A Framework for Learning from Demonstration with Minimal Human Effort

Marc Rigter[1], Bruno Lacerda[1], and Nick Hawes[1]

***Abstract*—We consider robot learning in the context of shared autonomy, where control of the system can switch between a human teleoperator and autonomous control. In this setting we address reinforcement learning, and learning from demonstration, where there is a cost associated with human time. This cost represents the human time required to teleoperate the robot, or recover the robot from failures. For each episode, the agent must choose between requesting human teleoperation, or using one of its autonomous controllers. In our approach, we learn to predict the success probability for each controller, given the initial state of an episode. This is used in a contextual multi-armed bandit algorithm to choose the controller for the episode. A controller is learnt online from demonstrations and reinforcement learning so that autonomous performance improves, and the system becomes less reliant on the teleoperator with more experience. We show that our approach to controller selection reduces the human cost to perform two simulated tasks and a single real-world task.**

***Index Terms*—Learning from demonstration, human-centered robotics, telerobotics and teleoperation**

I. Introduction

INTEGRATING demonstrations with Reinforcement Learning (RL) has been applied to a number of difficult problems in sequential decision making and control. Examples include Atari games [1] and manipulation tasks [2] [3]. In most settings, it is assumed that there is a fixed set of demonstration data. In this work, we are interested in *shared autonomy*, where control may switch back and forth between a human teleoperator, and autonomous control. This is common in many domains in which fully autonomous systems are not yet viable, and human intervention is required to increase system capability, or recover from failures (e.g. [4] [5] [6]). In shared autonomy it is desirable to reduce the burden on the human operator by only handing control to the human in situations where autonomous performance is poor, and only handing control to the robot when autonomous performance is good. The autonomous experience and human demonstrations may be used to improve autonomous performance so that the system becomes less reliant on the human for future tasks.

Elements of this problem have been addressed before. Optimally switching control in shared autonomy has been approached using Markov Decision Process (MDP) planning [7] [8]. However, most planning approaches assume a known model for the performance of the human and autonomous system, which may not be available in reality, or may change over time. Requesting human input has been approached with the *ask for help* framework, where the agent asks for a human demonstration if uncertainty is above a hand-tuned threshold [9] [10] [11] [12]. Unlike the ask for help approaches, we seek to explicitly minimise a cost function associated with human exertion, and unlike the planning approaches we do not assume models are known a priori, and learn a control policy online.

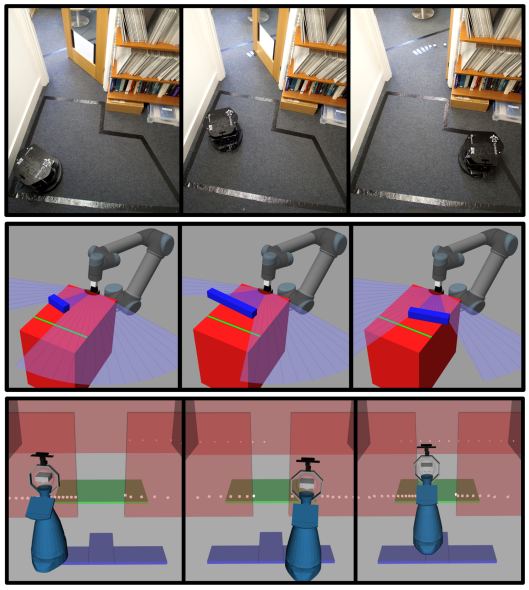


Fig. 1: Reprensentative initial configurations in our evaluation domains: real-world navigation (top), block pushing (middle) and simulated navigation (bottom).

This work considers episodic problems with a binary outcome of success or failure. We assume that there is a cost for asking the human for a demonstration, and a cost for failing an episode. These costs represent the time required for the human to perform a demonstration, or to recover the robot from failure. Motivating examples include robot navigation, where a human assists the robot when it is stuck [5], and manipulation, where a human recovers items dropped by the robot [13]. For each episode, the system must decide whether to ask for a human demonstration, or give control to one of the available autonomous controllers. We assume that an autonomous controller is learnt online as the system gathers more experience. We also allow for additional autonomous controllers, as in many domains there exist pre-programmed controllers which can be utilised to reduce operator workload.

Manuscript received: September, 10, 2019; Revised December, 18, 2019; Accepted January, 22, 2020.

This paper was recommended for publication by Editor Dongheui Lee upon evaluation of the Associate Editor and Reviewers' comments.

[1]Marc Rigter, Bruno Lacerda, and Nick Hawes are with the Oxford Robotics Institute, University of Oxford, United Kingdom (email: {mrigter, bruno, nickh} @robots.ox.ac.uk)

Digital Object Identifier (DOI): see top of this page.

The contributions of this paper are a general framework for this problem, and a specific instantiation of the framework. In the framework, controller choice is formulated as a contextual Multi-Armed Bandit (MAB). In our instantiation, we use continuous correlated beta processes to estimate the MAB probabilities. A control policy is learnt online from demonstrations plus RL so that performance improves and the system becomes increasingly autonomous. We evaluate our approach on three domains (Figure 1), including experiments on a real robot, and show empirically that our approach reduces the total human cost relative to other approaches.

II. Related Work

Previous research has combined human input with RL to guide the learning process. In supervised actor-critic RL [14], the action applied at each time step is a weighted sum of the actions suggested by a human supervisor and by the learner. In interactive RL, the reward signal is given by a human [15].

Other approaches modify the control input of the human to improve performance whilst maintaining the control authority of the operator. Reddy et al. [16] apply deep RL to learn an approximate Q-function. The action applied to the system is the action near the teleoperator input with highest value. [17] uses Model Predictive Control (MPC) to determine the optimal control from a known model, with a cost for deviating from the user control inputs. [18] also uses MPC, but learns the model from data collected from human-machine interaction. All of the above approaches require that the human operator is present at all times, while we are interested in methods which reduce the input required from the operator.

The ask for help framework introduced in [9] only asks for human input when the learner is uncertain. In [9], the agent nominally performs tabular Q-learning. If all actions in a state have a similar Q-value, the agent is deemed uncertain, and asks the human which action to take. A similar idea is applied in the Deep Q-Network (DQN) setting in [12], where the loss of the DQN is used to gauge uncertainty. Chernova and Veloso [11] consider policies defined by a support-vector machine classifier. For states near a decision boundary of the classifier, a demonstration is requested. In [10], Gaussian process policies are learnt from demonstrations. A demonstration is requested if the variance of the action output is too high. In contrast to our work, all of these approaches require a hand-tuned threshold for uncertainty.

In *predictive advising* [19], the teacher learns to predict the action that the learner will take in a given state. In a new state, if the predicted action does not equal the "correct" action known by the teacher, the teacher chooses the action taken. This method would be difficult to apply to continuous action spaces where it is unclear whether an action is correct.

MDP planning has been applied to the problem of optimally switching control between a human and agent. In [20], the authors find Pareto-optimal MDP policies which trade off minimising a cost function for operator effort, and maximizing the probability of success of satisfying a temporal logic specification. Such a cost representing human inconvenience is often referred to as "bother" cost [21]. Wray et al. [7] consider that the human and autonomous system have different known capabilities, and generate plans such that the system never enters states where the entity in control cannot act. These approaches assume a model is known for the human and autonomous agent. Lacerda et al. [22] plan over an MDP with transition probabilities learnt from executing each transition many times [23]. This approach does not require prior knowledge, but is limited to long-term deployment in the same environment. The authors use human intervention to recover from failures [5], but do not plan for human control.

Probabilistic policy reuse [24] assumes access to a library of baseline policies which may be reused, overriding the actions of the current policy with some probability. The algorithm maintains an average of the return from reusing each policy. Higher returns increase the likelihood a policy is chosen. This approach determines which baseline policy is most useful for learning the task. In our work, we consider variable controller performance throughout the state space, and choose the controller depending on the initial state.

A comparison can be made between our work and hierarchical RL [25] which uses *options*. An option is a temporally extended action, similar to the choice of controller in our work. In hierarchical RL, the policy which chooses options is also learnt with RL algorithms. In contrast, we consider controller choice as a contextual MAB and use uncertainty estimates to guide exploration towards promising choices.

In contrast to previous work, we both explicitly minimise a cost function associated with the human workload, and do not assume prior knowledge of models of performance.

III. Preliminaries

A. Multi-Armed Bandits

In a Multi-Armed Bandit (MAB), at each episode an agent must choose from a finite set Φ of arms, with unknown reward distributions r_ϕ for choosing each arm $\phi \in \Phi$. The objective is to maximise the cumulative reward received. Algorithms which solve the MAB problem must balance exploration to gather information about the arms with exploiting the best known arm. In this work, we consider an extension of the MAB that is *non-stationary* and *contextual.*

In a contextual MAB [26], at episode k the agent also receives information about the "context" in the form of a state s_k, which may inform the agent about the reward distribution of each arm for episode k. Over many trials, algorithms for the contextual MAB estimate the mean and variance of the reward for arm $\phi \in \Phi$, given state s_k where $\mu(\phi, s_k) = \mathbb{E}[r_\phi | s_k]$, and $\sigma(\phi, s_k)^2 = var[r_\phi | s_k]$. One effective approach to arm selection is to use an Upper Confidence Bound (UCB) algorithm [26]:

$$\phi_k = \arg\max_{\phi \in \Phi} \Big(\mu(\phi, s_k) + \alpha \sigma(\phi, s_k) \Big), \tag{1}$$

where α trades off exploration versus exploitation.

Our contextual MAB may also be non-stationary with the reward distribution changing between episodes (eg. due to learning). A simple approach to address this is to use a *sliding window* and only use the m most recent trials to estimate the distributions in Equation 1 [27].

B. Continuous Correlated Beta Processes

We wish to choose a function approximator to predict the probabilities required in the MAB. To estimate probabilities in the range [0,1], the output of a Gaussian Process (GP) can be passed through a logistic function to form a Logistic GP (LGP). However, for LGPs, the posterior can only be computed approximately, and computation is cubic with the size of the dataset. A simpler alternative is the Continuous Correlated Beta Process (CCBP), proposed in [28]. Like LGPs, CCBPs are a nonparametric function approximator with a model of uncertainty, and a range of [0,1], but have computation time linear with the number of data points.

Let S be a continuous state space and $\mathcal{B} = \{B_s \mid s \in S\}$ the space of Bernoulli trials indexed by states of S, with unknown success probability $p(s) = \Pr(B_s = 1)$. The outcome, $o \in \{0, 1\}$ of each trial may either be successful, denoted 1, or unsuccessful, denoted 0. The distribution over $p(s)$ after $\alpha(s)$ outcomes of success and $\beta(s)$ outcomes of failure is given by a beta distribution:

$$\Pr(p(s)) = Beta(\alpha(s), \beta(s)) \propto p(s)^{\alpha(s)-1}(1 - p(s))^{\beta(s)-1} \tag{2}$$

To obtain accurate values for $p(s)$ over the entire state space, we would need to observe many outcomes at each s, which is not possible in continuous space. In a CCBP we assume $p(s)$ is a smooth function, such that the Bernoulli trials are correlated, and experience can be shared between them. A kernel function, $K(s, s') \in [0, 1]$ is used to determine the extent to which experience from trial B_s should be shared with $B_{s'}$. Consider the set $O = \{B_{s_0} = o_0, B_{s_1} = o_1, \ldots, B_{s_T} = o_T\}$ of observed outcomes $o_0, \ldots, o_T$ after running $T + 1$ different trials. The posterior beta distribution for an experiment B_s is given by:

$$\begin{aligned}\Pr(p(s)|O) \propto\ & p(s)^{\alpha(s)-1+\sum_{t=0}^{T} o_t K(s_t,s)} \\ & \times (1 - p(s))^{\beta(s)-1+\sum_{t=0}^{T}(1-o_t)K(s_t,s)}.\end{aligned} \tag{3}$$

C. Deep Deterministic Policy Gradients

In an RL problem, an agent must learn to act by receiving a reward signal corresponding to performing an action $a \in A$ at a state $s \in S$. At discrete time step t, the agent takes action a_t according to a policy, $\pi : S \to A$. We define the return $G_t = \sum_{i=t}^{T} \gamma^{(i-t)} r_i$ where T is the horizon, $\gamma < 1$ is the discount factor, and r_i is the reward received at time step i. We consider episodic problems in which T is the end of the episode. The goal of RL is to find π which maximises the expected return, $J = \mathbb{E}_\pi[G_t]$.

Deep Deterministic Policy Gradients (DDPG) [29] is an off-policy, model-free, RL algorithm capable of utilizing neural network function approximators. DDPG maintains a state-action value function, or critic, $Q(s, a)$, with parameters θ^Q, and a policy, or actor, $\pi(s)$, with parameters θ^π. Additionally, a replay buffer R of recent transitions experienced is maintained containing tuples of (s_t, a_t, r_t, s_{t+1}).

The algorithm alternates between collecting experience by acting in the environment, and updating the actor and critic. For exploration, noise is added to the actions chosen by the actor during training: $a_t = \pi(s_t) + \mathcal{N}$, where $\mathcal{N}$ is a noise process. During each update step, a minibatch of N samples is taken from R to update the actor and critic functions. The critic parameters, θ_Q, are updated to minimise the loss

$$L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta_Q))^2, \tag{4}$$

where the targets y_i are computed from the Bellman equation:

$$y_i = r_i + \gamma Q(s_{i+1}, \pi(s_{i+1}|\theta^\pi)|\theta^Q). \tag{5}$$

The actor is updated using the deterministic policy gradient:

$$\nabla_{\theta^\pi} J = \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\pi(s_i)} \nabla_{\theta^\pi} \pi(s|\theta^\pi)|_{s=s_i} \tag{6}$$

Intuitively, this update changes the actor parameters to produce actions with a higher Q-value as judged by the critic. The target values in Equation 5 are usually computed using separate networks for the actor and critic whose weights, $\theta^{\pi'}$ and $\theta^{Q'}$, are an exponential average over time of θ^π and θ^Q respectively. This is necessary to stabilise learning.

IV. Methodology

We assume that our system must perform an episodic task. Each episode k has an initial state, $s_{k,0} \in S_0$, and a binary outcome, $o_k \in \{0, 1\}$, of success for reaching a goal state $s_g \in S_g$, or failure. For each episode a controller must be chosen. The controller may be human teleoperation, which we denote C_h, or one of the n available autonomous controllers, $C_{a,i}$. The autonomous controllers may be pre-programmed, or learnt online. We assume that there is a cost, c_d, per human demonstration representing the human time to give a demonstration, and a cost c_f for failure, representing the human time required to recover the robot from failure. For the k^{th} episode, the system chooses a controller, $C_k \in \mathcal{C}$, where $\mathcal{C} = \{C_h, C_{a,i}, \ldots, C_{a,n}\}$. The objective is to minimise the cumulative cost $\sum c_h$, defined as the sum of the demonstration and failure costs over the episodes.

In the next subsection, we describe our general approach for controller selection to minimise the cumulative cost. Controller selection is considered as a contextual MAB, and an upper-confidence bound algorithm is applied to choose the controller at each episode. We then describe how the CCBP can be used for the performance prediction aspect of our framework, and how a controller can be learnt using DDPG with demonstrations so that the autonomous performance improves, making the system less reliant on the human. An open-source implementation of our framework is available[1].

A. A General Approach for Controller Selection

Our approach to controller selection is illustrated in Figure 2. We assume the existence of a performance prediction function, that for each controller $C_i \in \mathcal{C}$, and an initial state $s_{k,0}$ returns the probability of success for that controller for the episode, $\hat{p}(s_{k,0}, C_i)$, and the standard deviation of that probability

[1] github.com/ori-goals/lfd-min-human-effort

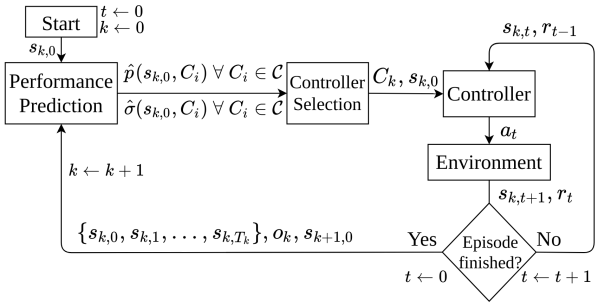


Fig. 2: High-level diagram of framework.

estimate, $\hat{\sigma}(s_{k,0}, C_i)$. One possibility for this function, based on CCBPs, will be described in Section IV-B.

These estimates are then used to perform controller selection. The controller selection problem is considered as a contextual MAB, in which we seek to minimise the total human cost. To choose a controller for episode k, we select

$$C_k = \arg\min_{C_i \in \mathcal{C}} \hat{c}_h(s_{k,0}, C_i), \tag{7}$$

where

$$\hat{c}_h(s_{k,0}, C_i) = \begin{cases} (1 - \hat{p}(s_{k,0}, C_i) - \alpha\hat{\sigma}(s_{k,0}, C_i))c_f + c_d & \text{if } C_i = C_h \\ (1 - \hat{p}(s_{k,0}, C_i) - \alpha\hat{\sigma}(s_{k,0}, C_i))c_f & \text{otherwise.} \end{cases} \tag{8}$$

Intuitively, $\hat{c}_h(s_{k,0}, C_i)$ is an optimistic lower bound on the cost for using controller C_i. This is analogous to UCB algorithms [26], but here we minimise cost, rather than maximise reward, and include the demonstration cost.

The chosen controller executes actions in the environment, until the episode ends. During each step, the environment produces a reward, r_t, which may be used to train the controllers. At the end of the episode, the states encountered during the episode, and the outcome of success or failure, o_k, are fed back to improve the performance prediction function. The environment chooses a new starting state, $s_{k+1,0}$, for the next episode. This framework is agnostic to the method for performance prediction, or the controllers available. In the following subsections, we describe the use of the CCBP for performance prediction, and DDPG with demonstrations as an approach to learning one of the controllers.

B. Performance Prediction with the CCBP

Under the assumption that the probability of succeeding at an episode for a given controller varies smoothly throughout the state space, we use a CCBP to estimate this probability, and its uncertainty. During episode k which is under the control of C_k, we observe a series of states from the continuous state space $\{s_{k,0}, s_{k,1}, \ldots, s_{k,T_k}\}$. At the end of the episode we observe outcome $o_k^{C_k}$, where we introduce the superscript to indicate the controller for that episode. We consider the outcomes of episodes as outcomes of correlated Bernoulli trials. In episode k, we observe the same outcome for each state, $O_k = \{B_{s_{k,0}} = o_k^{C_k}, B_{s_{k,1}} = o_k^{C_k}, \ldots, B_{s_{k,T_k}} = o_k^{C_k}\}$. After $n+1$ episodes, the entire set of observed outcomes is defined as $O = \bigcup_{k=0}^{n} O_k$. For a given controller C_i, we define the set of outcomes associated with that controller as $O^{C_i} = \{B_{s_{k,j}} = o_k^{C_k} \in O \mid C_k = C_i\}$

Given a previously unvisited state, s, we can calculate a beta distribution over $\Pr(B_s = 1 \mid O^{C_i})$ using a CCBP as defined in Equation 3, for each $C_i \in \mathcal{C}$. In the CCBP, we initialise $\alpha(s) = \alpha_0^{C_i}$, and $\beta(s) = \beta_0^{C_i}$. This encodes a prior assumption about the probability of success of C_i in the absence of information from correlated outcomes. The contributions from the outcomes in O^{C_i} are then incorporated per Equation 3 to produce the new beta distribution. For the success probability estimate, we take the expected value of this beta distribution: $\hat{p}(s, C_i) = \mathbb{E}[\Pr(B_s = 1 \mid O^{C_i})]$. The standard deviation of the probability estimate is also calculated from the beta distribution: $\hat{\sigma}(s, C_i)^2 = var[\Pr(B_s = 1 \mid O^{C_i})]$.

For the kernel function we chose the Gaussian kernel

$$K(s, s') = e^{-\frac{||s-s'||^2}{l}}, \tag{9}$$

where l is the length scale hyperparameter which determines the scale over which the correlation between outcomes decays with distance in the state space. The Gaussian kernel was chosen as it is a common choice for modelling smooth data.

In our framework, we also allow for a controller to be learnt online from demonstrations and RL. In this case, the performance of the controller is constantly changing, and so the performance prediction should not be influenced by out of date data. We follow the approach taken for GP value functions [30] and non-stationary MABs [27], and only consider recent data in the CCBP. This is based on the assumption that the policy changes gradually, such that recent outcomes approximate the performance of the current controller. Specifically, if C_l is a controller which is learnt online, then the outcomes we consider are those from the set $O^{C_l} = \{B_{s_{k,j}} = o_k^{C_k} \in O \mid C_k = C_l, \; k > n - m\}$, where n is the current episode number, and m is a constant. That is, O^{C_l} is the set of outcomes from episodes controlled by C_l in a sliding window of the most recent m episodes. If there are many recent episodes from other controllers, O^{C_l} is populated with fewer outcomes. This increases $\hat{\sigma}(s, C_l)$, indicating our reduced certainty in our probability estimates for C_l after many recent demonstrations.

To simplify our experiments we assume that the human teleoperation controller is always successful, that is: $\hat{p}(s, C_h) = 1$ and $\hat{\sigma}(s, C_h) = 0$ for all $s \in S$. However, this assumption is not necessary to apply our framework, and the human performance could also be predicted with a CCBP.

C. DDPG with Demonstrations

To provide an autonomous controller which is learnt online from demonstrations and RL, we adapt the approach from [3] to incorporate demonstrations into DDPG using behaviour cloning and a *Q-filter*. As DDPG is an off-policy algorithm, we add experience from all $C_i \in \mathcal{C}$ into the replay buffer R. Experience from successful episodes by controllers $C_i \neq C_l$ are also added into a demonstration replay buffer R_D.

An update is performed after each time step of any episode. During each update, we draw N experience tuples from R, and N_D tuples from R_D. The following Behaviour Cloning (BC) loss, L_{BC}, is applied only to the tuples from R_D:

$$L_{BC} = f \sum_{i=1}^{N_D} ||\pi(s_i|\theta^\pi) - a_i||^2, \tag{10}$$

$$f = \mathbb{1}_{Q(s_i,a_i) > Q(s_i,\pi(s_i)) - \epsilon|Q(s_i,\pi(s_i))|}, \tag{11}$$

where $\epsilon \geq 0$, a_i is the demonstrator action, and $\mathbb{1}_A$ is 1 if A is true and 0 otherwise. The Q-filter in Equation 11 results in the behaviour cloning loss not being applied when the critic determines that the actor action is significantly better than the demonstrator action. This prevents the actor being tied to the demonstrations if it discovers better actions. This Q-filter is modified from [3] by including the ϵ term, which when $\epsilon > 0$ reduces the experiences filtered out from the BC loss.

The gradient applied to θ^π is:

$$\lambda_1 \nabla_{\theta^\pi} J - \lambda_2 \nabla_{\theta^\pi} L_{BC}, \tag{12}$$

where λ_1 and λ_2 weight the policy gradient and behaviour cloning contributions to the update.

V. Experiments

We evaluated our approach in three domains. We assumed the availability of three controllers: $\mathcal{C} = \{C_h, C_b, C_l\}$. C_h was a human teleoperating the robot with a gamepad. C_b was a pre-programmed baseline controller capable of succeeding at some of the episodes. C_l was a control policy learnt online using the method outlined in Section IV-C. In all domains, the maximum length of an episode was 50 time steps, and the sliding window was $m = 50$ episodes. We set $\alpha_0^{C_i}$, $\beta_0^{C_i}$ to define a prior assumption that $\hat{p}(s_{k,0}, C_l) = \hat{p}(s_{k,0}, C_b) = 0.8$, and $\hat{\sigma}(s_{k,0}, C_l) = \hat{\sigma}(s_{k,0}, C_b) = 0.35$, in the absence of any observed outcomes ($\alpha_0^{C_i} = 0.245$, $\beta_0^{C_i} = 0.0612$). To break ties in Equation 7, we prioritised $C_b > C_l > C_h$.

A. Domains

1) Block Pushing: A 6-DOF arm was simulated in Gazebo [31] to push a block from a random initial configuration along a table into a goal region (see Figure 1). The state space is 32 dimensional, consisting of 30 depth measurements from a stationary laser, and 2 measurements specifying the current location of the end effector. The action space is 2 dimensional, specifying a horizontal change in position of the end effector. In each new episode, the arm position is reset, and a block is spawned onto a 0.3m×0.45m table. The block has a 0.04m×0.04m cross section and can be one of three lengths: 0.12, 0.2, or 0.3m. The initial position and orientation of the block is varied by up to ±10cm relative to the centre of the table, and ±20°. An episode is a failure if the block falls off the table, or the time limit is exceeded. An episode is a success if the entirety of the block is pushed past the green line (Figure 1). The baseline controller, C_b, was a partially trained policy using the method in Section IV-C and a sample of the recorded human demonstrations. Training of C_b was terminated when the policy was capable of ≈70% of the tasks. The reward r_t used to train the learnt policy consisted of a small dense reward for moving the block forwards, and a large reward for reaching the goal region.

2) Simulated Navigation: A Scitos G5[2] robot was simulated in the Morse simulator [32]. The task for the robot was to navigate from a starting location, through a gap, to a goal region. The starting location was randomly selected within a region. The start and goal regions are shown in Figure 1. The starting orientation was randomly varied ±10°. The width of the gap was varied according to a normal distribution with a mean of 0.83m (the standard wheelchair accessible door width in the United Kingdom), a standard deviation of 0.15m, and a lower bound of 0.68m. This lower bound is the minimum width at which the 0.62m robot can be reliably teleoperated through the gap. The position of the other walls was randomly varied by ±0.25m. The state space was a 40 dimensional laser-depth measurement, and the action space was 2 dimensional, specifying a distance to move and a change in orientation. The baseline controller, C_b, was a controller from the ROS navigation stack with the parameters fine-tuned for the Scitos G5 from the STRANDS project[3]. The reward r_t used to train the learnt policy consisted of a large reward for reaching the goal region, and 0 otherwise.

3) Real-World Navigation: The real-world navigation experiments used a Turtlebot 2[4]. The task was to navigate from a starting location and through a partially closed door. Possible starting locations are indicated by the region marked on the floor in Figure 1. To configure a new episode, the ROS navigation stack was used to navigate the robot to a random position within this region, and a random orientation varied up to ±15°. The door was set randomly to one of 8 possible positions indicated by markings on the floor. Three possible door positions are shown in Figure 1. The state space is 33 dimensional, comprising of 30 depth measurements given by the Kinect sensor on the Turtlebot, and 3 measurements specifying the estimated position and orientation of the robot given by a particle filter. The particle filter estimate is required because the Kinect has a narrow field of view, meaning that the depth measurements alone are insufficient to characterise the state. The action space was 2 dimensional, consisting of linear and angular velocity. The reward was a sparse reward for passing through the door, and 0 otherwise.

B. Length Scale Hyperparameter Estimation

The hyperparameter l was estimated from data in all domains. We initialised a policy π capable of completing approximately half of the tasks. The policy attempted 50 random tasks to populate O. The policy then executed another 50 episodes from new random initial states, which were not added to O. The likelihood of the outcomes of the second set of tasks was calculated using the CCBP for a range of values for l. The maximum likelihood estimate was then used for the CCBP kernel for both C_l and C_b. This value was $l = 4.1$ in the simulated navigation domain, $l = 2.1$ in the real-world navigation domain, and $l = 0.72$ in the block pushing domain. We leave estimating this parameter online to future work.

[2] metralabs.com/en/mobile-robot-scitos-g5/

[3] github.com/strands-project/strands_movebase

[4] clearpathrobotics.com/turtlebot-2-open-source-robot/

C. Training Details

To train C_l, we used Adam [33] with learning rate of 10^{-3} for Q, and 10^{-4} for π. The discount factor γ was 0.99. We used $N = 128$, $N_D = 64$, $\lambda_1 = 1$, $\lambda_2 = 10$. The value for λ_2 was chosen for best empirical performance after 100 demonstrations and 500 RL episodes out of $\{1, 10, 100\}$ in block pushing. The function approximators for π and Q are fully-connected neural networks with two hidden layers of 64 and 32 weights respectively, and ReLU activation. The output activation for π is tanh, and the value is rescaled into the action range. The exploration noise $\mathcal{N}$ is an Ornstein-Uhlenbeck process with $\sigma = 0.2$, $\theta = 0.15$. The noise was decayed by $0.998^{n_{C_l}}$, where n_{C_l} is the number of episodes completed by C_l. The Q-filter tolerance was $\epsilon = 0.02$ unless stated otherwise.

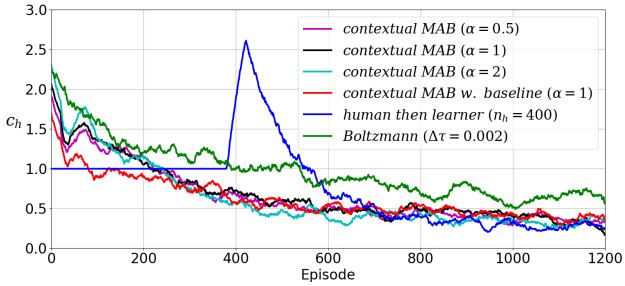
D. Simulated Human Cost Evaluation

We evaluated the cumulative human cost of several methods in simulation. The costs were $c_d = 1$ and $c_f = 5$ for both simulated domains. n random initial states were generated for each simulated domain. In box pushing, $n = 1200$ and in simulated navigation, $n = 500$. A human demonstration for each of the n possible initial states was recorded. Each run of the experiment consisted of randomly initialising the networks for C_l, and then performing a number of episodes (400 or 1200 in simulated navigation or box pushing respectively). In each run, the initial state for each episode was chosen in a random order from the set of possible initial states. In each run, the same recorded human demonstration was used for each initial state. We compared several methods, where α indicates the value used in the MAB: *contextual MAB (α)* is our approach with $\mathcal{C} = \{C_h, C_l\}$, *contextual MAB with baseline (α)* is our approach with $\mathcal{C} = \{C_h, C_b, C_l\}$, *baseline only* always uses C_b and *RL only* always uses C_l. The method *human then learner* (n_h), first executes n_h human demonstrations with C_h before switching to C_l. For *Boltzmann ($\Delta\tau$)*, $\mathcal{C} = \{C_h, C_b, C_l\}$, and we adapted the method in [24]. The controller is chosen according to:

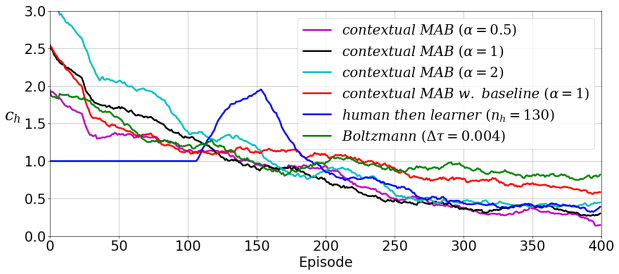
$$Pr(C_i) = \frac{e^{\tau(c_f - \bar{c}_h(C_i))}}{\sum_{C_j \in \mathcal{C}} e^{\tau(c_f - \bar{c}_h(C_j))}} \tag{13}$$

where $\bar{c}_h(C_i)$ is the average human cost incurred from the episodes in O^{C_i}. Temperature parameter τ is initially 0, and incremented by $\Delta\tau$ every episode.

Results: The average human cost over 15 runs of each method is plotted in Figure 3. For clarity, the cost is plotted using a sliding window average over 40 episodes. The average cumulative cost and episodes performed by each controller is displayed in Tables I and II. In block pushing, *contextual MAB* outperforms *human then learner* in total cost for all values of α and n_h. Smaller α values corresponded with asking for more demonstrations. The method *contextual MAB with baseline* further reduces the cost by using the baseline controller when the learnt policy is poor. Despite the fact that *baseline only* does not perform well, this is possible as *contextual MAB with baseline* tends to use the baseline for episodes it will likely succeed at. The *Boltzmann* method performs comparatively poorly for each of the values of $\Delta\tau$.



(a) Block pushing



(b) Simulated Navigation

Fig. 3: Mean cost from 15 runs of each method. For clarity, c_h is plotted using a sliding window average over 40 episodes.

TABLE I: Cumulative cost over 15 runs of each method in the box pushing domain in the format: mean (std dev.)

Method	Total Cost	Human Episodes	Baseline Episodes
context. MAB ($\alpha = 0.5$)	790 (117)	376 (76)	-
context. MAB ($\alpha = 1$)	820 (119)	332 (73)	-
context. MAB ($\alpha = 2$)	758 (167)	220 (67)	-
human then learner ($n_h = 200$)	1029 (90)	200	-
human then learner ($n_h = 300$)	959 (133)	300	-
human then learner ($n_h = 400$)	934 (94)	400	-
context. MAB w. baseline ($\alpha = 1$)	746 (75)	197 (32)	236 (35)
Boltzmann ($\Delta\tau = 0.001$)	1266 (74)	392 (15)	327 (25)
Boltzmann ($\Delta\tau = 0.002$)	1178 (94)	364 (30)	277(35)
Boltzmann ($\Delta\tau = 0.004$)	1204 (188)	500 (221)	244 (67)
baseline only	1613 (18)	-	1200
RL only	4947 (39)	-	-

TABLE II: Cumulative cost over in the simulated navigation domain in the format: mean (std dev.)

Method	Total Cost	Human Episodes	Baseline Episodes
context. MAB ($\alpha = 0.5$)	358 (51)	144 (33)	-
context. MAB ($\alpha = 1$)	366 (48)	110 (30)	-
context. MAB ($\alpha = 2$)	438 (53)	67 (16)	-
human then learner ($n_h = 70$)	517 (147)	70	-
human then learner ($n_h = 100$)	409 (75)	100	-
human then learner ($n_h = 130$)	349 (71)	130	-
context. MAB w. baseline ($\alpha = 1$)	433 (72)	57 (19)	118 (47)
Boltzmann ($\Delta\tau = 0.001$)	499 (46)	137 (9)	133 (11)
Boltzmann ($\Delta\tau = 0.002$)	505 (50)	155 (17)	147 (24)
Boltzmann ($\Delta\tau = 0.004$)	473 (56)	153 (27)	162 (26)
baseline only	443 (14)	-	400
RL only	1811 (46)	-	-

In the simulated navigation domain, *contextual MAB* performs better with a smaller value for α. This is likely because this simpler task can be learnt from demonstrations alone and no RL experience. Therefore, conservatively relying on more human demonstrations enables a good policy to be learnt

quickly and results in less cost. This also explains the strong performance of *human then learner* ($n_h = 130$). This suggests that our approach is better suited to more complex tasks which take longer to learn, and require RL experience to train a policy to complete the task. The method *contextual MAB with baseline* has less cost near the start of the runs, but performs worse overall. This is likely because the agent is slower to learn a good policy using the dissimilar demonstrations from the baseline controller and human in the simulated navigation domain. In the box pushing domain, the baseline and human give similar demonstrations because the baseline is a policy learnt from human demonstrations. Therefore this is not an issue in the box pushing domain.

E. Effect of Varying Costs

Some experiments were repeated in the block pushing domain to analyse the effect of the cost values. The demonstration cost, $c_d = 1$ for all experiments. We used $c_f \in \{3, 5, 7\}$ for *contextual MAB* and *human then learner* with a comparable number of demonstrations.

Results: Our approach incurs less cost than *human then learner* for all values of c_f (Table III). Increasing the value of c_f increases the number of human demonstrations used and vice versa. This is because if the cost for failure is higher the estimated probability of success must be higher for the algorithm to choose the autonomous controller.

TABLE III: Cost over 15 runs of 1200 episodes for block pushing with different c_f values in format: mean (std dev.)

Method	c_f	**Total Cost**	**Human Episodes**
context. MAB ($\alpha = 1$)	3	629 (82)	281 (52)
human then learner ($n_h = 300$)	3	715 (57)	300
context. MAB ($\alpha = 1$)	5	820 (119)	332 (73)
human then learner ($n_h = 300$)	5	959 (133)	300
context. MAB ($\alpha = 1$)	7	1067 (121)	421 (74)
human then learner ($n_h = 400$)	7	1154 (122)	400

F. Limited Demonstrations Evaluation

We evaluated the quality of the learnt policy when the number of human demonstrations is kept to a strict limit. After the demonstration budget was used up, C_l was used from then onwards. To evaluate policy performance, after each episode we performed an additional evaluation episode which always used C_l. The initial state for each evaluation episode was sampled randomly. The success rate of C_l in the evaluation episodes is plotted in Figure 4 for block pushing, where the limit on human demonstrations was 120. We additionally compared *human then learner* with $\epsilon = 0$ for the Q-filter.

Results: Early in training, *human then learner* outperforms *contextual MAB* as it receives all demonstrations immediately. However, *contextual MAB* converges to a policy with a high success rate more quickly. This indicates that *contextual MAB* received more informative demonstrations by asking for demonstrations at initial states where a good policy had not yet been learnt. *RL only* failed to learn a good policy in the limited number of episodes. Our results show that setting $\epsilon = 0$ for the Q-filter resulted in slower learning by filtering out most demonstration experience from the behaviour cloning loss. A promising approach may be to start with a large value for ϵ, and then gradually decay ϵ.

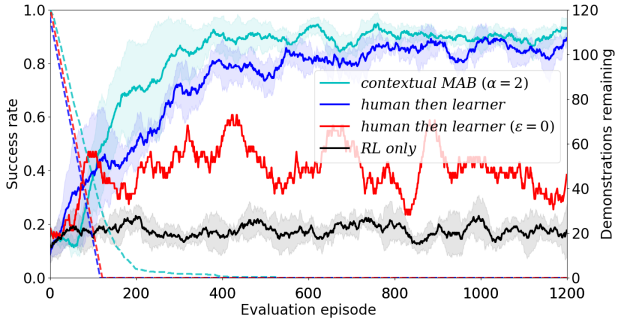


Fig. 4: Success rate on evaluation episodes for policies trained with a limited budget of 120 human demonstrations. Results are averaged over 5 runs for each method. Solid lines indicate average success rate over a sliding window of 40 episodes. Shaded regions illustrate the standard deviation over the 5 runs. Dashed lines indicate the demonstrations remaining.

G. Real-World Experiment

In the real-world experiment we compared *contextual MAB* and *human then learner* over one run each of 400 episodes. In the real-world experiments, we used $\epsilon = 0.1$ and a larger neural network with hidden layers of 128 and 64 weights as this was empirically found to improve the performance of the learnt policies. All other training parameters were the same as for the simulated experiments. For *contextual MAB* we used $\alpha = 1$. The costs were $c_d = 1$ and $c_f = 5$.

Results: *contextual MAB* used 142 human demonstrations and incurred a total cost of 442 over the 400 total episodes. With exactly the same number of demonstrations, *human then learner* incurred more failures, accumulating a total cost of 517. The cost incurred throughout the experiment is plotted in Figure 5. These results demonstrate that our approach can easily be applied to real-world robotics problems.

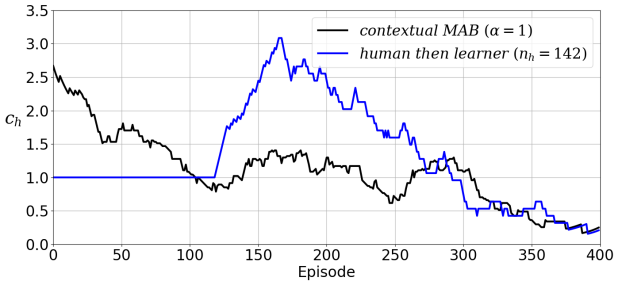


Fig. 5: Human cost in real-world navigation experiment. c_h is plotted using a sliding window average over 40 episodes.

H. Discussion

The α value in our approach enables the user to tune whether the system conservatively asks for more demonstrations, or explores trying the other controllers. A smaller value reduces the risk of failures, but with a higher value the system

more quickly gathers RL experience and more accurately predicts controller performance. Our results from our two simulated domains indicate that simple tasks that can be learnt from demonstrations alone favour a smaller value for α to conservatively ask for demonstrations and avoid failures. For more difficult problems which take longer to learn and require RL experience to train a good policy, a larger value may be favourable to gather more RL experience.

Choosing the controller for each episode based on context is key to the success of our approach. The *Boltzmann* method favours choosing the learnt policy when it begins to perform well overall. However, our method performs better by selectively favouring the autonomous controllers only from initial states where they are likely to succeed. As a side effect, our approach tends to ask for more informative demonstrations, resulting in better performance of the learnt policy with the same number of demonstrations.

Our results indicate that including demonstrations from different sources increases the amount of experience required to learn a good policy. It may be useful to use our approach to learn from many different controllers (for example, a number of different teleoperators who have different techniques). However, the issue of learning effectively from dissimilar demonstrations may need to be addressed before our approach is advantageous with a large number of different controllers.

VI. Conclusion

We have presented an approach to choosing between human teleoperation and autonomous controllers to minimise the cost of bothering the human operator. By estimating the performance of each controller throughout the state space, our system reduces the human cost by only asking for demonstrations when they are needed, and reducing autonomous failures. By learning one of the autonomous controllers online, our system becomes less reliant on the human with more experience. In future work, we will apply the framework presented in this paper to problems where the human operator may make mistakes, and so we will also need to predict human performance. Additionally, we will extend our approach to sequential decision making problems to consider *sequences* of controller selections. To further reduce the human effort required, our framework could be applied using more data-efficient learning methods for the learnt controller, such as model-based RL methods.

References

[1] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband *et al.*, "Deep q-learning from demonstrations," in *AAAI Conference on Artificial Intelligence*, 2018.

[2] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine, "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations," in *Robotics: Science and Systems*, 2018.

[3] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Overcoming exploration in reinforcement learning with demonstrations," in *ICRA*, 2018.

[4] G. Dorais, R. P. Bonasso, D. Kortenkamp, B. Pell, and D. Schreckenghost, "Adjustable autonomy for human-centered autonomous systems on mars," in *Mars society conference*, 1998.

[5] N. Hawes *et al.*, "The STRANDS project: Long-term autonomy in everyday environments," *IEEE RAM*, vol. 24, no. 3, 2017.

[6] M. Chiou, R. Stolkin, G. Bieksaite, N. Hawes, K. L. Shapiro, and T. S. Harrison, "Experimental analysis of a variable autonomy framework for controlling a remotely operating mobile robot," in *IROS*, 2016.

[7] K. H. Wray, L. Pineda, and S. Zilberstein, "Hierarchical approach to transfer of control in semi-autonomous systems," in *AAMAS*. IFAAMAS, 2016.

[8] N. Jansen, M. Cubuktepe, and U. Topcu, "Synthesis of shared control protocols with provable safety and performance guarantees," in *ACC*. IEEE, 2017.

[9] J. A. Clouse, *On integrating apprentice learning and reinforcement learning*. University of Massachusetts Amherst, 1996.

[10] F. Del Duchetto, A. Kucukyilmaz, L. Iocchi, and M. Hanheide, "Do not make the same mistakes again and again: Learning local recovery policies for navigation from human demonstrations," *IEEE RA-L*, vol. 3, no. 4, 2018.

[11] S. Chernova and M. Veloso, "Interactive policy learning through confidence-based autonomy," *JAIR*, vol. 34, 2009.

[12] Z. Lin, B. Harrison, A. Keech, and M. O. Riedl, "Explore, exploit or listen: Combining human feedback and policy model to speed up deep reinforcement learning in 3d worlds," *arXiv preprint arXiv:1709.03969*, 2017.

[13] S. C. Akkaladevi, M. Plasch, C. Eitzinger, S. C. Maddukuri, and B. Rinner, "Towards learning to handle deviations using user preferences in a human robot collaboration scenario," in *International Conference on Intelligent Human Computer Interaction*. Springer, 2016, pp. 3–14.

[14] A. G. Barto and M. T. Rosenstein, "Supervised actor-critic reinforcement learning," *Handbook of learning and approximate dynamic programming*, vol. 2, 2004.

[15] H. B. Suay and S. Chernova, "Effect of human guidance and state space size on interactive reinforcement learning," in *Ro-Man*. IEEE, 2011.

[16] S. Reddy, A. D. Dragan, and S. Levine, "Shared autonomy via deep reinforcement learning," in *Robotics: Science and Systems XIV*, 2018.

[17] W. Schwarting, J. Alonso-Mora, L. Pauli, S. Karaman, and D. Rus, "Parallel autonomy in automated vehicles: Safe motion generation with minimal intervention," in *ICRA*, 2017.

[18] A. Broad, T. D. Murphey, and B. Argall, "Learning models for shared control of human-machine systems with unknown dynamics," *Robotics: Science and Systems XIII*, 2017.

[19] L. Torrey and M. Taylor, "Teaching on a budget: Agents advising agents in reinforcement learning," in *AAMAS*, 2013, pp. 1053–1060.

[20] J. Fu and U. Topcu, "Synthesis of shared autonomy policies with temporal logic specifications," *IEEE T-ASE*, vol. 13, no. 1, 2016.

[21] R. Cohen, H. Jung, M. W. Fleming, and M. Y. Cheng, "A user modeling approach for reasoning about interaction sensitive to bother and its application to hospital decision scenarios," *User Modeling and User-Adapted Interaction*, vol. 21, no. 4-5, pp. 441–484, 2011.

[22] B. Lacerda, F. Faruq, D. Parker, and N. Hawes, "Probabilistic planning with formal performance guarantees for mobile service robots," *The International Journal of Robotics Research*, 2019.

[23] J. P. Fentanes, B. Lacerda, T. Krajník, N. Hawes, and M. Hanheide, "Now or later? Predicting and maximising success of navigation actions from long-term experience," in *ICRA*. IEEE, 2015.

[24] F. Fernández and M. Veloso, "Probabilistic policy reuse in a reinforcement learning agent," in *AAMAS*, 2006, pp. 720–727.

[25] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[26] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," in *WWW*. ACM, 2010.

[27] A. Garivier and E. Moulines, "On upper-confidence bound policies for switching bandit problems," in *ALT*. Springer, 2011.

[28] R. Goetschalckx, P. Poupart, and J. Hoey, "Continuous correlated beta processes," in *IJCAI*, 2011.

[29] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *ICLR*, 2016.

[30] H. Jakab and L. Csató, "Improving Gaussian process value function approximation in policy gradient algorithms," in *ICANN*, 2011.

[31] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IROS*, 2004.

[32] G. Echeverria, S. Lemaignan, A. Degroote, S. Lacroix, M. Karg, P. Koch, C. Lesire, and S. Stinckwich, "Simulating complex robotic scenarios with morse," in *SIMPAR*, 2012.

[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.