Learning User Preferences from Corrections on State Lattices

Nils Wilde, Dana Kulić, and Stephen L. Smith

***Abstract*— Enabling a broader range of users to efficiently deploy autonomous mobile robots requires intuitive frameworks for specifying a robot's task and behaviour. We present a novel approach using learning from corrections (LfC), where a user is iteratively presented with a solution to a motion planning problem. Users might have preferences about parts of a robot's environment that are suitable for robot traffic or that should be avoided as well as preferences on the control actions a robot can take. The robot is initially unaware of these preferences; thus, we ask the user to provide a correction to the presented path. We assume that the user evaluates paths based on environment and motion features. From a sequence of corrections we learn weights for these features, which are then considered by the motion planner, resulting in future paths that better fit the user's preferences. We prove completeness of our algorithm and demonstrate its performance in simulations. Thereby, we show that the learned preferences yield good results not only for a set of training tasks but also for test tasks, as well as for different types of user behaviour.**

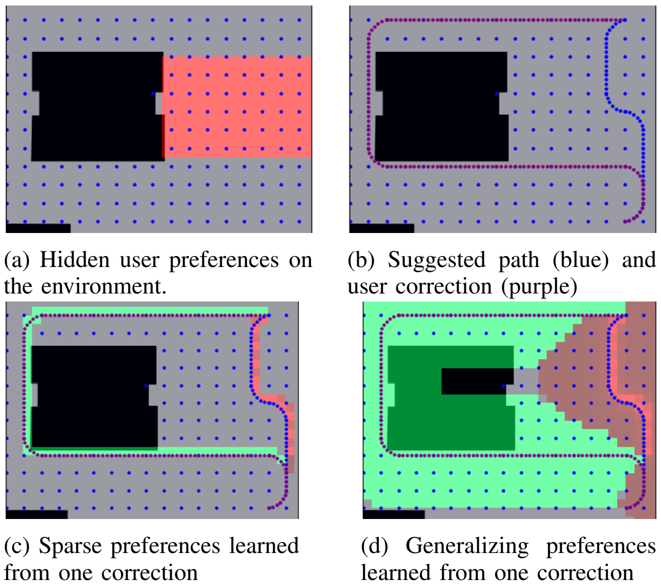

(a) Hidden user preferences on the environment.
(b) Suggested path (blue) and user correction (purple)
(c) Sparse preferences learned from one correction
(d) Generalizing preferences learned from one correction

Fig. 1: Example of the learning from correction framework, with an environment with obstacles (black), avoidance areas (red) and areas where robot traffic is encouraged (green).

I. Introduction

Recent research in human robot interaction (HRI) focuses on enabling inexperienced users to efficiently deploy autonomous mobile robots. Common techniques for intuitive specification of robot tasks and behaviour are imitation learning, often based on inverse reinforcement learning (IRL) [1], [2], active preference learning [3] and learning from corrections (LfC) [4], [5].

User preferences for how a robot should accomplish its task can be described by constraints on the robot's task and action space. For instance, when a mobile robot acts in an environment shared with humans, users might want certain areas to be avoided by the robot while others are more suitable for robot traffic. A set of such constraints defines rewards and penalties for corresponding parts of the task space, usually expressed by weights, which can then be considered by the robot's motion planner. However, a user may find it challenging to specify constraints on robot motion while ensuring good task performance, for instance traffic rules for mobile robots performing material transport tasks [6]. Instead of asking users to define constraints, in the LfC approach, we suggest a solution for a task, i.e., a path from a start to a goal location, and then ask the user to provide a correction if that solution does not fit their preferences. This can be done on an interface showing a map of the environment together with the presented path. The user then can correct the parts of the presented path they dislike by either setting via-points or by drawing an alternative sub-path. From the difference between the presented solution and its correction we learn about the user's preferences for robot behavior in different parts of the environment. We illustrate this in Figure 1. In (a) the environment is shown together with a penalty area, i.e., the user prefers that a robot does not traverse the red-shaded area on the right. We do not ask the user to specify this region, but rather show them the current optimal solution computed by the motion planner. The user then provides a correction, as shown in (b), from which the planner learns that there must be some user preference making the presented path inferior.

Given a set of tasks, users are queried for corrections over multiple iterations. From the user feedback, we learn about user preferences about the robot's state relative to the *environment*, e.g., areas where robot traffic is encouraged or discouraged, and preferences about the robot's *motion* describing control actions that should be avoided such as sharp turn maneuvers. It is neither necessary nor realistic to expect that the learned preferences exactly equal the hidden ones. Rather, the objective is that the learned preferences result in a similar behaviour, i.e., the planner finds paths that are similar to the ones the planner would find after a user had precisely defined constraints. Further,

This research is partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and OTTO Motors.

The authors would like to thank Alex Botros for valuable discussions.

N. Wilde and S. Smith are with the Department of Electrical and Computer Engineering, University of Waterloo, D. Kulić is with the University of Waterloo and Monash University. (nwilde@uwaterloo.ca; dana.kulic@uwaterloo.ca; stephen.smith@uwaterloo.ca)

the learning should generalize the information obtained from the corrections, as illustrated in Figure 1 (c) and (d). When updating preferences without any generalization, the planner might only avoid the exact part of the environment traversed by the presented path, or prefer to use the part of environment used by the correction, shown in sub-figure (c). As a result, in the next iteration it might propose a path that is only slightly different than the one presented previously. The goal of generalizing the learned information is to infer the user's intent when they provide a correction. This can be done by updating the weights for the area around the presented path and the correction, as illustrated in sub-figure (d). This potentially allows for faster learning and might improve the performance on tasks for which no corrections were obtained, i.e., that were not available during the interaction with the user.

Related Work: Our framework for learning from corrections (LfC) combines inverse reinforcement learning (IRL) [1], [2], [7], [8] and active preference learning [3], [9].

In IRL, or learning from demonstrations, a user usually demonstrates the desired behaviour, e.g., trajectories [2]. Assuming that the user is optimizing some hidden cost function, the robot then tries to learn this function in order to reproduce the demonstrations. In active preference learning users are presented with possible solutions for a task and provide feedback by ranking them. From this feedback a cost function that describes the user's preferences can be learned. For a Bayesian user model, [3] presents a near optimal learning method considering noise. This work was later adapted by the authors of [9] to a comparison based learning algorithm that accounts for user uncertainty when the presented paths are similar. Our previous work uses active preference learning to revise user specifications [6], [10] based on spatial features that describe which areas of the task space are traversed by a robot.

Learning from corrections was proposed as an alternative to IRL in [4], aiming to reduce the burden on the user by not asking for optimal demonstrations, but only to iteratively improve the current behaviour of a manipulator robot. Recent work in LfC investigates how to extrapolate the information obtained from a correction to learn about the user cost function efficiently [11]. The authors of [12] introduce a Kalman Filter for LfC to express uncertainty for the learned preferences. In [5] corrective demonstrations are used to learn about task models described by a finite-state automaton. In this work we adapt our previous cost function [6] to a lattice planner [13], evaluating a robot's motion based on motion and environment features of trajectories which are generic, i.e., not explicitly chosen for a specific application but rather based on the lattice graph.

Contributions: We study a novel framework for learning user preferences from corrections for a state lattice motion planner. In our previous work learning from user preferences [6], [10], our linear cost function considered only *environment* features characterizing robot behaviour based on its current location in the environment. Using a lattice planner allows us to include *motion* features that allow us to model preferences about control actions across the entire state lattice. From a user's correction to a presented path we derive inequalities about weights for both types of features. Given a sequence of presented paths and their corrections, we update the weights to satisfy the user's preferences while generalizing the information learned about weights to improve the performance. We prove completeness of our algorithm and demonstrate its performance in simulations. Thereby, we show the weights learned over 20 iterations result in paths similar to the optimal paths for tasks used during learning as well as for a set of test tasks.

A. Preliminaries

Graph Theory: Following [14], a graph is an ordered pair $G = (V, E)$, where V is a set of vertices and E is a set of edges. In a weighted graph $G = (V, E, w)$ a real valued function associates a weight to each edge of the graph: $w : E(G) \to \mathbb{R}$. A walk between two vertices v_1 and v_{k+1} on a graph G is a finite sequence of vertices and edges $v_1, e_1, v_2, e_2, \ldots, e_k, v_{k+1}$ where $e_i = (v_i, v_{i+1})$ for all $i = 1, 2, \ldots, k$ and $e_1, e_2, \ldots e_k$ are distinct. A path $P_{v_1,v_{k+1}}$ between two vertices v_1 and v_{k+1} is defined as a graph $(\{v_1, v_2, \ldots, v_{k+1}\}, \{e_1, e_2, \ldots, e_k\})$ where $v_1, e_1, v_1, e_2, \ldots, e_k, v_{k+1}$ is a walk. On a weighted graph, the cost of a path is defined as $c(P) = \sum_{e \in E(P)} w(e)$.

State lattice planner: Following [13] a lattice point is a triple (x, y, θ) in a two dimensional Euclidean work space $W \subset SE(2)$. We consider a discrete set of lattice points with a heading set Θ and locations x and y on a regular grid with Δx and Δy resolutions. A robot navigates between lattice points using a set of pre-defined controls, i.e., motion primitives, B. For any given heading $\theta' \in \Theta$, there exists a subset B' with the control actions that can be applied to all lattice points of the form (x', y', θ'). Any lattice points with identical relative positions are connected by applying the same motion primitive. The set of all lattice points define a set of vertices V while the primitive paths connecting any two lattice points uniquely define edges E. The graph (V, E) can be weighted using the length of the primitive paths.

Notation: Vectors are written with bold, lower case letters, e.g., $\boldsymbol{v}$, we address elements of the vector with a subscript index v_i. A superscript index $\boldsymbol{v}^i$ identifies a specific vector. Sets are denoted by upper case letters (G).

II. Problem Formulation

In the problem we consider the following input:

- A directed graph $G = (V, E, t)$ induced by a state lattice [13] with a control set, i.e., motion primitives, B. The graph G encodes a robotic road-map of the environment. The travel times t_i are non-negative for all $e_i \in E$.
- A set of training tasks, $\mathcal{T}^{\texttt{train}}$ and a set of *hidden* test tasks $\mathcal{T}^{\texttt{test}}$. Both contain ordered pairs $\{(s_1, g_1), (s_2, g_2), \ldots\}$ where s_j and g_j are vertices on G and constitute the start and goal, for which we want to find the shortest path that satisfies all user preferences.

- A grid-based cost map M [13] of the environment. Each cell $\mu_i \in M$ has a *hidden* user weight w_i^* where $w_i^* \in [0, w^{\max}]$. Weights $w_i > 1$ express that it is undesired for robots to visit cell μ_i while $w_i < 1$ correspond to a reward. We collect all weights in a vector $\boldsymbol{w}^* \in \mathbb{R}_{\geq 0}^{|M|}$.
- A *hidden* vector $\boldsymbol{u}^* \in \mathbb{R}_{\geq 1}^{|B|}$ where each u_l^* describes the user's preference for the robot using the motion primitive $b_l \in B$.
- Vectors $\boldsymbol{w}^0$ and $\boldsymbol{u}^0$ with prior weights.
- A user providing feedback: Given a path P for a task (s_i, g_i) the user provides a corrected path Q.

The graph G can be combined with weights $\boldsymbol{w}$ and $\boldsymbol{u}$ to obtain the graph $G^{\boldsymbol{w},\boldsymbol{u}} = (V, E, c)$. Given an edge e and its primitive path, let $\phi_j(e)$ be the length of the section of its primitive path in the cell μ_j. Further, let $\eta_l(e)$ be a binary function indicating if the edge e is an instance of the primitive b_l. The cost c_i of an edge e_i is then defined as

$$c_i = \sum_{\mu_j \in M} w_j \cdot \phi_j(e_i) + \sum_{b_l \in B} u_l \cdot t_l \cdot \eta_l(e_i). \tag{1}$$

Thus, each w_j expresses a relative reward or penalty for an edge passing through the cell μ_j. The weight u_l describes a preference for using the corresponding motion primitive. As $w_j \geq 0$ for all μ_j and $u_l \geq 1$ for all b_l, we ensure that $c_i \geq 0$ always holds. As a consequence, the graph cannot contain negative cycles. As $\boldsymbol{w}^*$ and $\boldsymbol{u}^*$ are hidden we can only obtain the graph G^0, composed of G, $\boldsymbol{w}^0$ and $\boldsymbol{u}^0$. The objective is to learn about all weights $\boldsymbol{w}$ and $\boldsymbol{u}$ such that the behaviour of a robot planning with the learned weights is equivalent to the behaviour described by $\boldsymbol{w}^*$ and $\boldsymbol{u}^*$, i.e., that the optimal solution for all tasks is the same.

A. User cost function

We model the user to have an internal linear cost function for paths, extending our prior work in [6] to incorporate both environment and motion features. Given a graph $G^{\boldsymbol{w},\boldsymbol{u}}$ combined from G with some $\boldsymbol{w}$ and $\boldsymbol{u}$, the cost of a path is

$$C(P, \boldsymbol{w}, \boldsymbol{u}) = \sum_{e_i \in P} c_i. \tag{2}$$

Let $\boldsymbol{\phi}(P)$ be the vector summarizing the path length for each cell $\mu \in M$, defined as

$$\boldsymbol{\phi}(P) = \left[\sum_{e \in P} \phi_1(e) \ \sum_{e \in P} \phi_2(e) \ \dots \ \sum_{e \in P} \phi_{|M|}(e) \right]. \tag{3}$$

Similarly, $\boldsymbol{\eta}(P)$ counts how often each primitive $b \in B$ is used on the path P and multiplies it with the duration of b:

$$\boldsymbol{\eta}(P) = \left[\sum_{e \in P} t_1 \eta_1(e) \ \sum_{e \in P} t_2 \eta_2(e) \ \dots \ \sum_{e \in P} t_{|B|} \eta_{|B|}(e) \right]. \tag{4}$$

This allows us to rewrite the cost function as

$$C(P, \boldsymbol{w}, \boldsymbol{u}) = \boldsymbol{\phi}(P)\boldsymbol{w} \ + \boldsymbol{\eta}(P)\boldsymbol{u}. \tag{5}$$

Similar to reward functions in reinforcement learning, $\boldsymbol{\phi}(P)$ and $\boldsymbol{\eta}(P)$ describe features of a path, which are then weighted by $\boldsymbol{w}$ and $\boldsymbol{u}$, respectively. We call $\boldsymbol{\phi}(P)$ *environment* features as they describe a preference for the robot's behaviour in particular regions of the environment. The vector $\boldsymbol{\eta}(P)$ expresses *motion* features for the robot; they describe preferences over the motion primitives in the lattice and apply globally. For instance a user might want the robot to avoid sharp curves to reduce risk, or left turns as they can affect performance in dense traffic.

B. Problem Statement

We consider arbitrary weights $\boldsymbol{w}$ and $\boldsymbol{u}$ defining a graph $G^{\boldsymbol{w},\boldsymbol{u}}$, as well as the hidden optimal weights $\boldsymbol{w}^*$ and $\boldsymbol{u}^*$ defining a graph G^*. The graphs $G^{\boldsymbol{w},\boldsymbol{u}}$ and G^* have the same vertices and edges, but different costs c on the edges. For any task (s_j, g_j) we can find a shortest path on $G^{\boldsymbol{w},\boldsymbol{u}}$, denoted by P_j and a shortest path on G^*, denoted by P_j^*. Finally, let $C(P_j, \boldsymbol{w}^*, \boldsymbol{u}^*)$ denote the cost of a path P_j on G^*. This allows us to define the loss function for a set of tasks $\mathcal{T}$ as the summed relative error in cost:

$$L(\mathcal{T}, \boldsymbol{w}, \boldsymbol{u}) = \sum_{(s_j, g_j) \in \mathcal{T}} \frac{C(P_j, \boldsymbol{w}^*, \boldsymbol{u}^*)}{C(P_j^*, \boldsymbol{w}^*, \boldsymbol{u}^*)} - 1. \tag{6}$$

We notice that this error cannot be calculated as $\boldsymbol{w}^*$ and $\boldsymbol{u}^*$ are hidden. Nonetheless, the objective is to find weights $\boldsymbol{w}$ and $\boldsymbol{u}$ that minimize the error for all tasks.

Problem 1 (Learning user preference)**.** Given G and $\mathcal{T}^{\texttt{train}}$ and a budget of K user interactions, find weights $\begin{bmatrix}\boldsymbol{w}^K & \boldsymbol{u}^K\end{bmatrix}$ where

$$\begin{aligned} \begin{bmatrix}\boldsymbol{w}^K & \boldsymbol{u}^K\end{bmatrix} = \underset{[\boldsymbol{w}' \ \boldsymbol{u}']}{\arg\min}\ & L(\mathcal{T}^{\texttt{train}} \cup \mathcal{T}^{\texttt{test}}, \boldsymbol{w}', \boldsymbol{u}') \\ \text{s.t. } & 0 \leq w_j', \ j = 1, 2, \dots, |M| \\ & 1 \leq u_l', \ l = 1, 2, \dots, |B|. \end{aligned} \tag{7}$$

III. Approach

We present our framework for solving Problem 1 in Algorithm 1. The approach is similar to active preference learning [10], [15], but while an active preference learning algorithm proposes at least two new paths in each iteration, LfC presents only one path and asks the user for a correction. In contrast to our previous work [6], [10], the user is not required to provide any initial input such as specifying constraints. In each iteration k we maintain an estimate of the weights $\begin{bmatrix}\boldsymbol{w}^k & \boldsymbol{u}^k\end{bmatrix}$, based on the user corrections obtained so far. A planner can compute shortest paths for all tasks based on $\begin{bmatrix}\boldsymbol{w}^k & \boldsymbol{u}^k\end{bmatrix}$, from which we randomly select one path that is shown to the user (line 5). After the user provides a correction (line 6), the weights are updated (line 11).

In the next sections we describe our user model for how they provide corrections and show how weights are updated. Finally, we prove completeness of the algorithm.

A. User Model

In our framework users are presented with a path for a task (s_j, g_j) and provide feedback in the form of corrections:

Algorithm 1: Learning from Corrections

```
   Input: $G$, $\mathcal{T}^{\mathtt{train}}$, $[\boldsymbol{w}^0\ \boldsymbol{u}^0]$, $K$
   Output: $[\boldsymbol{w}^k\ \boldsymbol{u}^k]$
1  Initialize $\Psi = \emptyset$, $\mathcal{T} = \mathcal{T}^{\mathtt{train}}$
2  for $k = 1$ to $K$ do
3      if $\mathcal{T} = \emptyset$ then
4          return $[\boldsymbol{w}^k\ \boldsymbol{u}^k]$
5      $P^k$, $(s_j, g_j) \leftarrow$ Sample_new_path($\mathcal{T}$)
6      $Q^k \leftarrow$ Get_user_correction($P^k$)
7      if $Q^k = \emptyset$ then
8          $\mathcal{T} = \mathcal{T} \setminus (s_j, g_j)$
9          continue
10     $\Psi \leftarrow \Psi \cup \{P^k, Q^k\}$
11     $[\boldsymbol{w}^k\ \boldsymbol{u}^k] \leftarrow$ Update($\boldsymbol{w}^0, \boldsymbol{u}^0, \boldsymbol{w}^{k-1}\boldsymbol{u}^{k-1}, \Psi$)
12 return $[\boldsymbol{w}^k\ \boldsymbol{u}^k]$
```

Given a path P they return a corrected path Q. We assume that the correction fits the user preferences better than P, i.e., has a lower cost with respect to the hidden weights, as summarized in the following Assumption:

Assumption 1 (User feedback)**.** Given a path P with $C(P, \boldsymbol{w}^*, \boldsymbol{u}^*) > C(P^*, \boldsymbol{w}^*, \boldsymbol{u}^*)$ the user returns a corrected path Q such that

$$C(Q, \boldsymbol{w}^*, \boldsymbol{u}^*) < C(P, \boldsymbol{w}^*, \boldsymbol{u}^*). \tag{8}$$

If $C(P, \boldsymbol{w}^*, \boldsymbol{u}^*) = C(P^*, \boldsymbol{w}^*, \boldsymbol{u}^*)$ an empty set is returned.

In case the presented path is optimal and thus an empty set was returned the algorithm cannot learn about the weights any more from that task and discards it (line 8).

B. Learning from Corrections

In order to update the weights in line 11 of Algorithm 1 we show how information about weights is derived from user corrections. Using the user cost function from equation (5) allows us to rewrite inequality (8) as

$$(\boldsymbol{\phi}(Q) - \boldsymbol{\phi}(P))\, \boldsymbol{w}^* + (\boldsymbol{\eta}(Q) - \boldsymbol{\eta}(P))\, \boldsymbol{u}^* < 0. \tag{9}$$

To avoid a strict inequality we use some small ϵ:

$$(\boldsymbol{\phi}(Q) - \boldsymbol{\phi}(P))\, \boldsymbol{w}^* + (\boldsymbol{\eta}(Q) - \boldsymbol{\eta}(P))\, \boldsymbol{u}^* \leq -\epsilon. \tag{10}$$

This then defines a half-space for feasible weights for each (P^k, Q^k). Hence, any weight $[\boldsymbol{w}\ \boldsymbol{u}]$ within that half-space guarantees that Q^k has a lower cost than P^k. Further, as we can only observe corrections, all weights within the half-space are indistinguishable from $[\boldsymbol{w}^*\ \boldsymbol{u}^*]$ with respect to the user cost function, and thus we can pick any such $[\boldsymbol{w}\ \boldsymbol{u}]$.

C. Updating weights

Feasible weights: Our primary objective is to find weights $\boldsymbol{w}^k$ and $\boldsymbol{u}^k$ that minimize the loss function $L(\boldsymbol{w}^k, \boldsymbol{u}^k)$. However, as $\boldsymbol{w}^*$ and $\boldsymbol{u}^*$ are hidden $L(\boldsymbol{w}^k, \boldsymbol{u}^k)$ cannot be computed. Nonetheless, we say weights are feasible if they are consistent with the user feedback, i.e., satisfy the inequalities based on Assumption 1. Given a sequence of paths and their user corrections $\Psi = (P^1, Q^1, P^2, Q^2, \ldots, P^k, Q^k)$, we can intersect the half-spaces described in equation (10) to define a convex set of feasible weights. This allows us to write the objective as a constraint in the optimization problem: To minimize $L(\boldsymbol{w}^k, \boldsymbol{u}^k)$ we have to pick weights $\boldsymbol{w}^k$ and $\boldsymbol{u}^k$ such that equation (10) holds for all $k = 1, 2, \ldots, K$.

Generalization: Among all weights that satisfy the inequalities from Assumption 1, we want to choose $\boldsymbol{w}^k$ and $\boldsymbol{u}^k$ that *generalize* the information we obtain for each (P, Q). As we express the objective from equation (7) as constraints, we can choose a new objective function $g(\boldsymbol{w}, \boldsymbol{u})$, leading to the optimization problem

$$\begin{aligned} [\boldsymbol{w}^K \quad \boldsymbol{u}^K] = & \underset{[\boldsymbol{w}'\ \boldsymbol{u}']}{\arg\min}\, g(\boldsymbol{w}', \boldsymbol{u}') \\ \text{s.t.} \quad & \left(\boldsymbol{\phi}(Q^k) - \boldsymbol{\phi}(P^k)\right) \boldsymbol{w}' \\ & + \left(\boldsymbol{\eta}(Q^k) - \boldsymbol{\eta}(P^k)\right) \boldsymbol{u}' \\ & \leq -\epsilon, \text{ for } k = 1, 2, \ldots, K \\ & 0 \leq w'_j, \text{ for } j = 1, 2, \ldots, |M| \\ & 1 \leq u'_l, \text{ for } l = 1, 2, \ldots, |B|. \end{aligned} \tag{11}$$

The idea of generalization is that the learned weights do not only affect edges that belong to the presented path or the obtained correction, but to rather obtain more homogeneous *environment* weights, i.e., encourage neighbouring cells to have the same weight. This is motivated by observations from previous user studies, where users typically have preferences for areas in the environment instead of single locations [6]. Let $N(\mu_j)$ be the set of cells in M that are adjacent to μ_j. We want to minimize the mean square difference in weights between neighbours. On the other hand, it is desirable to avoid weights $w_j \neq 1$. This can be captured by an l_2-norm regularization as used in machine learning [16]. Combining the generalization with the regularization, we obtain

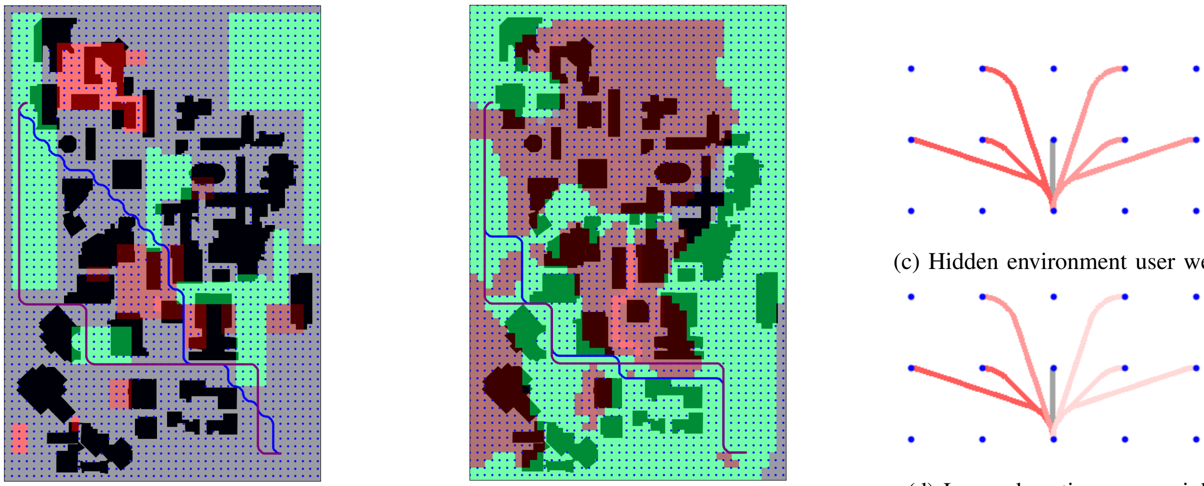
$$g(\boldsymbol{w}) = \lambda \sum_{\mu_i \in M} \sum_{\mu_j \in N(\mu_i)} (w_i - w_j)^2 + (1-\lambda) \sum_{\mu_i \in M} (w_i - 1)^2. \tag{12}$$

Thereby, λ takes values in $[0, 1)$ and balances between generalization and regularization. We do not allow $\lambda = 1$ as this would result in all weights being close to zero. For the *motion* features we only use a regularization $g(\boldsymbol{u}) = \sum_{b_l \in B} (u_l - 1)^2$. Finally, we define the objective in equation (7) as $g(\boldsymbol{w}, \boldsymbol{u}) = g(\boldsymbol{w}) + g(\boldsymbol{u})$. We notice that $g(\boldsymbol{w}, \boldsymbol{u})$ is a convex quadratic function implying that the optimization problem in equation (11) can be solved in polynomial time.

D. Completeness

Based on the user model and the proposed weight update we now show that Algorithm 1 is complete.

Proposition 1 (Completeness)**.** For each problem instance there exists a finite K such that Algorithm 1 finds an optimal solution, i.e., returns weights $[\boldsymbol{w}^K\ \boldsymbol{u}^K]$ where $L(\boldsymbol{w}^K, \boldsymbol{u}^K) = 0$.


(a) Hidden user weights $\boldsymbol{w}^*$ with initially presented path and optimal correction.

(b) Learned weights $\boldsymbol{w}^{20}$ using $\lambda = 0.5$ with learned path and optimal correction.

(c) Hidden environment user weights $\boldsymbol{u}^*$

(d) Learned motion user weights $\boldsymbol{u}^{20}$.

Fig. 2: Environment with learned environment and motion weights. Red cells indicate a weight of $w_j > 1$, i.e., a penalty. Green cells correspond to weights $w_j < 1$ belonging to a reward while grey cells have a weight of $w_j = 1$. The blue and purple lines show the learned path and the optimal path for an example task. In (c) and (d) red indicates weights $u_l > 1$. Even though the learned cost map in (b) has different weights on many cells than the optimal cost map in (a), they lead to similar optimal paths on the training and test task set.

Proof. Consider the case that the path P^k presented to the user is optimal. Then the corresponding task (s_j, g_j) gets removed from $\mathcal{T}$ and $C(P_j^k, \boldsymbol{w}^*, \boldsymbol{u}^*)/C(P_j^*, \boldsymbol{w}^*, \boldsymbol{u}^*) - 1 = 0$ (line 8 in Algorithm 1).

If P^k is not optimal, a correction Q^k satisfying Assumption 1 is obtained from which we derive an inequality as in equation (10). Then all weights $\left[\boldsymbol{w}^l \; \boldsymbol{u}^l\right]$ where $l > k$ satisfy this inequality, implying that for any $l > k$ the path P^k is never optimal for any feasible weight vector $\left[\boldsymbol{w}^l \; \boldsymbol{u}^l\right]$ and thus cannot be presented again. That guarantees that each update removes at least one path from the set of all paths that could be shown for the same task in a later iteration.

For every pair (s_j, g_j) there exists only a finite number of paths; hence, a path P^l where $C(P_j^l, \boldsymbol{w}^*, \boldsymbol{u}^*) = C(P_j^*, \boldsymbol{w}^*, \boldsymbol{u}^*)$ must be presented to the user after a finite number of iterations, and $C(P_j^l, \boldsymbol{w}^*, \boldsymbol{u}^*)/C(P_j^*, \boldsymbol{w}^*, \boldsymbol{u}^*) - 1 = 0$ for every task (s_j, g_j) is achieved. $\square$

Proposition 1 ensures that the presented framework learns weights for robot motion planning that result in the same behaviour as described by the hidden optimal weights $\boldsymbol{w}^*$. In the next section we evaluate the performance in a realistic transportation scenario.

IV. Evaluation

In the evaluation we use a state lattice with $11,008$ vertices, based on the layout of the campus of the University of Waterloo, and a cost map with $2,752$ quadratic cells. The user corrections are simulated. Each simulated user has a hidden cost map with weights $\boldsymbol{w}^*$ such that the map is structured into neighbourhoods of multiple cells that have equal weights. This allows to define areas in the environment that are rewarded or penalized, i.e., have a weight different than one, which we call constraints. In each trial the user weights are generated by randomly drawing from a subset of 60 constraints. These constraints were predefined such that narrow gaps on the environment might be blocked by penalty constraints while wide open areas are usually rewarded. An example specification with 25 constraints is illustrated in Figure 2 (a), Further, in the experiments we use the control set $B = \{(0,1,0), (1,1,\pi/2), (-1, 1{-}\pi/2), (2,1,\pi/2), (-2,1,-\pi/2), (1,2,\pi/2), (-1,2,-\pi/2)\}$, always with *motion* user preferences that heavily penalize left turns and slightly penalize right turns, as shown in sub-figure (c).

We consider two types of user behaviour: *Optimal users* that always provide a correction that is an optimal solution for the task, and *uniform users* who randomly generate a correction satisfying Assumption 1. As the optimal weights $\boldsymbol{w}^*$ and $\boldsymbol{u}^*$ are known to the simulated user, optimal paths can be computed using shortest path search. For uniform users, the correction is the shortest path Q for some weights $\bar{\boldsymbol{w}}$ and $\bar{\boldsymbol{u}}$. We generate $\bar{\boldsymbol{w}}$ using a random walk: Let $\boldsymbol{w}$ be the weight for which the presented path P is optimal. Then, the random walk to find $\bar{\boldsymbol{w}}$ is constrained by the polyhedron $\{\boldsymbol{w}' \in [0, w^{\max}| \min\{w_i, w_i^*\} \leq w_i \leq \max\{w_i, w_i^*\}, \forall \mu_i \in M\}$. This ensures that Q has a lower cost than P.

All experiments were repeated for 24 trials where each trial runs Algorithm 1 with $K = 20$ and a uniform prior for $\boldsymbol{w}$ and $\boldsymbol{u}$. In every trial the set of training and test tasks are randomly generated pairs of vertices. The run time for the quadratic program with 2752 variables is $\approx 9s$; finding a new query takes $\approx 11s$ for 5 training tasks and up to $20s$ for 10 training tasks. However, the query generation can potentially be sped up using a parallel implementation.

Experiment 1 - Generalization: First, we investigate how different values for λ affect the algorithm. We use an optimal user, 25 constraints, 5 training and 20 test tasks.

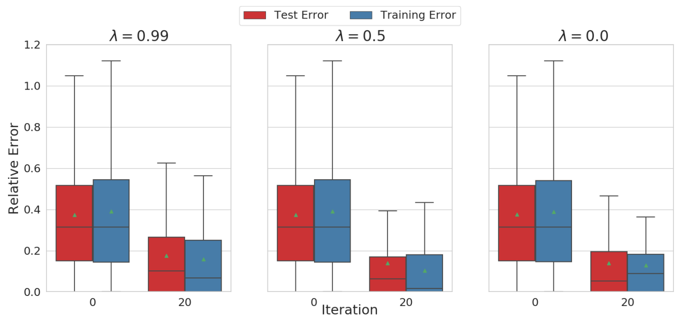


Fig. 3: Comparison of different values for λ.

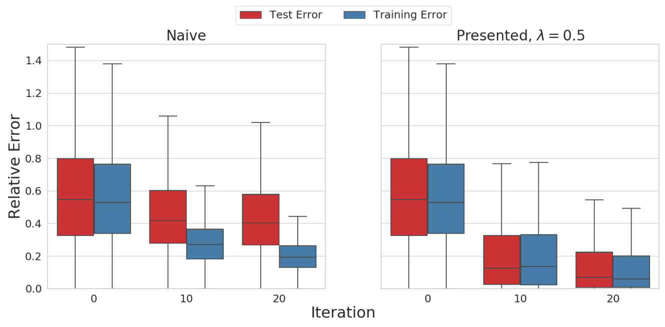


Fig. 4: Comparison between the naive and the proposed approach, with $\lambda = 0.5$.

We illustrate the relative errors $L(\mathcal{T}^{\texttt{test}}, \boldsymbol{w}^K, \boldsymbol{u}^K)$ and $L(\mathcal{T}^{\texttt{train}}, \boldsymbol{w}^K, \boldsymbol{u}^K)$ before and after learning in Figure 3.

We observe no conclusive difference in the test error between $\lambda = 0.0$ and $\lambda = 0.5$. However, the final median training error is 0.09 for $\lambda = 0.0$, i.e., paths generated with the learned weights have 9% higher cost than those with the hidden user weight. The final median training error for $\lambda = 0.5$ is 0.01. Thus, using a generalization allows for more efficient learning on the training set, as it avoids presenting paths that are too similar to those previously presented. On the other hand, the training and test error is worse for $\lambda = 0.99$ than for $\lambda = 0.5$. A large generalization overestimates the size of user constraints and leads to poor behaviour. In the subsequent experiments we use $\lambda = 0.5$.

Experiment 2 - Naive algorithm: In the second experiment we compare the presented algorithm to a naive approach. Given a pair (P, Q) the naive approach updates the weights on the cost map by directly penalizing cells that are traversed by the presented path but not in the correction and rewarding cells which are traversed by only the correction. Hence,

$$w_j^{k+1} = \begin{cases} 2\, w_j^k & \text{if } \phi_j(P) > 0 \text{ and } \phi_j(Q) = 0, \\ {}^1\!/_2\, w_j^k & \text{if } \phi_j(P) = 0 \text{ and } \phi_j(Q) > 0, \\ w_j^k & \text{otherwise.} \end{cases} \quad (13)$$

We use the same experimental setup as before, but with 10 training tasks to reduce the risk of over-fitting for the naive approach. The results are illustrated in Figure 4.

The naive approach shows moderate results for the training data with a final median error of 0.19. However, the progress on the test error is marginal with a final median of 0.41. In contrast, our algorithm improves on both sets of tasks, achieving a final median error of 0.06 on the training and 0.07 on the test set. We conclude by noting that the optimization problem in equation (11) generalizes about environment weights and achieves good performance on training instances. Moreover, the algorithm successfully generalizes the information obtained from the corrections and thus achieves good results on the test tasks.

Experiment 3 - User types: In this experiment we investigate how our algorithm behaves for the two different types of users, using the same setup as for the previous experiment. The results are summarized in Figure 5. Overall

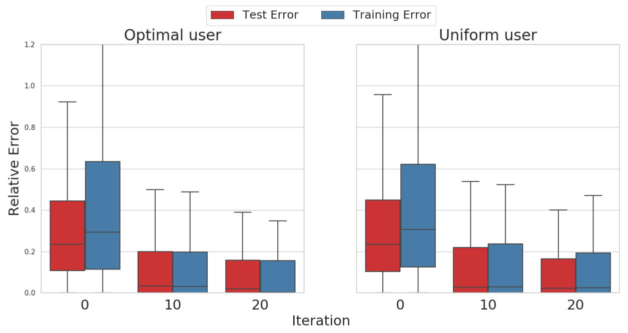


Fig. 5: Relative error when using $\lambda = 0.5$ for optimal and uniform user corrections.

we observe that the learning behaves similarly for both user types. The optimal user shows a better result on the training data where 42% of trials achieve a final error of 0, compared to 31% for the uniform user. However, the final median values are 0.02 and 0.01 for the test and training error of the optimal user and 0.03 and 0.02 for the uniform user, respectively. Hence, the optimal user leads to only marginally better results. In conclusion, the third experiment shows that the presented algorithm learns about the user's preferences efficiently even when the user is not always providing optimal corrections.

V. Discussion and Future Work

We propose an LfC framework for a state lattice planner that learns about environment and motion user preferences. We update weights using a quadratic program, which is constrained by the assumption that users provide corrections that fit their preferences better than the presented path, while the objective function combines generalization with regularization. In simulations we show that the proposed approach generates paths that closely approximate user preferences for both training and test tasks after only a few corrections, with either optimal or non-optimal user corrections.

Future work directions include a more robust user model by introducing noise as well as an active selection of the path that is presented. Further, a user study should investigate how well the learning framework captures the preferences of real users from observing corrections.

REFERENCES

[1] P. Abbeel and A. Y. Ng, “Apprenticeship learning via inverse reinforcement learning,” in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 1.

[2] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, J. Peters *et al.*, “An algorithmic perspective on imitation learning,” *Foundations and Trends® in Robotics*, vol. 7, no. 1-2, pp. 1–179, 2018.

[3] D. Golovin, A. Krause, and D. Ray, “Near-optimal bayesian active learning with noisy observations,” in *NIPS*, 2010, pp. 766–774.

[4] A. Jain, S. Sharma, T. Joachims, and A. Saxena, “Learning preferences for manipulation tasks from online coactive feedback,” *The International Journal of Robotics Research*, vol. 34, no. 10, pp. 1296–1313, 2015.

[5] R. A. Gutierrez, V. Chu, A. L. Thomaz, and S. Niekum, “Incremental task modification via corrective demonstrations,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1126–1133.

[6] N. Wilde, A. Blidaru, S. L. Smith, and D. Kulic, “Improving user specifications for robot behavior through active preference learning: Framework and evaluation,” *The International Journal of Robotics Research*, 2020.

[7] A. Shah, P. Kamath, J. A. Shah, and S. Li, “Bayesian inference of temporal task specifications from demonstrations,” in *Advances in Neural Information Processing Systems*, 2018, pp. 3804–3813.

[8] S. Thakur, H. van Hoof, J. C. G. Higuera, D. Precup, and D. Meger, “Uncertainty aware learning from demonstrations in multiple contexts using bayesian neural networks,” pp. 768–774, 2019.

[9] R. Holladay, S. Javdani, A. Dragan, and S. Srinivasa, “Active comparison based learning incorporating user uncertainty and noise,” in *RSS Workshop on Model Learning for Human-Robot Communication*, 2016.

[10] N. Wilde, D. Kulić, and S. L. Smith, “Bayesian active learning for collaborative task specification using equivalence regions,” *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1691–1698, April 2019.

[11] J. Y. Zhang and A. D. Dragan, “Learning from extrapolated corrections,” in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 7034–7040.

[12] D. P. Losey and M. K. OMalley, “Including uncertainty when learning from human corrections,” in *Conference on Robot Learning*, 2018, pp. 123–132.

[13] M. Pivtoraiko, R. A. Knepper, and A. Kelly, “Differentially constrained mobile robot motion planning in state lattices,” *Journal of Field Robotics*, vol. 26, no. 3, pp. 308–333, 2009.

[14] B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms*, 4th ed. Springer Publishing Company, Inc., 2007.

[15] D. Sadigh, A. D. Dragan, S. Sastry, and S. A. Seshia, “Active preference-based learning of reward functions,” in *Robotics: Science and Systems (RSS)*, 2017.

[16] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.