Learning from Interventions Using Hierarchical Policies for Safe Learning

Jing Bi,[1] Vikas Dhiman,[2] Tianyou Xiao,[1] Chenliang Xu[1]
[1]University of Rochester
[2]University of California San Diego
jbi5@ur.rochester.edu, vdhiman@ucsd.edu
txiao3@u.rochester.edu, chenliang.xu@rochester.edu

Abstract

Learning from Demonstrations (LfD) via Behavior Cloning (BC) works well on multiple complex tasks. However, a limitation of the typical LfD approach is that it requires expert demonstrations for all scenarios, including those in which the algorithm is already well-trained. The recently proposed Learning from Interventions (LfI) overcomes this limitation by using an expert overseer. The expert overseer only intervenes when it suspects that an unsafe action is about to be taken. Although LfI significantly improves over LfD, the state-of-the-art LfI fails to account for delay caused by the expert's reaction time and only learns short-term behavior. We address these limitations by 1) interpolating the expert's interventions back in time, and 2) by splitting the policy into two hierarchical levels, one that generates sub-goals for the future and another that generates actions to reach those desired sub-goals. This sub-goal prediction forces the algorithm to learn long-term behavior while also being robust to the expert's reaction time. Our experiments show that LfI using sub-goals in a hierarchical policy framework trains faster and achieves better asymptotic performance than typical LfD.

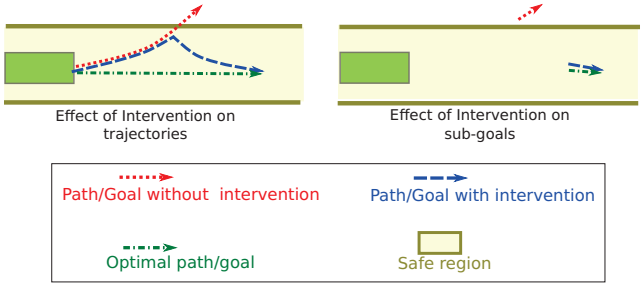

Figure 1: An autonomous driving example to show the advantage of intervening on sub-goals rather than trajectories. An expert-overseer intervenes whenever the car is about to go outside the safe region, with some delay. The figure on the left shows three trajectories: without intervention, with intervention, and the optimal trajectory. There is a difference between the intervened trajectory and the desired trajectory due to the expert's reaction time. However, by formulating the intervention in terms of sub-goals, we can minimize this discrepancy, as shown in the right figure.

Introduction

Methodologies for learning control policies can be categorized into Reinforcement Learning (RL) and Imitation Learning (IL). RL depends on the availability of a well-designed reward function, while IL depends on demonstrations from an expert. Recently, RL has had success in simulated environments and games (Mnih et al. 2015; Silver et al. 2016). However, in many real-world robotic applications like autonomous driving, it is challenging to design explicit reward functions. Moreover, training by trial-and-error in the real world is highly unsafe. For these reasons, IL is often preferable to RL.

There are two main categories of Imitation Learning: Inverse Reinforcement Learning (IRL) and Behavior Cloning (BC), which is also known as Learning from Demonstration (LfD). In IRL, the algorithm learns a reward function from expert demonstrations, then learns the optimal policy through trial and error. On the other hand, BC directly learns the policy to match expert demonstration, without ever learning the reward function. In safety-critical scenarios, BC is preferred over IRL because it requires no trial-and-error during training. However, BC requires a large number of data samples, as it depends on the demonstration data in all scenarios, even if the algorithm has learned the scenarios satisfactorily.

(Saunders et al. 2018) and (Waytowich, Goecks, and Lawhern 2018) have recently introduced the framework of Learning from Interventions (LfI) that allows for safe RL in the presence of a human overseer. This approach guarantees the safety and is shown to need fewer samples from an expert than LfD. State-of-the-art approaches in both LfD and LfI learn reactive behaviors (Tai et al. 2016) and very little long-term behavior. Moreover, no LfI framework so far considers the reaction delay in modeling expert supervision. We find that the expert only signals after a non-negligible amount of delay and that compensating for these delays leads to improvements in both sample efficiency and performance.

To address both of these challenges in LfI, we employ ideas from Hierarchical Reinforcement Learning (Kulkarni et al. 2016; Savinov, Dosovitskiy, and Koltun 2018) by mod-

Figure 2: Top view of the map in CARLA (Dosovitskiy et al. 2017) simulator where experiments were conducted.

eling our policy as multiple hierarchical policies operating at different time-scales, one focused on the long-term and another focused on the short-term. Also, as demonstrated in Fig. 1, long term predictions are less likely to suffer from an expert's reaction delay. Therefore our model can compensate for the expert's reaction time.

Our proposed approach is as follows. We start with conditional IL (Codevilla et al. 2018) as our non-hierarchical baseline. All our policies are *conditioned* on a command signal, which represents a high-level instruction, such as whether to turn left or right at an intersection. We split the policy into two hierarchy levels, where the top-level policy predicts a *sub-goal* to be achieved k steps ahead in the future. The bottom-level policy chooses the best action to achieve the *sub-goal* generated by the top-level policy. We assume any achieved state is a desirable goal from the perspective of a past state - embeddings from the present safe state are used as a sub-goal for the past-state, similar to in Hindsight Experience Replay (Andrychowicz et al. 2017). Since we work with images as our state space, we do not generate sub-goals in the image space, as it is unnecessary to predict future images at pixel-level detail when we only want to reach the sub-goal. Instead, we generate intermediate sub-goal embeddings using Triplet Networks (Hoffer and Ailon 2015). The details of our approach are discussed further in the Methods section.

We evaluate our proposed system and relevant baselines on the task of autonomous driving in a realistic simulated environment of urban driving. The agent is spawned in a random location on the map. It navigates autonomously according to a controller's high-level commands. To be successful at this task, the agent must stay in its lane and not crash into buildings or other vehicles. Our results suggest that by combining expert intervention and hierarchical policies, our method helps the agent in two fundamental ways: Firstly, it helps the agent to improve its policy online without taking unsafe actions. Secondly, hierarchical policies force the agent to predict and plan longer-term behaviors, leading to more effective use of data.

To summarize, our contributions are three-fold:

1. We propose a *new problem formulation* of Learning from Interventions that incorporates the expert's reaction delay.
2. We propose a *novel algorithm* to address the proposed problem, which combines Learning from Interventions with Hierarchical Reinforcement Learning.
3. We propose an interpolation trick, called *Backtracking*, that allows us to use state-action pairs before and after the intervention.
4. We propose a *novel Triplet-Network based architecture* to train the hierarchical policies in the absence of ground-truth sub-goals.

We further demonstrate the practical effectiveness of our approach on a real-world RC robotic truck. Experiments with the physical system demonstrate that our approach can be successfully deployed in the physical world.

Related Work

Imitation Learning

The approaches for Imitation Learning can be categorized into Behavior Cloning (Ross et al. 2013; Giusti et al. 2016) and Inverse Reinforcement Learning (Abbeel and Ng 2004). For the scenarios, when high-level command instructions are available, Codevilla et al. (Codevilla et al. 2018) proposed conditional IL method that learns conditional policies based on the high-level command instructions.

Behavior Cloning might lead to unsafe actions, especially in the less frequently sampled states (Argall et al. 2009). Therefore, in real-world scenarios, the expert's oversight and interaction are necessary to help the agent train without failure and unsafe actions.

The approaches to Learning from Interventions (LfI) can be categorized according to the frequency of the expert's engagement. The expert's engagement varies from high frequency in Learning from Demonstrations (Akgun et al. 2012b; Zhang and Cho 2017), medium frequency in learning from Interventions (Akgun et al. 2012a; Bi et al. 2018) to low frequency in Learning from Evaluations (Knox and Stone 2009; MacGlashan et al. 2017). Recent work has applied LfI to safe RL (Saunders et al. 2018) and IL (Ho and Ermon 2016), which allows the agent to perform a certain task without entering unsafe states. In (Hilleli and El-Yaniv 2018), a classifier is trained from the expert demonstration that used to separate safe states from undesired states. When an undesired state is detected, another policy is activated to take over actions from the agent when necessary. In several works (Hilleli and El-Yaniv 2018; Peng et al. 2018), human intervention is often used to reshape the reward of RL, which encourages the agent to avoid certain states. The work closest to our approach is (Goecks et al. 2019), which proposes *Cycle-of-Learning* that combines different interaction modalities into a single learning

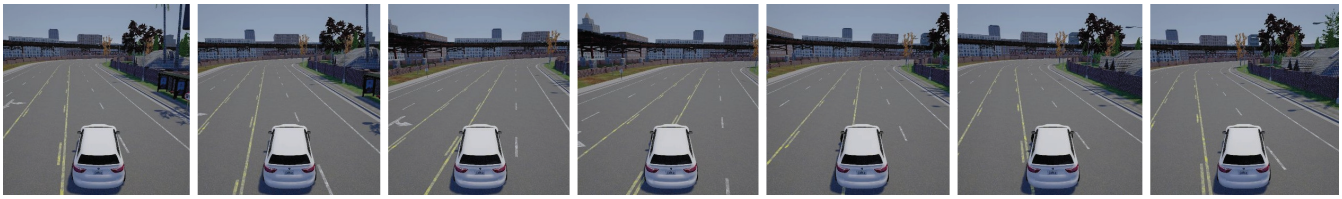

Figure 3: Snapshots from the CARLA simulator (Dosovitskiy et al. 2017) showing that the car is already out of lane by image (1), but that the expert does not intervene until image (4) due to reaction delay and veers it back to desired trajectory (4-7).

framework. This method allows the expert to intervene by providing corrective actions when the agent encounters unwanted states and further aggregates these states and actions into the training dataset. The intervention data from failure states help the agent better generalize and overcome the deficiencies in its policy. We further discuss the effect of states not only used to recover from the unwanted states but also the states, which will lead to failure.

Inverse Dynamics

Learning a supervised policy is known to have "myopic" strategies, since it ignores the temporal dependence between consecutive states. In contrast, for making informative decisions, classical planning approaches in robotics can reason far into the future, but often require designing sophisticated cost functions. Savinov et al. (Savinov, Dosovitskiy, and Koltun 2018) propose to directly model Inverse dynamics to represent a trade-off between learning robust policies for long term planning. Formally, the inverse dynamics is probability of taking an action a_t given the current state s_t and next states s_{t+k}, i.e. $\mathcal{P}(a_t|s_t, s_{t+k})$. For instance, if $k = 1$, the network learns action to perform in one-step transitions with neighbor states. For $k > 1$, training data become noisier but sufficient for short-term predicting (Savinov, Dosovitskiy, and Koltun 2018).

Recent works use inverse dynamics to learn features for recognition tasks (Agrawal, Carreira, and Malik 2015) and control tasks (Dosovitskiy and Koltun 2016). In (Agrawal et al. 2016), the authors constructed a joint inverse-forward model to learn feature representation for the task of pushing objects. In (Pathak et al. 2017), an Intrinsic Curiosity Module was introduced to tackle the exploration problem in RL, which uses Inverse Dynamic Model to predict action based on current and future state and a forward model to predict the future state. In this way, the agent explores the unfamiliar states more than familiar ones. The work shows that predicting the future state helps the agent to plan better. We use the inverse dynamics module as the bottom-level policy in our proposed method.

In (Le et al. 2018) and (Yu et al. 2018), the high-level algorithm focuses on decomposing the multi-stage human demonstration into primitives actions, and the low-level policy executes a series of actions to accomplished the primitive. In our model, we generate sub-goals that are feature embeddings directly generated by the network, instead of the primitive actions which have precise semantic meanings like "go to the elevator," then "take the elevator down." Therefore, our approach does not need the design of a library of primitive actions.

Problem Formulation

We consider a set-up similar to (Saunders et al. 2018): an agent interacting with an environment in discrete time steps. We assume that this process is a fully observable Goal-conditioned Markov Decision Process $(\mathcal{S}, \mathcal{A}, T, R, \mathcal{C})$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $T : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}$ is the unknown transition model and $R : \mathcal{S} \times \mathcal{A} \times \mathcal{C} \mapsto \mathbb{R}^+$ is the unknown reward function that depends upon the command vector $c_t \in \mathcal{C}$ for specifying the expert's intention. At each time step t, the agent gets an observation $o_t \in \mathcal{O}$ (sequence of images), and a command c_t from the expert, then takes action a_t. We assume observations provide full information of the current state $s_t \in \mathcal{S}$ through a feature extraction mapping $s_t = \psi(o_t; \theta_e)$. Instead of reward function, we only have access to the expert with policy $\pi^* : \mathcal{O} \times \mathcal{C} \mapsto \mathcal{A}$, who can intervene whenever s/he perceives a probability of catastrophic damage is higher than a threshold: $p_c^*(s_{t-k^*}, a_{t-k^*}) > 1 - \delta_c$. So the effective policy followed with expert intervention is given by:

$$\pi_{\text{eff}}(s_t, c_t) = \begin{cases} \pi^*(s_t, c_t), & \text{if } p_c^*(s_{t-k^*}, a_{t-k^*}) > 1 - \delta_c, \\ \pi(s_t, c_t; \theta), & \text{otherwise}, \end{cases} \tag{1}$$

where $\pi(s_t, c_t; \theta)$ is the learned policy so far, and k^* is the unknown expert's reaction delay. Our aim is to improve the learned $\pi(s_t, c_t; \theta)$ without any catastrophic failure:

$$\theta^* = \arg \min_{\theta} \sum_{t \in \tau_{\text{intervention}}} \|\pi_{\text{des}}(s_t, c_t) - \pi(s_t, c_t; \theta)\|_{\mathcal{A}}, \tag{2}$$

where $\tau_{\text{intervention}}$ is the set of data-samples affected by interventions and π_{des} is the desired policy which is a heuristic approximation of π_{eff} that accounts for the expert's reaction delay.

Methods

We present two approaches to address the problem of delayed expert, *hierarchical policies* and *backtracking*. Please note that both these approaches are not mutually exclusive and can be combined together.

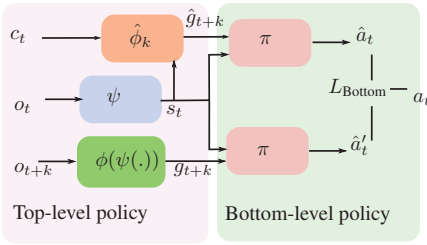

Figure 4: Network diagram for our hierarchical policy. We split our policy into two levels of a hierarchy. The top-level policy, $\hat{\phi}_k$, generates sub-goals $\hat{g}_{t+k}$ that the bottom-level policy π follows. We also generate "true sub-goals" g_{t+k} using a goal encoder ϕ on future observation o_{t+k}. The loss $\mathcal{L}_{\text{Bottom}}$ compares predicted actions $\hat{a}_t$ and $\hat{a}'_t$ with true intervention actions a_t. We train the goal-encoder and top-policy togther using the TripletNet by comparing "true sub-goals" with randomly generated sub-goals.

Learning from Intervention using Hierarchical policies

Instead of learning a direct policy network $\pi(s_t, c_t; \theta)$, we divide the policy learning into two levels of a hierarchy. The top-level policy $\hat{\phi}_k$ learns to predict a sub-goal vector $\hat{g}_{t+k}$, k steps ahead, while the bottom-level policy π learns to generate action to achieve that sub-goal. The full policy $\pi(s_t, c_t; \theta)$ can be written as nested policies:

$$\pi(o_t, c_t; \theta) = \pi(s_t, \hat{g}_{t+k}; \theta_b), \tag{3}$$

$$\hat{g}_{t+k} := \hat{\phi}_k(s_t, c_t; \theta_t), \tag{4}$$

where $s_t = \psi(o_t; \theta_e)$, and $\hat{g}_{t+k}$ is the predicted sub-goal by the top-level policy.

We do not have access to the ground truth value of $\hat{g}_{t+k}$ from the expert policy π^*. In order to learn $\hat{\phi}_k(.; \theta_t)$, we need another network $g_{t+k} := \phi(s_{t+k}; \theta_g)$, which outputs the desired sub-goal embedding g_{t+k} from an achieved state $s_{t+k} = \psi(o_{t+k}; \theta_e)$. This strategy is also called Hindsight Experience Replay, because we consider an achieved goal as a desired goal for past observations (Andrychowicz et al. 2017). We train $\hat{\phi}_k$ and ϕ together by adapting a Triplet Network (Hoffer and Ailon 2015), which provides a loss that brings closer the embeddings $\hat{g}_{t+k}$ and g_{t+k} while pushing away the embedding $\hat{g}_{t+k}$ from a randomly chosen goal embedding $g_r = \phi(s_r; \theta_g)$. The split of the overall policy into two levels while using the future state as the sub-goal embedding is summarized in Figure 4.

Training the bottom-level policy Given the state embedding s_t and sub-goal embedding $\hat{g}_{t+k} = \hat{\phi}_k(s_t, c_t; \theta_t)$, we want to optimize the parameter θ_b for bottom-level policy. Let $\hat{a}_t = \pi(s_t, \hat{g}_{t+k}; \theta_b)$ be the action prediction and $a_t = \pi^*(o_t, c_t)$ be the optimal action from the expert.

For our car experiments, the action space is composed of two actions $\mathcal{A} = \mathcal{A}_s \times \mathcal{A}_b$, where the steering angle is continuous $\hat{a}_{t,s} \in \mathcal{A}_s = [-1, 1]$, and the brake is binary $\hat{a}_{t,b} \in \mathcal{A}_b = \{0, 1\}$. In this way, we have:

$$L_{\text{Bottom}}(\hat{a}_t, a_t) = L_{\text{BCE}}(\hat{a}_{t,b}, a_{t,b}) + \|a_{t,s} - \hat{a}_{t,s}\|_2^2, \tag{5}$$

where L_{BCE} is the Binary Cross Entropy loss.

Training the top-level policy As discussed earlier, we want to predict the future sub-goal from the current state using a top-level policy $\hat{g}_{t+k} = \hat{\phi}_k(s_t, c_t; \theta_t)$. Since we do not have access to the true sub-goal embedding, we generate a sub-goal embedding from a future achieved observation o_{t+k} and treat it as the true sub-goal embedding $g_{t+k} = \phi(\psi(o_{t+k}; \theta_e); \theta_g)$.

It is important to note why we prefer Triplet Network over other options. We critique two naïve approaches: Auto-encoder approach and Direct minimization approach.

Auto-encoder approach can be summarized as training auto-encoders to predict goal-images. However, the prediction in the image space is undesirable because it is unnecessarily hard and not useful for the purpose of describing task-goals. For example (from (Pathak et al. 2017)), there is no use predicting tree leafs in a breeze, when instructing a car to go near the tree.

Direct minimization approach minimizes $\|\hat{\phi}(s_t, c_t; \theta_t) - \phi(s_{t+k}; \theta_g)\|$ directly. Although this approach is intuitive, it does not give desirable results. The reason is that there exists a trivial solution when the network outputs a constant embedding irrespective of the input, which makes the loss to be always zero.

Our goal is to assign high similarity to the pair of states that is temporally aligned, and differentiate all other $o_\tau : \tau \neq t + k$ in the goal embedding space. This is the main idea behind Triplet Loss (Hoffer and Ailon 2015). In Figure 5, each training sample contains two triples $[(s_t, s_{t+k}, y_1), (s_t, s_r, y_2)]$, and each triple consists of two observations and a binary label. Two observations are considered close $y_1 = 1$ if they are separated by exactly k time steps. In order to differentiate these two states with all others, we add a random observations s_r with label $y_2 = 0$. This structure is shown in Fig. 5, and we have:

$$\text{TripletNet}(s_t, s_{t+k}, s_r) = \begin{bmatrix} \underbrace{\|\hat{\phi}_k(s_t, c_t) - \phi(s_{t+k})\|}_{d_+} \\ \underbrace{\|\hat{\phi}_k(s_t, c_t) - \phi(s_r)\|}_{d_-} \end{bmatrix}, \tag{6}$$

$$L_{\text{Triplet}}(s_t, s_{t+k}, s_r) = \left(\frac{\exp(d_+)}{\exp(d_+) + \exp(d_-)} \right)^2. \tag{7}$$

The idea of Triple-Net is illustrated in Figure 5. Each of the input observation o_t is embedded as a vector $\psi(o_t)$ using an encoder based on ResNet50 (He et al. 2016).

By learning the bottom-level policy and top-level policy jointly, we yield to the following objective function:

$$\begin{aligned} \theta_t^*, \theta_g^*, \theta_b^* = \arg\min_{\theta_{t,g,b}} \sum_{t \in D} \{ & L_{\text{Bottom}}(\pi(s_t, \hat{g}_{t+k,\theta_t}; \theta_b), a_t) \\ & + L_{\text{Bottom}}(\pi(s_t, g_{t+k,\theta_g}; \theta_b), a_t) \\ & + L_{\text{Triplet}}(s_t, s_{t+k}, s_r; \theta_t, \theta_g) \}. \end{aligned} \tag{8}$$

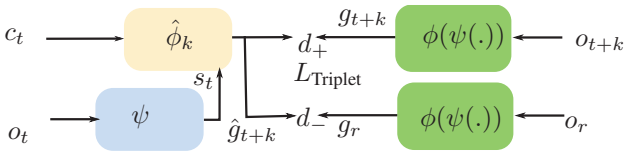

Figure 5: We train goal-embeddings by using Triplet Network (Hoffer and Ailon 2015). The matching pair of goal-embeddings $(\hat{g}_{t+k}, g_{t+k})$ is considered as a positive example, while a random goal-embedding g_r is used to construct a negative example $(\hat{g}_{t+k}, g_r)$. Refer to Eq (7) for details.

Desired policy approximation The main claim of our work is that using hierarchical sub-goals helps in constructing a better desired policy π_{des}. As constructed above, we assume that we can factorize the π_{des} into two levels:

$$\begin{cases} \pi_{\text{des}}(s_t, c_t) & := \pi_{\text{eff}}(s_t, g_{t+k,\theta_g}), \\ g_{t+k,\theta_g} & = \phi(s_{t+k}; \theta_g). \end{cases} \tag{9}$$

In order to achieve a good approximation for above functions, we need two assumptions. Firstly, we assume that reaction-delay of the expert does not change the achieved state s_{t+k} down the trajectory (as in Fig. 1). Secondly, we need to assume that the lower-level policy $\pi_{\text{eff}}(s_t, g_{t+k,\theta_g})$ following sub-goals can be learned more easily and in fewer iterations than the full policy. These two assumptions depend on the value of k. The first assumption holds well for large ks. For example, a large k means that we choose a far away s_{t+k} as a proxy for the goal. Even with a large expert reaction delay, the trajectory is likely to return to desired trajectory by time $t + k$. The second assumption holds well for small ks. Consider the example of $k = 1$, where the direction of movement can almost be inferred by optical flow. Because of this trade-off on k, we conduct experiments to evaluate the effect of k on our algorithm. The details are discussed in the Experiments section.

Learning from intervention by Backtracking

To construct a better approximation of the expert's desired behavior, we collect data not only at the point of intervention but also for a few time steps around it. Therefore, during the data collection phase, we keep track of past M time-steps in a data queue, D_β. Whenever the intervention happens at time t, we interpolate the actions between a_{t-M} and the intervened action a_t^* to update the action in data-queue D_β. The Backtracking function, Backtrack(a_t^*, a_{t-M}, j), interpolates the actions to time step $t - j$. This data-collection algorithm is summarized in Alg 1.

Algorithm 1 Learn-form-Intervention by Backtracking

Input: Parameter M: the number of time-steps before intervention time t s.t. a_{t-M} is a desired action, maximum episode length T, policies $\pi_k(s_t, c_t)$, and the demonstration data from expert D_h
Output: Updated policies $\pi_{k+1}(s_t, c_t)$

repeat
  Sample a new state s_0 from the environment
  Initialize data queue $D_\beta \leftarrow \emptyset$ $D_\alpha \leftarrow \emptyset$
  for $t = 1, \ldots, T$ **do**
    Sample a new state s_t from the environment
    if no Intervention **then**
      Execute $a_t = \pi_k(s_t, c_t)$
      Append (s_t, a_t) to D_β
      if len$(D_\beta) > M$ **then**
        POP(D_β)
      if len$(D_\alpha) > 0$ **then**
        Append $D_h \leftarrow D_h \bigcup \{D_\alpha\}$
        $\alpha \leftarrow \emptyset$
    else
      Execute $a_t^* = \pi^*(s_t, c_t)$
      for $j = 1, \ldots, M$ **do**
        $s_{t-M}, a_{t-M} \leftarrow D_\beta[M]$
        $D_\beta[j][a] \leftarrow$ Backtrack(a_t^*, a_{t-M}, j)
      Append $D_h \leftarrow D_h \bigcup \{D_\beta\}$
      Append (s_t, a_t) to D_α
      $D_\beta \leftarrow \emptyset$
until End of collecting
Update policies $\pi_{k+1} \leftarrow$Train(π_k, D_h)

Experimental Setup

Simulated Environment

We use a 3D urban driving simulator CARLA (Dosovitskiy et al. 2017). It offers a realistic emulation of vehicle dynamics and information, including distance traveled, collisions, and the occurrences of infractions (e.g., drifting into the opposite lane or the sidewalk). By employing the agent in CARLA simulator, we are able to automate the training and evaluation process.

Data Collection

The agent is initialized at a random location with 80 other vehicles in town at each iteration. A local-planner was used to generate roaming routes containing waypoints using topological commands, c_t (e.g., `TURN_LEFT`), provided by a human expert.

The expert policy is presented by a mixture of a Proportional-Integral-Derivative (PID) controller and a human. PID controller uses the route provided by the planner to compute the steer-angle. A human expert is presented with a third-person view of the environment, who takes control over the PID controller when the simulated vehicle stuck into unrecoverable states. The expert intervenes only in either of the following situations, potential crash or potential lane evasion. For now, we ignore traffic lights and stop signs in order to keep the same setting with (Codevilla et al. 2018), but these can be easily added to our framework when needed.

As discussed earlier, the recorded control signal contains the steering angle $a_{t,s}$ and the brake $a_{t,b}$. The steering angle is between $\mathcal{A}_s = [-1, 1]$, with extreme values corresponding to full left and full right, respectively. The brake is a binary signal $\mathcal{A}_b = \{0, 1\}$, where 1 indicates full brake. The

speed of the car is controlled by a separate PID controller at 50km/h. We don't record the acceleration signal because it depends on the instantaneous velocity of the car, which is subject to various factors and hard to model.

We also recorded topological commands c_t that are provided by either human or planner. We split all topological commands into three categories, $\mathcal{C} = \{0, 1, 2\}$: `lane-following` ($c_t = 0$), turning `left` ($c_t = 1$) or `right` ($c_t = 2$) at the intersection.

Evaluation

Implementation details The observation from CARLA at each time-step is an 800×600 resolution image. For the baseline **Branched** model, the observation is first encoded into a 1024-dimension feature vector $\psi(s_t)$ using ResNet-50 (He et al. 2016). It consists of 5 stages of identity and convolution blocks, both of which has 3 convolution layers, respectively.

Three separate two-headed Multi-Layer Perceptrons (MLP) follow the Resnet-50 encoder. Each of them has 3 fully-connected (FC) layers followed by Exponential Linear Unit (ELU) non-linearity. This is used as the bottom-level policy $\pi(s_t, g_{t+k})$.

Our top-level policy, $\hat{\phi}(s_t, c_t)$, again contains three separate MLPs that use command c_t as switch. All other modules are implemented as standard MLPs. We use ELU non-linearities after all hidden layers and applied 50% dropout after fully-connected hidden layers We initiate ResNet-50 with pre-trained parameters and only fine-tune the top three stages. The Equation 8 is minimized with a learning rate of 1e-5 using Adam solver.

For each experiment, we use behavior cloning with 30 mins recorded data ($\sim$7200 frames) in our first iteration and test agent for 15 mins in each subsequent iteration. We vary two properties of the algorithms for evaluation: data collection and policy representation. We have three variations for the data-collection approach:

- **Demo** stands for Behavior Cloning (Argall et al. 2009) using intervention data as well as non-intervention data as additional demonstration data.
- **CoL** stands for Cycle-of-Learning (Goecks et al. 2019), which uses only intervention data as additional demonstration data and ignores the non-intervention data.
- **LbB** stands for our approach, Learning from intervention by Backtracking, as described earlier in the Methods section. We ignore most of the non-intervention data, except M steps before and after the intervention.

For policy representation, we use two variations:

- **Branched** denotes the best setting in (Codevilla et al. 2018), which uses a feature extractor followed by three parallel Multi-Layer Perceptrons (MLPs). The condition of the expert's intention acts as a switch that selects which MLP is used. This model does not use hierarchy.
- **Sub-goal** denotes our proposed hierarchical network structure with $k = 5$ as described in the Methods section.

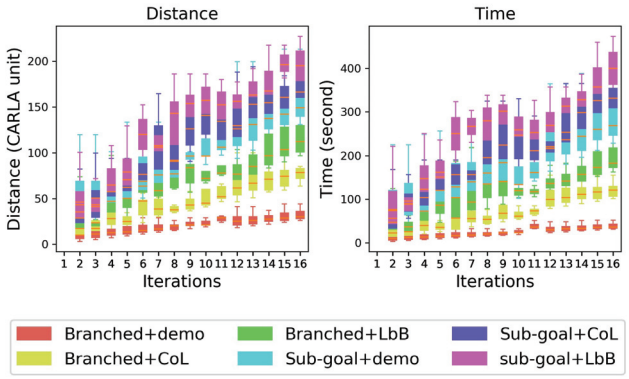

Figure 6: Graph showing the progress of the agent as it grows to be able to travel for longer distances and times, respectively, before needing expert intervention. For both graphs, higher is better. We compare the effects of our contributions (**Sub-goal** and **LbB**) with baseline approaches. The combination of our two contributions **Sub-goal+LbB** produces the best results.

Experiments and Results

We perform three sets of experiments. The first two experiments evaluate the accuracy and data-usage of different algorithms over training iterations. The effect of the hyper-parameter k (see Eq (4)) on our proposed algorithm is evaluated in the third experiment.

Accuracy We evaluated our method in terms of the two metrics: the **time** and **distance** without intervention. Time and distance without intervention measure is a standard measure of progress of autonomous driving. In the absence of an expert overseer, it can be assumed to be the same as the duration of the agent successfully driving itself in autonomous mode. The results across different algorithms are shown in Figure 6. Note that Behaviour Cloning, **Branched+demo**, has a moderate performance increase in the first few iterations due to the increase of dataset, but fails to keep increasing as the dataset increments. The results also show a significant increase in performance due to both of our contributions **LbB** and **Sub-goal**.

Data efficiency In this experiment, we explore how much data each algorithm consumes as a function of intervention iterations. In Figure 7, we show that as the learning policy improves, the algorithm needs fewer interventions, hence less data. As **demo** uses all the data, including the data with interventions and without interventions, the number of data-samples is highest and stays constant per iteration. **CoL** and **LbB** use only intervention data, so the data-samples used depend upon the accuracy of the algorithm. Again due to faster-learning, our proposed method **sub-goal+LbB** uses the least amount of data. Our results also confirm results from (Goecks et al. 2019) that Learning from Intervention (LfI) is data-efficient, as it uses only the intervention data

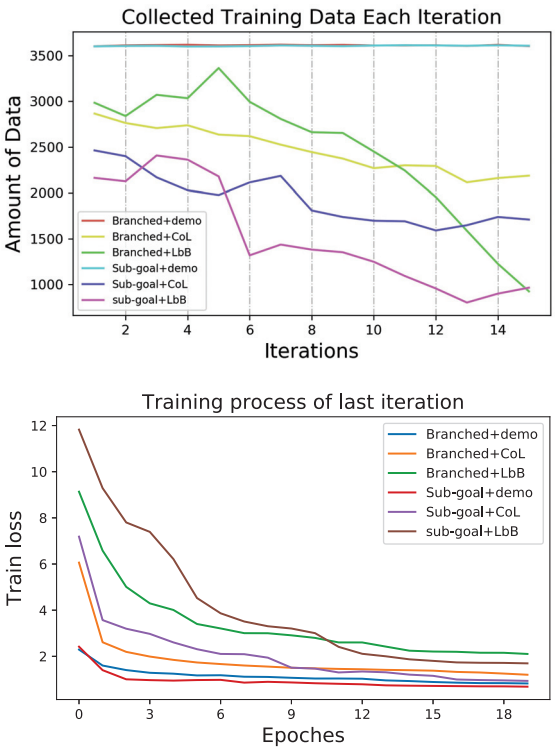

Figure 7: **Top:** Comparison of the number of data-samples per iteration needed to train the various algorithms. The number of data-samples used is proportional to the number of expert interventions needed per iteration. Lower is better, indicating that the algorithm is learning from less training data. Our algorithm **Sub-goal+LbB** shows a sharp drop in the amount of data needed per iteration, followed by consistently low values. **Bottom:** Loss curves during training for different algorithms. Each algorithm uses different amount of data (proportional to number of interventions) during training. For fairer comparison refer to Figure 6. The plot is included for completeness.

rather than all data. As the demonstrations continue, it becomes more likely to encounter seen sates and the states where the algorithm already performs well. LfI allows the algorithm to focus on critical intervention-related states and learn corrective actions.

To confirm the idea that the number of data samples is proportional to the number interventions, we calculate the mean and variance of the number of frames per intervention for six algorithms:**branched+demo**, **branched+CoL**, **branched+LbB**, **sub-goal+demo**, **sub-goal+CoL**, **sub-goal+LbB**. We find the mean = $[22, 17, 23, 20, 23, 25]$ (4 frame/second) and the variance = $[4, 6, 5, 6, 3, 4]$ in that order. The low variance shows that the total number of intervention is roughly proportional to the data added, and will have a similar decay as shown in Figure 7 for the number of interventions.

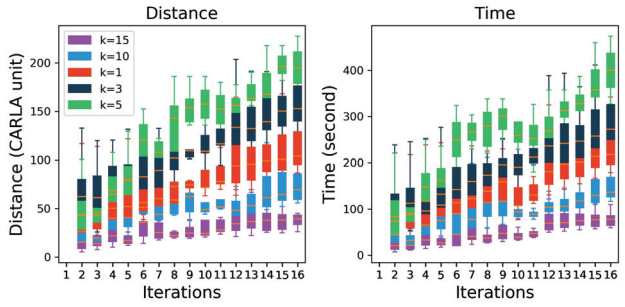

Figure 8: Evaluation of the effect of k on our proposed **Sub-goal+LbB** algorithm. The graph shows the time and distance traveled by the agent without intervention. Higher is better. Matching our expectations, the results show that k represents a trade-off between predicting the sub-goals far into the future (high k) by the top-level policy with the ability of the bottom-level policy to follow sub-goals correctly (low k). $k = 5$ provided the best performance, so we continued to use that value for the rest of our experiments.

Training loss We show the training curves for a single intervention batch in Figure 7. We make two observations. First, **Sub-goal** uses less data than **Branched**, which leads to a slower convergence but eventually leads to better accuracy. Secondly, **LbB** results in bigger initial loss than other data-collection methods (**CoL** or **Demo**). We think this is because **LbB** is able to collect more relevant data than **Demo** but yet produces more data-samples than **CoL**.

Please note that these loss curves do not reflect the performance of the algorithms. This is because of two reasons. First, these loss curves are for the last iteration of the many training cycles performed during full online training. Second, different models have different data sizes (based on the number of interventions) after the initial iteration, so the comparison might seem misleading.

Effect of the hyper-parameter k We also investigate the effect of the hyper-parameter k in Eq (4), which represents the trade-off between the long-term sub-goal prediction and robustness to the expert's reaction time. Our dataset is captured at 4 frames per second, hence $k = 4$ is equivalent to 1 second. The results are shown in Figure 8.

We evaluate the effect of the hyper-parameter k on our proposed algorithm **Sub-goal+LbB**. As we discussed above, the parameter k indicates how far in the future the top-level policy is learning to predict the sub-goal g_{t+k}. We choose $k \in \{1, 3, 5, 10, 15\}$ which is 0.25s to 3.75 second in the future. The results match our intuition that k represents a trade-off between predicting sub-goal far into the future by top-level policy while the ability of bottom-level policy to follow it correctly. With a small value of k, like $k = 1$, the policy learned is reactive and similar to the baseline `Branched`. Increasing k up to a point improves the performance, but after a point, the bottom-level policy is unable to learn to follow the far-off goals correctly. We find $k = 5$ as the sweet spot and use it for all other experiments.

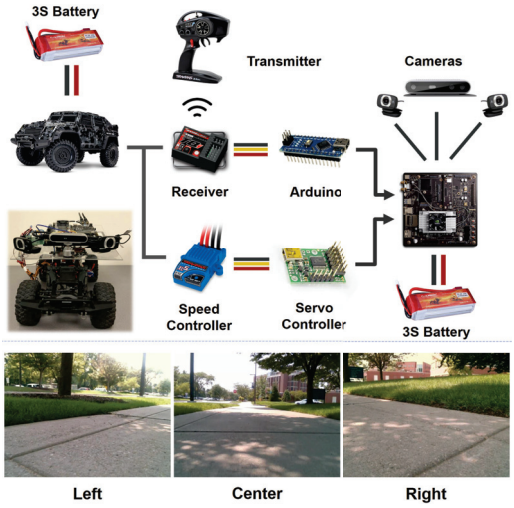

Figure 9: The architecture of our hardware system setup indicating how wires connect and frames captured by three cameras in one spot. This three-camera setting provides a wide field of view to ensure the sides of paths can be recorded.

Real-World Demonstration

To demonstrate the practicality of our approach, we deploy **Branched+LbB** on a real RC car with a few differences. Instead of a car rich environment, we study navigation in a pedestrian-rich environment. Also, we have different command categories (or scenarios): {`path-following` (with no pedestrian), `pedestrian-following`, `confronting` (avoid hitting a confronting person), `crossing` (avoid hitting a crossing pedestrian)}. We equipped an off-the-shelf 1/10 scale ($13'' \times 10'' \times 11''$) truck with an embedded computer (Nvidia TX2), an Intel RealSense D415 as the primary central camera and two webcams on the sides. The setup of the physical system is shown in Figure 9. We also employ an Arduino bboard and a USB servo controller as our control system to the robot. The decrease in the amount of data required, corresponding to the number of interventions needed, is shown in Figure 10. We include the demo video in the supplementary material.

Conclusions and Future Work

To conclude, we introduced a new problem formulation that accounts for the expert's reaction delay in Learning From Interventions (LfI). We proposed a new method to solve the proposed variation of LfI, which combines LfI with Hierarchical RL. We implemented the method on an autonomous driving problem in CARLA simulator using a Triplet-Network based architecture. We also proposed an interpolation trick called Backtracking, that allows us to use state-action pairs before and after the intervention. Our experiments show the value of both innovations. Our experiments also confirm the superiority of LfI approaches over Learning from Demonstrations in terms of data efficiency.

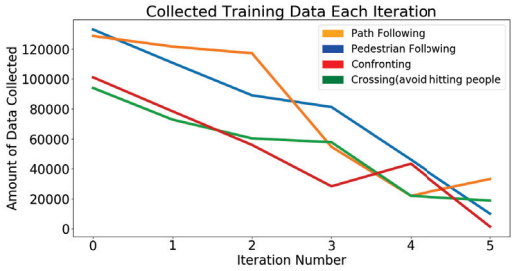

Figure 10: The amount of data collected (number of frames) for the real RC car experiment. Changes in size of data-samples per iteration, proportional to the number of interventions, used in four scenarios. Lower is better.

Additionally, experimenting with the values of k provides insight into how we can help the agent understand the environment, by allowing it to consider the temporal relation of subsequent states.

While the presented results are encouraging, there also remains significant room for progress. Learning from Interventions can easily be extended to IRL, which will allow us to improve data-efficiency further while ensuring safety. Learning from Intervention is timely, because there already exist Level-3 autonomy cars on the roads, with abundantly available intervention data that can be used with our approach to training the algorithms further. Finally, our algorithm has the limitation of resorting to a heuristic-based approximation of the effective policy. In future work, we will devise a principled way of coming up with that approximation.

Acknowledgments

J. Bi, T. Xiao, and C. Xu are supported by NSF IIS 1741472, IIS 1813709, and NIST 60NANB17D191 (Sub). V. Dhiman is supported by the Army Research Laboratory - Distributed and Collaborative Intelligent Systems and Technology Collaborative Research Alliance (DCIST-CRA). This article solely reflects the opinions and conclusions of its authors but not the funding agents.

References

Abbeel, P., and Ng, A. Y. 2004. Apprenticeship learning via inverse reinforcement learning. In *ICML 2004*, 1. ACM Press.

Agrawal, P.; Nair, A. V.; Abbeel, P.; Malik, J.; and Levine, S. 2016. Learning to poke by poking: Experiential learning of intuitive physics. In *NeurIPS*, 5074–5082.

Agrawal, P.; Carreira, J.; and Malik, J. 2015. Learning to see by moving. In *ICCV*, 37–45.

Akgun, B.; Cakmak, M.; Yoo, J. W.; and Thomaz, A. L. 2012a. Trajectories and keyframes for kinesthetic teaching: A human-robot interaction perspective. In *HRI*, 391–398.

Akgun, B.; Cakmak, M.; Jiang, K.; and Thomaz, A. L. 2012b. Keyframe-based learning from demonstration: Method and evaluation. *IJSR* 4(4):343–355.

Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Abbeel, O. P.; and Zaremba, W. 2017. Hindsight experience replay. In *NIPS*, 5048–5058.

Argall, B.; Chernova, S.; Veloso, M.; and Browning, B. 2009. A survey of robot learning from demonstration. *Robotics and Autonomous Systems* 57(5):469–483.

Bi, J.; Xiao, T.; Sun, Q.; and Xu, C. 2018. Navigation by imitation in a pedestrian-rich environment.

Codevilla, F.; Miiller, M.; Lopez, A.; Koltun, V.; and Dosovitskiy, A. 2018. End-to-end driving via conditional imitation learning. In *2018 ICRA*, 1–9. IEEE.

Dosovitskiy, A., and Koltun, V. 2016. Learning to act by predicting the future.

Dosovitskiy, A.; Ros, G.; Codevilla, F.; López, A.; and Koltun, V. 2017. CARLA: an open urban driving simulator. *CoRR* abs/1711.03938.

Giusti, A.; Guzzi, J.; Cireşan, D. C.; He, F.; Rodríguez, J. P.; Fontana, F.; Faessler, M.; Forster, C.; Schmidhuber, J.; Caro, G. D.; Scaramuzza, D.; and Gambardella, L. M. 2016. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters* 1(2):661–667.

Goecks, V. G.; Waytowich, N. R.; Gremillion, G. M.; Valasek, J.; and Lawhern, V. J. 2019. Efficiently combining human demonstrations and interventions for safe training of autonomous systems in real-time. *AAAI Conference on Artificial Intelligence (2019). Frames/sec vs Params No GPU* 9.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *CVPR*, 770–778.

Hilleli, B., and El-Yaniv, R. 2018. Toward deep reinforcement learning without a simulator: An autonomous steering example. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Ho, J., and Ermon, S. 2016. Generative adversarial imitation learning. In Lee, D. D.; Sugiyama, M.; Luxburg, U. V.; Guyon, I.; and Garnett, R., eds., *NeurIPS 29*. Curran Associates, Inc. 4565–4573.

Hoffer, E., and Ailon, N. 2015. Deep metric learning using triplet network. In *International Workshop on Similarity-Based Pattern Recognition*, 84–92. Springer.

Knox, W. B., and Stone, P. 2009. Interactively shaping agents via human reinforcement: The TAMER framework. In *Proceedings of the Fifth International Conference on Knowledge Capture*, K-CAP '09, 9–16. ACM. event-place: Redondo Beach, California, USA.

Kulkarni, T. D.; Narasimhan, K.; Saeedi, A.; and Tenenbaum, J. 2016. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *NeurIPS*, 3675–3683.

Le, H. M.; Jiang, N.; Agarwal, A.; Dudík, M.; Yue, Y.; and Daumé III, H. 2018. Hierarchical imitation and reinforcement learning. *arXiv preprint arXiv:1803.00590*.

MacGlashan, J.; Ho, M. K.; Loftin, R.; Peng, B.; Wang, G.; Roberts, D. L.; Taylor, M. E.; and Littman, M. L. 2017. Interactive learning from policy-dependent human feedback. *Proceedings of the 34th ICML-Volume 70* 10.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Pathak, D.; Agrawal, P.; Efros, A. A.; and Darrell, T. 2017. Curiosity-driven exploration by self-supervised prediction. In *2017 IEEE Conference on CVPR Workshops (CVPRW)*, 488–489. IEEE.

Peng, X. B.; Abbeel, P.; Levine, S.; and van de Panne, M. 2018. DeepMimic: Example-guided deep reinforcement learning of physics-based character skills. 37(4):1–14.

Ross, S.; Melik-Barkhudarov, N.; Shankar, K. S.; Wendel, A.; Dey, D.; Bagnell, J. A.; and Hebert, M. 2013. Learning monocular reactive UAV control in cluttered natural environments. In *ICRA*, 1765–1772.

Saunders, W.; Sastry, G.; Stuhlmüller, A.; and Evans, O. 2018. Trial without error: Towards safe reinforcement learning via human intervention. 3.

Savinov, N.; Dosovitskiy, A.; and Koltun, V. 2018. Semi-parametric topological memory for navigation. In *ICLR 2018*.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Tai, L.; Zhang, J.; Liu, M.; Boedecker, J.; and Burgard, W. 2016. A survey of deep network solutions for learning control in robotics: From reinforcement to imitation. *arXiv preprint arXiv:1612.07139*.

Waytowich, N. R.; Goecks, V. G.; and Lawhern, V. J. 2018. Cycle-of-learning for autonomous systems from human interaction.

Yu, T.; Abbeel, P.; Levine, S.; and Finn, C. 2018. One-shot hierarchical imitation learning of compound visuomotor tasks. *arXiv preprint arXiv:1810.11043*.

Zhang, J., and Cho, K. 2017. Query-efficient imitation learning for end-to-end simulated driving. In *31st AAAI 2017*, 2891–2897. AAAI press.