{I}\{\cdot\} \in \{0, 1\}$ is an indicator function.*

Definition 2 implies that a repaired policy generates safe controls most of the time and thus the SC rarely intervenes. The first idea is to treat controls generated by the SC as repairs at specific states, and then use this data to repair the whole policy. A solution based on this idea is described as follows.

5.1 Naive Policy Repair

During the execution of the LC and SC pair $\langle \pi_\theta, \pi^{safe}\rangle$, due to the presence of the SC, all the generated traces are safe. The basic idea of the naive policy repair approach is to let the unsafe LC learn from the interventions produced by the SC. Specifically, we iteratively execute the LC and SC pair to generate new safe traces. After each iteration, the state-action pairs in all the previously generated traces are used as training data to update the policy of the LC. We present the steps in Algorithm 1 and illustrate them with a high-level diagram in Fig. 1, where Γ_i is the set of traces of the $\langle \pi_{\theta_i}, \pi^{safe}\rangle$ pair at the i^{th} iteration. We use supervised learning to fine-tune the policy parameter to minimize the expected error $\mathbb{E}_{(x,u)\sim\cup\Gamma_i}[e(x, u; \pi_\theta)]$ as in line 9 of Algorithm 1. Note that at this stage, with a slight abuse of notation, we view Γ_i as a data set containing (x, u) pairs. In line 5 $\sim$ 7, if the SC no longer intervenes, then we have a high confidence that the current policy is safe. According to the law of large numbers, this confidence increases with increasing number of sampled traces. The algorithm also terminates if a maximum iteration number is reached, in which case the SC may still intervene and the policy repair is only partially successful.

Algorithm 1 Naive_Policy_Repair

1: **Input** an initial policy π_{θ_0};
2: **Given** an SC π^{safe}; iteration parameter $N > 0$; policy parameter set Θ.
3: **for** iteration $i = 0$ to N **do**
4: Run the $\langle \pi_{\theta_i}, \pi^{safe}\rangle$ pair to generate a set Γ_i of trajectories.
5: **if** $\forall (x, u) \in \Gamma_i, u = \pi_{\theta_i}(x)$ **then**
6: π^{safe} never intervenes $\Rightarrow \pi_{\theta_i} \models \Phi$ with high probability.
7: **return** π_{θ_i}, Γ_i
8: **end if**
9: $\theta_{i+1} = \arg\min_{\theta\in\Theta} \mathbb{E}_{(x,u)\sim\cup_{j=0}^{i}\Gamma_j}[e(x, u; \pi_\theta)]$
10: **end for**
11: **return** $\pi_{\theta_N}, \emptyset$

5.2 Analysis of Performance Degradation due to SC Intervention

In this section, we analyze the performance degradation due to the application of safe controls from the SC and use it to motivate the study of better policy repair strategies. We assume that the initial learnt policy π_{θ_0} is given as a white-box and its parameter θ_0 has already been optimized for the control task. Inspired from lemma 1 in [30], we analyze the performance degradation of naive policy repair in a fixed-horizon task with maximum step length H. Recall the definition of cost function c in Section 3. Without loss of generality, we simplify it into a function of state, that is, from $c(x, u)$ to $c(x)$ and normalize it to the range $[0, 1]$. We use $\eta(\pi) = \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{H} c(x_t)]$ to denote the expected cumulative cost of following a policy π from initialization to step H. Define the value function $V_\pi(x_t) = \mathbb{E}_{x_t,u_t,x_{t+1}\ldots\sim\pi}[\sum_{l=t}^{H} c(x_l)]$ as the expected cost accumulated by following π after reaching state x_t at step t till step H. Define the state-action value function $Q_\pi(x_t, u_t) = \mathbb{E}_{x_t,x_{t+1},u_{t+1}\ldots\sim\pi,u_t}[\sum_{l=t}^{H} c(x_l)]$ as the expected cost accumulated by executing u_t in state x_t, then following π henceforth til step H. We use an advantage function $A_\pi(x_t, u_t) = Q_\pi(x_t, u_t) - V_\pi(x_t)$ to evaluate the additional cost incurred by applying control action u_t in x_t instead of adhering to π. Based on the lemma 1 in [30] for infinite-horizon scenario, we have the equation (9) for any two policies $\pi, \hat{\pi}$ in finite-horizon scenario.

$$\mathbb{E}_{\tau\sim\hat{\pi}}[\sum_{t=0}^{H} A_\pi(x_t, u_t)] = \mathbb{E}_{\tau\sim\hat{\pi}}[\sum_{t=0}^{H} c(x_t) + V_\pi(x_{t+1}) - V_\pi(x_t)]$$

$$= \mathbb{E}_{\tau\sim\hat{\pi}}[-V_\pi(x_0) + \sum_{t=0}^{H} c(x_t)] = \mathbb{E}_{x_0\sim d_0}[-V_\pi(x_0)] + \mathbb{E}_{\tau\sim\hat{\pi}}[\sum_{t=0}^{H} c(x_t)] = \eta(\hat{\pi}) - \eta(\pi) \quad (9)$$

Assuming that $\eta(\pi_{\theta_0})$ is the minimum for the desired task, i.e. π_{θ_0} is the optimal policy with respect to a cost function c, we bound the additional cost $\eta(\pi^{safe}) - \eta(\pi)$ incurred by possible interventions of π^{safe}.

Theorem 1. *Given a $\langle \pi_{\theta_0}, \pi_{safe} \rangle$ pair, let ϵ_1, ϵ_2 and ϵ_3 be the probability of $\langle \pi_{\theta_0}, \pi_{safe} \rangle$ generating a H-length trajectory where $\pi^{safe}(x, \pi_{\theta_0}(x)) \neq \pi_{\theta_0}(x)$ happens in at least one, two and three states respectively. Then, $\eta(\pi^{safe}) - \eta(\pi_{\theta_0}) \leq \epsilon_1 H + \epsilon_2(H-1) + \frac{\epsilon_3(H-1)H}{2}$.*

The theorem[6] shows the additional cost can grow quadratically in H when the probability of multiple interventions from the SC becomes higher. The implication of this is that even if the repaired policy π_{θ^*} replicates π^{safe} with zero error, the repaired policy can still suffer from significant performance degradation. Since the training error is non-zero in practice, $\pi_{\theta^*}(x) \neq \pi_{\theta_0}(x)$ may happen in more states where $\pi^{safe}(x, \pi_{\theta_0}(x)) \neq \pi_{\theta_0}(x)$. One major challenge in mitigating this performance loss is that the training information of π_{θ_0}, especially the cost function c, could be unknown. In the next section, we describe our approach of repairing a policy so that it also minimally deviates from the original one.

[6] Proof can be found in an extended version https://arxiv.org/abs/2008.07667

5.3 Minimally Deviating Policy Repair via Trajectory Synthesis

We firstly formally define the minimally deviating policy repair problem.

Definition 3. *Given an initial policy* π_{θ_0} *and an SC* π^{safe} *as defined in Definition 1,* ***minimally deviating policy repair*** *is the problem of finding a policy* π_{θ^*} *where* $\theta^* = \arg\min_{\theta \in \Theta} \mathbb{E}_{x \sim d_{\pi_\theta}}[e(x, \pi_{\theta_0}; \pi_\theta)]$ *subject to* $\pi^{safe}(x, \pi_\theta(x)) = \pi_\theta(x), \forall x \in X$.

Informally, the objective of this repair problem is to reduce the chance of $\pi_{\theta^*}(x) \neq \pi_{\theta_0}(x)$ while maintaining the safety of π_{θ^*}. Observe that the error term $e(\cdot)$ in Definition 3 resembles the one in an imitation learning setting. Then minimizing the expected error can be viewed as imitating π_{θ_0}. On the other hand, the equality constraint in Definition 3 can be understood as requiring π_{θ^*} to satisfy (7) at all steps in all its trajectories. Hence, the minimally deviating policy repair is essentially a problem of optimizing an imitation learning objective with safety constraints. The major challenge is that, the decision variable for the imitation learning objective is the policy parameter θ while for safety constraints (7) it is the state x.

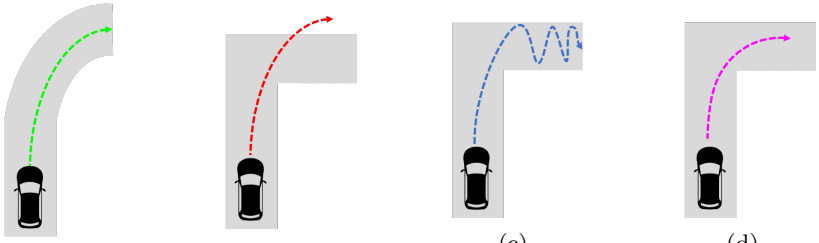

Fig. 2. (a) The grey area is the lane. The green dashed curve is the trajectory of the vehicle. (b) The red dashed curve is the trajectory of the initial policy. (c) The blue dashed curve is the trajectory of the policy and safety controller pair. (d) The magenta dashed curve is the trajectory produced by the repair policy that deviates minimally from the original one.

We use a simple example below to illustrate our problem setting and desired solution. Consider a policy that was trained to steer a vehicle around a specific corner as shown in Fig. 2(a). When deployed in a slightly different environment as shown in Fig. 2(b), the policy fails to keep the vehicle inside the lane. Fig. 2(c) illustrates that with the basic simplex setup as shown in Fig. 1, although the safety controller manages to keep the vehicle inside the lane, frequent switching between the two controllers can lead to undesirable behaviors such as an oscillating trajectory. Fig. 2(d) shows a more desirable trajectory produced by a new policy trained using minimally deviating policy repair. Our approach to the problem stated in Definition 3 is to 'imitate' the original policy by first synthesizing and then learning from new trajectories that are similar to ones produced by the original policy but instead do not violate the safety requirements. The

synthesis algorithm works by iteratively improving the trajectories produced by a naively repaired policy such as the one in Fig. 2(c) until trajectories such as the one in Fig. 2(d) are obtained. The improvement is achieved by solving a trajectory optimization problem of which the objective is transformed from the imitation learning objective in Definition 3. We mainly focus on showing such transformation in the rest of this section.

As mentioned in Section 3, to solve an imitation learning problem, we can minimize the KL-divergence which is related to maximal log-likelihood, i.e. $\arg\min_{\theta\in\Theta} D_{KL}[\pi_\theta||\pi_{\theta_0}] = \arg\max_{\theta\in\Theta} \mathbb{E}_{\tau\sim\pi_\theta}[\log Prob(\tau|\pi_{\theta_0})]$. Note that $Prob(\tau|\pi_\theta)$ is induced from a Dirac Delta distribution $u \sim \delta(\pi(x))$ and $Prob(\tau|\pi_{\theta_0})$ is carried out by adding to π_{θ_0} an isotropic Gaussian noise $\mathcal{N}(0, \Sigma)$ with diagonal $\Sigma = \sigma^2 I$. When a finite set Γ of trajectories of π_θ is obtained, the log-likelihood is equivalent to (10).

$$
\begin{aligned}
&\mathbb{E}_{\tau\sim\pi_\theta}[\log Prob(\tau|\pi_{\theta_0})] \approx \frac{1}{|\Gamma|}\sum_{\tau\in\Gamma}\log Prob(\tau|\pi_{\theta_0}) \\
\propto\ & \sum_{\tau\in\Gamma}\log\{\prod_{(x_t,u_t)\in\tau} exp[-\frac{(\pi_\theta(x_t)-\pi_{\theta_0}(x_t))^T\Sigma^{-1}(\pi(x_t,\theta)-\pi_{\theta_0}(x_t))}{2}]\} \\
\propto\ & -\frac{1}{2}\sum_{\tau\in\Gamma}\sum_{(x_t,u_t)\in\tau}||\pi_\theta(x_t)-\pi_{\theta_0}(x_t)||_2^2 \qquad (10)
\end{aligned}
$$

Suppose that at iteration $i \geq 1$, a safe policy π_{θ_i} is obtained and executed to generate a set Γ_i of safe traces. Define $l_{x_t,\pi_{\theta_i}} = \frac{1}{2}||\pi_{\theta_0}(x_t) - \pi_{\theta_i}(x_t)||_2^2$ and $J_{\Gamma_i}(\pi_{\theta_i}) = \sum_{\tau\in\Gamma_i}\sum_{(x_t,u_t)\in\tau} l_{x_t,\pi_{\theta_i}}$. To decrease J_{Γ_i}, a new policy parameter $\theta_{i+1} = \theta_i + \delta\theta_i$ can be obtained by solving $\delta\theta_i = \arg\min_{\delta\theta} J_{\Gamma_i}(\pi_{\theta_i+\delta\theta}) - J_{\Gamma_i}(\pi_\theta)$. We further use the Gauss-Newton step [19] to expand this as shown in (11) below.

$$
\begin{aligned}
&\arg\min_{\delta\theta}\ \delta\theta^T\nabla_\theta J_{\Gamma_i}(\pi_{\theta_i}) + \frac{1}{2}\delta\theta^T\nabla_\theta J_{\Gamma_i}(\pi_{\theta_i})\nabla_\theta J_{\Gamma_i}(\pi_{\theta_i})^T\delta\theta \\
=\ &\arg\min_{\delta\theta}\ \sum_{\tau\in\Gamma_i}\sum_{(x_t,u_t)\in\tau}\delta\theta_i\nabla_\theta\pi_{\theta_i}(x_t)\nabla_{\pi_{\theta_i}}l_{x_t,\pi_{\theta_i}} \\
&\qquad + \frac{1}{2}\delta\theta_i^T\nabla_\theta\pi_{\theta_i}(x_t)\nabla_{\pi_{\theta_i}}l_{x_t,\pi_{\theta_i}}\nabla_{\pi_{\theta_i}}l_{x_t,\pi_{\theta_i}}^T\nabla_\theta\pi_{\theta_i}(x_t)^T\delta\theta_i \qquad (11)
\end{aligned}
$$

We note that the changes of the policy control output $u_t = \pi_{\theta_i}(x_t)$ at arbitrary state x_t can be locally linearized as from (12) to (13).

$$
\begin{aligned}
&u_t + \delta u_t = \pi_{\theta_i+\delta\theta_i}(x_t+\delta x_t) \qquad u_t = \pi_{\theta_i}(x_t) && (12)\\
&\delta u_t^T - \delta x_t^T\nabla_x\pi_{\theta_i}(x_t) \approx \delta\theta_i^T\nabla_\theta\pi_{\theta_i}(x_t) && (13)
\end{aligned}
$$

It implies that due to $\delta\theta_i$, each trajectory $\tau = \{(x_0,u_0),(x_1,u_1),\ldots\}$ of π_{θ_i} is approximately perturbed by $\delta\tau = \{(\delta x_0,\delta u_0),(\delta x_1,\delta u_1),\ldots\}$. Motivated by the fact that $\pi_{\theta_i+\delta\theta_i}$ is safe if all of the trajectories are still safe after such perturbations, we optimize w.r.t the trajectory perturbations $\delta\tau$'s instead of $\delta\theta_i$ by exploiting the relation between each $(\delta x_t,\delta u_t) \in \delta\tau$ and $\delta\theta_i$ as in (13). Interpolating the LHS of (13) in (11), we obtain a trajectory optimization problem (14)

with linear and quadratic costs as shown in (15) $\sim$ (19). Note that this trajectory optimization problem treats the trajectories from Γ_i as nominal trajectories and solves for optimal perturbations to update those nominal trajectories. Local linearization is used to derive the dynamics constraints as in (20) for each noiminal trajectory. By adding the safety constraints (21), the trajectories can remain safe after adding the solved perturbations. Here, we use the constrained iLQR approach from [7] to resolve this constrained trajectory optimization problem.

$$\underset{\{\delta x_{0:H}, \delta u_{0:H}\}}{\arg\min} \quad \frac{1}{4|\Gamma_i|} \sum_{\tau \in \Gamma_i} \sum_{(x_t, u_t) \in \tau} \begin{bmatrix} 1 \\ \delta x_t \\ \delta u_t \end{bmatrix}^T \begin{bmatrix} 0 & Q_x^T & Q_u^T \\ Q_x & Q_{xx} & Q_{xu} \\ Q_u & Q_{xu}^T & Q_{uu} \end{bmatrix} \begin{bmatrix} 1 \\ \delta x_t \\ \delta u_t \end{bmatrix} \tag{14}$$

$$where\ Q_x = -2\nabla_x \pi_{\theta_i}(x_t) \nabla_{\pi_{\theta_i}} l_{x_t, \pi_{\theta_i}} \tag{15}$$

$$Q_u = 2\nabla_{\pi_{\theta_i}} l_{x_t, \pi_{\theta_i}} \tag{16}$$

$$Q_{xx} = \nabla_x \pi_{\theta_i}(x_t) \nabla_{\pi_{\theta_i}} l_{x_t, \pi_{\theta_i}} \nabla_{\pi_{\theta_i}} l_{x_t, \pi_{\theta_i}}^T \nabla_x \pi_{\theta_i}(x_t)^T \tag{17}$$

$$Q_{xu} = \nabla_x \pi_{\theta_i}(x_t) \nabla_{\pi_{\theta_i}} l_{x_t, \pi_{\theta_i}} \nabla_{\pi_{\theta_i}} l_{x_t, \pi_{\theta_i}}^T \tag{18}$$

$$Q_{uu} = \nabla_{\pi_{\theta_i}} l_{x_t, \pi_{\theta_i}} \nabla_{\pi_{\theta_i}} l_{x_t, \pi_{\theta_i}}^T \tag{19}$$

$$s.t. \quad \delta x_{t+1}^T = \delta x_t^T \nabla_x f(x_t, u_t) + \delta u_t^T \nabla_u f(x_t, u_t) \qquad t = 0, 1, \ldots, H-1 \tag{20}$$

$$\varphi(x_t + \delta x_t) \leq 0 \qquad t = 0, 1, 2, \ldots, H \tag{21}$$

One major benefit of this formulation is that *imitation learning objective and safety constraints can be reasoned at the same time via optimal control*. As the optimization is now separable, (14) $\sim$ (20) provide a lower bound for (11). By solving the linear equations (13), $\delta\theta_i$ can be inferred from the solved perturbations $\{\delta x_{0:H}, \delta u_{0:H}\}$, and then be used to modify θ_i. Alternatively, $\pi_{\theta_i + \delta\theta_i}$ can be obtained by training π_{θ_i} with the trajectories induced from $\{x_{1:H} + \delta x_{1:H}, u_{1:H} + \delta u_{1:H}\}$.

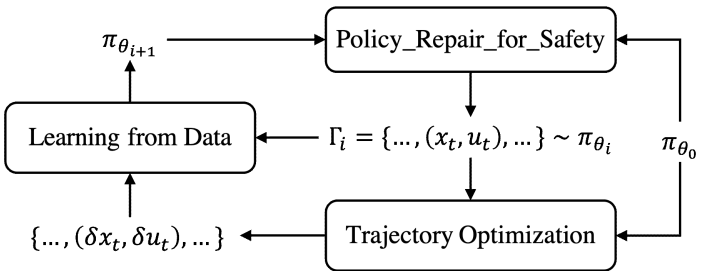

Fig. 3. Key steps in our minimally deviating policy repair algorithm. π_{θ_0} refers to the initial learnt policy.

The key steps of this iterative approach are shown in Fig.3 and the details are in Algorithm 2. As indicated by line 2 and 6, Algorithm 1 is used to find safe policies and generate safe nominal trajectories. This is because safe nominal trajectories guarantee that the trajectory optimization problem (14) $\sim$ (21) has feasible solutions, e.g. $\delta x = 0, \delta u = 0$. We terminate Algorithm 2 if Algorithm 1 fails to output a set of safe trajectories. In each iteration, we solve for the

trajectory perturbations in line 4 and use them to update the policy as shown in line 5. The algorithm ends in line 7 if the trajectory optimization step no longer helps in decreasing the deviation.

Algorithm 2 Policy_Repair_for_Minimal_Deviation

1: **Given** an initial learnt policy π_{θ_0}; iteration parameters $\epsilon \in [0,1], N > 1$.
2: **Initialization** Obtain π_{θ_1}, Γ_1 from Naive_Policy_Repair(π_{θ_0}) via **Algorithm 1**; **if** Γ_1 is $\emptyset$, **then return** *fail*
3: **for** iteration $i = 1$ to $i = N$ **do**
4: Solve the optimal $\{\delta x_{0:H}, \delta u_{0:H}\}$ from (14) $\sim$ (21).
5: Solve $\delta\theta_i$ via (13) and let $\theta_{i+1} = \theta_i + \delta\theta_i$.
Alternatively, search for $\theta_{i+1} = \arg\min_{\theta\in\Theta} \mathbb{E}_{(x,u)\sim\Gamma_i}[e(x+\delta x, u+\delta u; \pi_\theta)]$ by training π_{θ_i} with $\{(x+\delta x, u+\delta u) | (x,u) \in \Gamma_i\}$.
6: Obtain $\pi_{\theta_{i+1}}, \Gamma_{i+1}$ from Naive_Policy_Repair($\pi_{\theta_{i+1}}$) via **Algorithm 1**; **if** Γ_{i+1} is $\emptyset$, **then return** π_{θ_i}
7: **if** $|J_{\Gamma_{i+1}}(\pi_{\theta_{i+1}}) - J_{\Gamma_i}(\pi_{\theta_i})| \leq \epsilon$, **then return** $\pi_{\theta_{i+1}}$
8: **end for**
9: **return** π_{θ_N}

Complexity analysis. The main complexity of Algorithm 2 comes from solving the quadratic programming (QP) in (14) $\sim$ (21). Since cost (14) is convex as indicated by (10), if the constraint (21) is also convex, then the QP can be solved in polynomial time [18]; otherwise, it is NP-hard [21]. The trajectory optimization in line 4 needs to be solved only once off-line at the beginning of each iteration based on the safe trajectories collected from the previous iteration. In our experiments, the trajectory optimization is solved in a receding horizon manner as an MPC. In this case, the QP will be solved repeatedly over time to determine an appropriate sequence of control outputs. The nominal trajectories are obtained at each step by forward simulating the policy for a number of steps. The total computation time will be the same as that of a standard MPC. Besides trajectory optimization, the time complexity of policy updates in line 5 is either the same as that of solving an approximated linear equation (13) or training a neural network in a standard supervised manner.

6 Experiments

We perform two case studies to evaluate the effectiveness of our proposed approach. The key metrics of evaluation are (1) safety of the repaired policy and (2) performance preservation with respect to the original policy. The experiments were performed on a computer with the following configurations: Intel Core i7-8700 CPU @ 3.2GHz x 12 Processors, 15.4 GiB Memory, GeForce GTX 1050Ti, Ubuntu 16.04 LTS OS.

6.1 Mountaincar

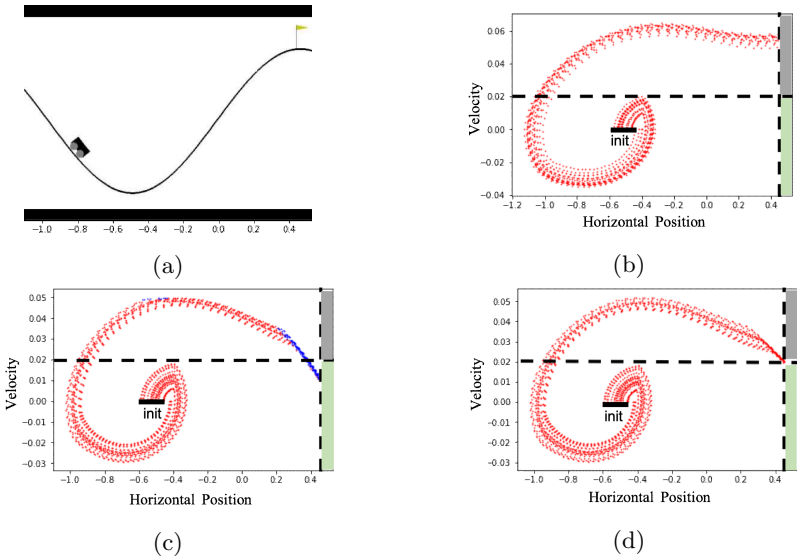

Fig. 4. (a) The mountaincar environment. (b) The red patterns represent a set of trajectories produced by executing the initial policy. The y-axis indicates the velocity and the x-axis indicates the horizontal position of the car. The car reaches the right mountain top in 83.8 steps on average with velocity higher than the safety threshold (0.02). (c) The interventions by the SC are indicated by the blue dots. A naively repaired policy takes the 89.3 steps on average to reach the mountaintop. (d) A minimally deviating repaired policy accomplishes the same task in 84.9 steps on average without violating the safety requirement.

Our first case study is Mountaincar[7], as shown in Fig.4(a). In this environment, the goal is to push an under-powered car from the bottom of a valley to the mountain top on the right with as few steps as possible. The state $\mathbf{x} = [p, v]$ includes the horizontal position $p \in [-1.2, 0.6]$ and the velocity $v \in [-0.07, 0.07]$ of the car. The control $u \in [-1.0, 1.0]$ is the force to be applied to the car. The car has a discrete-time dynamics that can be found in the source code the simulator. For the LC, we train a neural network policy via the Proximal Policy Optimization (PPO) algorithm [31]. The neural network takes the state variables as input and generates a distribution over the action space. An additional layer is added at the end of the network to calculate the expected action. In Fig.4$(b) \sim (d)$, the x and y axes indicate the horizontal position and the velocity respectively. The car starts from a state randomly positioned within $[-0.6, -0.4]$ as indicated by the black line above 'init'. The step length for the PPO-trained policy to reach the mountain top ($p \geq 0.45$) is 83.8 averaged over 1000 runs.

[7] https://gym.openai.com/envs/MountainCarContinuous-v0/

Now consider the safety requirement *'velocity v should not exceed* 0.02 *when reaching the mountain top at* $p \geq 0.45$*'*. The goal states and unsafe states are indicated by the green and grey areas in Fig.4(b). It can be observed that the PPO-trained policy does not satisfy this requirement as all the red trajectories in Fig.4(b) end up at $p = 0.45$ with $v > 0.02$. Then an SC is implemented by following the Model Predictive Safe Control scheme introduced in Section 4.1. The function $\varphi(x)$ in (7) evaluates whether the state x is in the grey unsafe area. The LC and SC pair generates the red trajectories in Fig.4(c). The blue dots indicate the intervention of the SC. While implementing Algorithm 1 and Algorithm 2, in each iteration we collect 20 trajectories in the trajectory set Γ_i. Algorithm 1 produces a naively repaired policy that can reach the green area with 89.3 steps on average. When using the minimally deviating policy repair algorithm (Algorithm 2), the resulting policy produces the red trajectories in Fig.4(d). It shows that in all the runs the resulting policy satisfies the safety requirement and in addition the SC does not intervene. In terms of performance, the policy reaches the green area with only 84.9 steps on average, which is much closer to the performance of the original policy.

6.2 Traction-Loss Event in Simulated Urban Driving Environment

In this experiment, we show that our approach is effective even with an approximate dynamical model. The environment is in an open urban driving simulator, CARLA [11], with a single ego car on an empty road. The state variables include position, velocity and yaw angle of the car and the control variables include acceleration and steering angles. We use a simple bicycle model from [7] to approximate the unknown dynamical model of the car. The model simulates a discrete-time system where the control actions are supplied to the system at an interval of $0.03s$. For the LC, an initial neural network policy is trained with data collected from manually driving the car on different empty roads while maintaining a speed of $8m/s$ and keeping the car to the middle of the lane. During testing, we put the vehicle in a roundabout as shown in Fig.5(a) where the white curves are the lane boundary. The starting and finishing lines are fixed. The safety requirement can be described informally as *'once the vehicle crosses outside a lane boundary, the controller should drive the vehicle back to the original lane within* 5 *seconds'*.

The initial, learnt policy drives the car well in the roundabout, as shown in Fig.5(a). We then consider an unforeseen traction-loss event, as shown by the yellow rectangle in Fig.5(a) where the friction is reduced to 0 (e.g. an icy surface). As a result, the vehicle skids out of the outer lane boundary. The initial policy alone does not satisfy the safety requirement, as it keeps driving the vehicle outside the lane boundary after the traction-loss event, as shown by the red trajectory in Fig.5(a). An SC is implemented by following the Model Predictive Safe Control scheme introduced in Section 4.1. The function $\varphi(x)$ in (7) checks whether the distance between the vehicle and the middle of the lane is larger than half of the lane width. In Fig.5(b), the blue segment indicates the

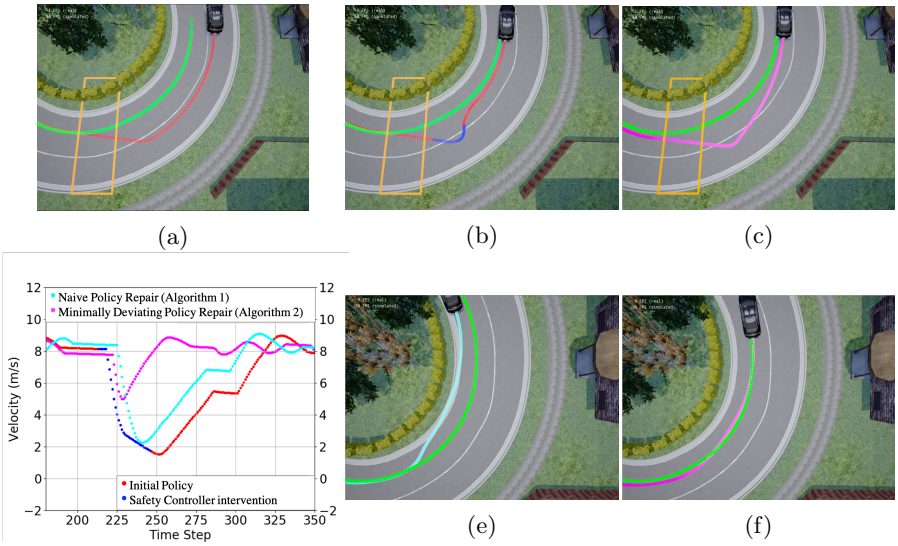

Fig. 5. The green trajectories represent normal trajectories of the car when there is no traction loss. The occurrence of the traction-loss event is indicated by the yellow rectangle. (a)Red trajectory: the initial policy fails to correct itself from skidding. (b) With interventions by the SC (the blue segment), the vehicle manages to return to the lane. (c) Magenta trajectory: policy repaired via Algorithm 2 corrects itself from skidding and does so better than using the SC. (d) The Y-axis represents velocity of the car and the X-axis represents time steps. The red curve indicates that the initial policy is in control and the blue segments represents the interventions from the SC. The cyan curve is generated by a policy repaired via Algorithm 1. The magenta curve is generated by a minimally deviating policy repaired via Algorithm 2. (e) Cyan trajectory: after the traction-loss area is removed, the naively repaired policy drives the vehicle towards the center of the roundabout, even going out the inner lane boundary for a short amount of time. (f) Magenta trajectory: after the traction-loss area is removed, by using Algorithm 2, the vehicle stays close to the middle of the lane.

interventions of the SC. It shows that due to the coupling of the LC and SC, the vehicle satisfies the safety requirement as it moves back to the lane.

We next consider repairing the LC using Algorithm 1 and 2. We set ϵ to 0.001 in our experiments. For every intermediate policy in each iteration, 10 trajectories are collected in its trajectory set Γ. It takes 5 iterations for Algorithm 1 to synthesize a safe policy that does not require the SC to intervene. Starting with this safe policy, Algorithm 2 runs for 25 iterations before termination. The magenta trajectory in Fig.5(c) is from the minimally deviating policy repaired via Algorithm 2. Obviously the policy is able to correct itself without any intervention from the SC. In Fig.5(d), we compare the velocities of the vehicles controlled by different policies. It can be observed that the velocities of all policies drop drastically due to traction-loss at around step 220. The minimally deviating repaired policy performs the best in restoring the velocity back to

$8m/s$. It is worth noting that velocity stability is important from the viewpoint of passenger comfort.

We summarize the results in Table.1. The performances of the algorithms are evaluated from multiple aspects. We evaluate how well the task is finished from 1) average speed (the closer to the target speed $8m/s$ the better); 2) average distance of the vehicle to the middle of the lane (the smaller the better); 3) total time taken for the driving task in number of simulation steps (the fewer the better). We evaluate the smoothness of the trajectories based on the variances/standard deviations of speeds, distances, changes of speed and distance respectively in consecutive time steps. A smaller variance/standard deviation indicates a smoother navigation.

Table 1. Avg. Speed: average speed of the vehicle in each run. Lowest Speed: the lowest speed since the vehicle firstly reaches $8m/s$ in each run. Aveg. Distance: the average distance between the vehicle and the middle of the lane at each step. Tot. Steps: the total number of steps that the vehicle outputs control actions in one run. Var. Speed: the variance of the speed at each step in each run. Std Dev. Speed Change: the standard deviation of the speed changes between consecutive steps. Var. Distance: the variance of the distance between the vehicle and the middle of the lane at each step. Std Dev. Distance Change: the standard deviation of the distance (from vehicle to the middle of the lane) changes between consecutive steps. Initial policy is tested before and after the traction-loss area is placed. The initial policy and SC pair is tested after the traction-loss event occurs. 'Algorithm 1' and 'Algorithm 2' respectively refer to the policies repaired via those two algorithms.

	Avg. Speed(m/s)	Lowest Speed(m/s)	Avg. Distance(m)	Tot. Steps ($0.03s/step$)
Initial Policy (No Traction-Loss Event)	8.0	7.1	0.27	396
Initial Policy	8.0	5.2	1.7	420
Initial Policy w/ SC	7.1	1.2	0.81	454
Algorithm 1	7.5	2.4	1.1	440
Algorithm 2	**7.9**	**5.2**	**0.63**	**413**
	Var. Speed	Std Dev. Speed Change	Var. Distance	Std Dev. Distance Change
Initial Policy (No Traction-Loss Event)	0.53	0.074	0.10	0.0096
Initial Policy	0.79	0.16	4.4	0.026
Initial Policy w/ SC	2.1	0.17	1.0	0.033
Algorithm 1	2.4	0.17	1.4	0.042
Algorithm 2	**0.73**	**0.15**	**1.0**	**0.033**

Before the traction-loss area is placed, the initial policy drives the vehicle at $8m/s$ on average and keeps the vehicle close to the middle of the lane. After the traction-loss event occurs, the initial policy still maintains the speed but the car slides out of the lane as indicated by the average distance. The initial policy and SC pair has the lowest average and lowest speed. As a result, the task

time (represented by number of simulation steps) is also the longest. In terms of policy repair, both Algorithm 1 and 2 are successful in finding safe policies. The policy repaired via Algorithm 1 behaves similar to the initial policy and SC pair – the vehicle experiences significant speed changes and takes longer to finish the driving task. The minimally deviating policy repaired via Algorithm 2 behaves similarly to the initial policy in terms of maintaining the target speed, staying close to the middle of the lane while producing a smooth trajectory. In summary, the repaired policy using Algorithm 2 outperforms the initial policy with SC and the repaired policy using solely Algorithm 1 in almost all metrics. In terms of runtime overhead savings, the average neural network inference time on our machine configuration is $0.0003s$ while the average time for SC to solve (3) $\sim$ (8) is $0.39s$.

To further measure the impact of policy repair and evaluate the performance difference between a naive repair (using Algorithm 1) and a minimally deviating repair (using Algorithm 2), we remove the traction-loss area and execute both repaired policies in the original environment. It can be observed in Fig.5(e) that the naively repaired policy cuts inside the lane, since it learns (possibly due to overfitting) to steer inward in the states where traction loss is supposed to occur. In contrast, the policy repaired using Algorithm 2 manages to keep the car in the lane, as it learns to imitate the original policy. This thus further validates our approach of finding a minimally deviating repair.

7 Conclusion

We consider a Simplex architecture where a learning-based controller is paired with a backup safety controller for ensuring runtime safety. We show that this setup, while provides added safety assurance, can produce undesired outputs or cause significant performance degradation. We propose to address this problem by fine-tuning the learning-based controller using interventions from the safety controller, and addressing the issue of performance degradation via imitation learning. Our experiments indicate that our proposed approach is effective in achieving both safety and performance even when the dynamical model used by the safety controller is not exact. In the future, we plan to consider other types of safety controllers and extend our techniques to end-to-end control settings.

Acknowledgements. We gratefully acknowledge the support from National Science Foundation (NSF) grants 1646497.

References

1. J. Achiam, D. Held, A. Tamar, and P. Abbeel. Constrained policy optimization. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 22–31. JMLR. org, 2017.
2. M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu. Safe reinforcement learning via shielding. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
3. D. Amodei, C. Olah, J. Steinhardt, P. F. Christiano, J. Schulman, and D. Mané. Concrete problems in AI safety. *CoRR*, abs/1606.06565, 2016.
4. B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.
5. D. Bareiss and J. Van den Berg. Reciprocal collision avoidance for robots with linear dynamics using lqr-obstacles. In *2013 IEEE International Conference on Robotics and Automation*, pages 3847–3853. IEEE, 2013.
6. F. Borrelli, T. Keviczky, and G. J. Balas. Collision-free uav formation flight using decentralized optimization and invariant sets. In *2004 43rd IEEE Conference on Decision and Control (CDC)(IEEE Cat. No. 04CH37601)*, volume 1, pages 1099–1104. IEEE, 2004.
7. J. Chen, W. Zhan, and M. Tomizuka. Constrained iterative lqr for on-road autonomous driving motion planning. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–7, Oct 2017.
8. R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick. End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks. *ArXiv*, abs/1903.08792, 2019.
9. Y. Chow, O. Nachum, E. Duenez-Guzman, and M. Ghavamzadeh. A lyapunov-based approach to safe reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 8092–8101, 2018.
10. J. A. DeCastro and H. Kress-Gazit. Guaranteeing reactive high-level behaviors for robots with complex dynamics. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 749–756. IEEE, 2013.
11. A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. CARLA: An open urban driving simulator. In S. Levine, V. Vanhoucke, and K. Goldberg, editors, *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 1–16. PMLR, 13–15 Nov 2017.
12. N. Fulton and A. Platzer. Safe reinforcement learning via formal methods: Toward safe control through proof and learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
13. H. Kwakernaak and R. Sivan. *Linear optimal control systems*, volume 1. Wiley-interscience New York, 1972.
14. W. Li and E. Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *ICINCO*, 2004.
15. J. Maciejowski. Predictive control: with constraints, 2002.
16. O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer, 2004.
17. K. Menda, K. Driggs-Campbell, and M. J. Kochenderfer. Ensembledagger: A bayesian approach to safe imitation learning. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5041–5048, 2019.

18. Y. Nesterov and A. Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. Society for Industrial and Applied Mathematics, 1994.
19. J. Nocedal and S. Wright. *Numerical optimization*. Springer Science & Business Media, 2006.
20. T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters. An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, 7(1-2):1–179, 2018.
21. P. M. Pardalos and S. A. Vavasis. Quadratic programming with one negative eigenvalue is np-hard. *Journal of Global Optimization*, 1:15–22, 1991.
22. M. Pereira, D. D. Fan, G. N. An, and E. A. Theodorou. Mpc-inspired neural network policies for sequential decision making. *CoRR*, abs/1802.05803, 2018.
23. D. Phan, N. Paoletti, R. Grosu, N. Jansen, S. A. Smolka, and S. D. Stoller. Neural simplex architecture. *ArXiv*, abs/1908.00528, 2019.
24. A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57. IEEE, 1977.
25. D. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *NIPS*, 1988.
26. V. Raman, A. Donzé, M. Maasoumy, R. M. Murray, A. Sangiovanni-Vincentelli, and S. A. Seshia. Model predictive control with signal temporal logic specifications. In *53rd IEEE Conference on Decision and Control*, pages 81–87. IEEE, 2014.
27. S. Ross and D. Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668, 2010.
28. S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.
29. D. Sadigh and A. Kapoor. Safe control under uncertainty with probabilistic signal temporal logic. In *Proceedings of Robotics: Science and Systems XII*, June 2016.
30. J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.
31. J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.
32. D. Seto, B. Krogh, L. Sha, and A. Chutinan. The simplex architecture for safe online control system upgrades. In *Proceedings of the 1998 American Control Conference. ACC (IEEE Cat. No.98CH36207)*, volume 6, pages 3504–3508 vol.6, 1998.
33. Y. Tassa, T. Erez, and E. Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4906–4913, Oct 2012.
34. K. P. Wabersich and M. N. Zeilinger. Linear model predictive safety certification for learning-based control. *2018 IEEE Conference on Decision and Control (CDC)*, pages 7130–7135, 2018.
35. C.-J. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia, et al. Machine learning at facebook: Understanding inference at the edge. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 331–344. IEEE, 2019.
36. J. Zhang and K. Cho. Query-efficient imitation learning for end-to-end simulated driving. In *AAAI*, 2017.
37. W. Zhou and W. Li. Safety-aware apprenticeship learning. In *International Conference on Computer Aided Verification*, pages 662–680. Springer, 2018.