Abstract—In shared autonomy, user input and robot autonomy are combined to control a robot to achieve a goal. Often, the robot does not know a priori which goal the user wants to achieve, and must both predict the user’s intended goal, and assist in achieving that goal. We formulate the problem of shared autonomy as a Partially Observable Markov Decision Process with uncertainty over the user’s goal. We utilize maximum entropy inverse optimal control to estimate a distribution over the user’s goal based on the history of inputs. Ideally, the robot assists the user by solving for an action which minimizes the expected cost-to-go for the (unknown) goal. As solving the POMDP to select the optimal action is intractable, we use hindsight optimization to approximate the solution. In a user study, we compare our method to a standard predict-then-blend approach. We find that our method enables users to accomplish tasks more quickly while utilizing less input. However, when asked to rate each system, users were mixed in their assessment, citing a tradeoff between maintaining control authority and accomplishing tasks quickly.

I. INTRODUCTION

Robotic teleoperation enables a user to achieve their intended goal by providing inputs into a robotic system. In direct teleoperation, user inputs are mapped directly to robot actions, putting the burden of control entirely on the user. However, input interfaces are often noisy, and may have fewer degrees of freedom than the robot they control. This makes operation tedious, and many goals impossible to achieve.

Shared Autonomy seeks to alleviate this by combining teleoperation with autonomous assistance.

A key challenge in shared autonomy is that the system may not know a priori which goal the user wants to achieve. Thus, many prior works [14, 1, 27, 7] split shared autonomy into two parts: 1) predict the user’s goal, and 2) assist for that single goal, potentially using prediction confidence to regulate assistance. We refer to this approach as predict-then-blend.

In contrast, we follow more recent work [11] which assists for an entire distribution over goals, enabling assistance even when the confidence for any particular goal is low. This is particularly important in cluttered environments, where it is difficult - and sometimes impossible - to predict a single goal.

We formalize shared autonomy by modeling the system’s task as a Partially Observable Markov Decision Process (POMDP) [21, 12] with uncertainty over the user’s goal. We assume the user is executing a policy for their known goal without knowledge of assistance. In contrast, the system models both the user input and robot action, and solves for an assistance action that minimizes the total expected cost-to-go of both systems. See Fig. 1.

The result is a system that will assist for any distribution over goals. When the system is able to make progress for all goals, it does so automatically. When a good assistance strategy is ambiguous (e.g. the robot is in between two goals), the output can be interpreted as a blending between user input and robot autonomy based on confidence in a particular goal, which has been shown to be effective [7]. See Fig. 2.

Solving for the optimal action in our POMDP is intractable. Instead, we approximate using QMDP [19], also referred to as hindsight optimization [5, 24]. This approximation has many properties suitable for shared autonomy: it is computationally efficient, works well when information is gathered easily [16].
and will not oppose the user to gather information.

Additionally, we assume each goal consists of multiple targets (e.g., an object has multiple grasp poses), of which any are acceptable to a user with that goal. Given a known cost function for each target, we derive an efficient computation scheme for goals that decomposes over targets.

To evaluate our method, we conducted a user study where users teleoperated a robotic arm to grasp objects using our method and a standard predict-then-blend approach. Our results indicate that users accomplished tasks significantly more quickly with less control input with our system. However, when surveyed, users tended towards preferring the simpler predict-then-blend approach, citing a trade-off between control authority and efficiency. We found this surprising, as prior work indicates that task completion time correlates strongly with user satisfaction, even at the cost of control authority [7, 11, 9]. We discuss potential ways to alter our model to take this into account.

II. RELATED WORKS

We separate related works into goal prediction and assistance strategies.

A. Goal Prediction

Maximum entropy inverse optimal control (MaxEnt IOC) methods have been shown to be effective for goal prediction [28, 29, 30, 7]. In this framework, the user is assumed to be an intent driven agent approximately optimizing a cost function. By minimizing the worst-case predictive loss, Ziebart et al. [28] derive a model where trajectory probability decreases exponentially with cost, and show how this cost function can be learned efficiently from user demonstrations. They then derive a method for inferring a distribution over goals from user inputs, where probabilities correspond to how efficiently the inputs achieve each goal [29]. While our framework allows for any prediction method, we choose to use MaxEnt IOC, as we can directly optimize for the user’s cost in our POMDP.

Others have approached the prediction problem by utilizing various machine learning methods. Koppula and Saxena [15] extend conditional random fields (CRFs) with object affordances to predict potential human motions. Wang et al. [23] learn a generative predictor by extending Gaussian Process Dynamical Models (GPDMs) with a latent variable for intention. Hauser [11] utilizes a Gaussian mixture model over task types (e.g., reach, grasp), and predicts both the task type and continuous parameters for that type (e.g., movements) using Gaussian mixture autoregression.

B. Assistance Methods

Many prior works assume the user’s goal is known, and study how methods such as potential fields [2, 6] and motion planning [26] can be utilized to assist for that goal.

For multiple goals, many works follow a predict-then-blend approach of predicting the most likely goal, then assisting for that goal. These methods range from taking over when confident [8, 14], to virtual fixtures to help follow paths [11], to blending with a motion planner [7]. Many of these methods can be thought of as an arbitration between the user’s policy and a fully autonomous policy for the most likely goal [7]. These two policies are blended, where prediction confidence regulates the amount of assistance.

Recently, Hauser [11] presented a system which provides assistance while reasoning about the entire distribution over goals. Given the current distribution, the planner optimizes for a trajectory that minimizes the expected cost, assuming that no further information will be gained. After executing the plan for some time, the distribution is updated by the predictor, and a new plan is generated for the new distribution. In order to efficiently compute the trajectory, it is assumed that the cost function corresponds to squared distance, resulting in the calculation decomposing over goals. In contrast, our model is more general, enabling any cost function for which a value function can be computed. Furthermore, our POMDP model enables us to reason about future human actions.

Planning with human intention models has been used to avoid moving pedestrians. Ziebart et al. [29] use MaxEnt IOC to learn a predictor of pedestrian motion, and use this to predict the probability a location will be occupied at each time step. They build a time-varying cost map, penalizing locations likely to be occupied, and optimize trajectories for this cost. Bandy et al. [4] use fixed models for pedestrian motions, and focus on utilizing a POMDP framework with SARSOP [17] for selecting good actions. Like our approach, this enables them to reason over the entire distribution of potential goals. They show this outperforms utilizing only the maximum likelihood estimate of goal prediction for avoidance.

Outside of robotics, Fern and Tadepalli [22] have studied MDP and POMDP models for assistance. Their study focuses on an interactive assistant which suggest actions to users, who then accept or reject the action. They show that optimal action selection even in this simplified model is PSPACE-complete. However, a simple greedy policy has bounded regret.

Nguyen et al. [20] and Macindoe et al. [19] apply similar models to creating agents in cooperative games, where autonomous agents simultaneously infer human intentions and take assistance actions. Here, the human player and autonomous agent each control separate characters, and thus affect different parts of state space. Like our approach, they model users as stochastically optimizing an MDP, and solve for assistance actions with a POMDP. In contrast to these works, our action space and state space are continuous.

III. PROBLEM STATEMENT

We assume there are a discrete set of possible goals, one of which is the user’s intended goal. The user supplies inputs through some interface to achieve their goal. Our shared autonomy system does not know the intended goal a priori, but utilizes user inputs to infer the goal. It selects actions to minimize the expected cost of achieving the user’s goal.

Formally, let \( x \in X \) be the continuous robot state (e.g., position, velocity), and let \( a \in A \) be the continuous actions (e.g., velocity, torque). We model the robot as a deterministic
dynamical system with transition function $T : X \times A \rightarrow X$. The user supplies continuous inputs $u \in U$ via an interface (e.g. joystick, mouse). These user inputs map to robot actions through a known deterministic function $D : U \rightarrow A$, corresponding to the effect of direct teleoperation.

In our scenario, the user wants to move the robot to one goal in a discrete set of goals $g \in G$. We assume access to a stochastic user policy for each goal $\pi_g(x) = p(u|x, g)$, usually learned from user demonstrations. In our system, we model this policy using the maximum entropy inverse optimal control (MaxEnt IOC) \cite{niepert2016b} framework, which assumes the user is approximately optimizing some cost function for their intended goal $g$. $C^\text{usr}_g : X \times U \rightarrow \mathbb{R}$. This model corresponds to a goal specific Markov Decision Process (MDP), defined by the tuple $(X, U, T, C^\text{usr}_g)$. We discuss details in Sec. IV.

Unlike the user, our system does not know the intended goal. We model this with a Partially Observable Markov Decision Process (POMDP) with uncertainty over the user’s goal. A POMDP maps a distribution over states, known as the belief $b$, to actions. Define the system state $s \in S$ as the robot state augmented by a goal, $s = (x, g)$ and $S = X \times G$. In a slight abuse of notation, we overload our transition function such that $T : S \times A \rightarrow S$, which corresponds to transitioning the robot state as above, but keeping the goal the same.

In our POMDP, we assume the robot state is known, and all uncertainty is over the user’s goal. Observations in our POMDP correspond to user inputs $u \in U$. Given a sequence of user inputs, we infer a distribution over system states (equivalently a distribution over goals) using an observation model $\Omega$. This corresponds to computing $\pi^\text{usr}_g(x)$ for each goal, and applying Bayes’ rule. We provide details in Sec. IV.

The system uses cost function $C^\text{rob} : S \times A \times U \rightarrow \mathbb{R}$, corresponding to the cost of taking robot action $a$ when in system state $s$ and the user has input $u$. Note that allowing the cost to depend on the observation $u$ is non-standard, but important for shared autonomy, as prior works suggest that users prefer maintaining control authority \cite{niepert2016a}. This formulation enables us to penalize robot actions which deviate from $D(u)$. Our shared autonomy POMDP is defined by the tuple $(S, A, T, C^\text{rob}, U, \Omega)$. The optimal solution to this POMDP minimizes the expected accumulated cost $C^\text{rob}$. As this is intractable to compute, we utilize Hindsight Optimization to select actions, described in Sec. V.
confident about goals it already believes are likely. In order to avoid this, we ensure that the prediction probability comes from user inputs only, and not robot actions:

\[ p(\xi|g) = \prod_t \pi_t^{\text{usr}}(u_t|x_t, g) \]

Finally, to compute the probability of a goal given the partial sequence up to \( t \), we use Bayes’ rule:

\[ p(g|x^{0:t}) = \frac{p(x^{0:t}|g)p(g)}{\sum_{g'} p(x^{0:t}|g')p(g')} \]

This corresponds to our POMDP observation model \( \Omega \).

V. Hindsight Optimization

Solving POMDPs, i.e. finding the optimal action for any belief state, is generally intractable. We utilize the QMDP approximation \([13]\), also referred to as hindsight optimization \([5,23]\) to select actions. The idea is to estimate the cost-to-go of the belief by assuming full observability will be obtained at the next time step. The result is a system that never tries to gather information, but can plan efficiently in the deterministic subproblems. This concept has been shown to be effective in other domains \([24,25]\).

We believe this method is suitable for shared autonomy for many reasons. Conceptually, we assume the user will provide inputs at all times, and therefore we gain information without explicit information gathering. In this setting, works in other domains have shown that QMDP performs similarly to methods that consider explicit information gathering \([16]\). Computationally, QMDP is efficient to compute even with continuous state and action spaces, enabling fast reaction to user inputs. Finally, explicit information gathering where the user is treated as an oracle would likely be frustrating \([10,3]\), and this method naturally avoids it.

Let \( Q(b,a,u) \) be the action-value function of the POMDP, estimating the cost-to-go of taking action \( a \) when in belief \( b \) with user input \( u \), and acting optimally thereafter. In our setting, uncertainty is only over goals, \( b(s) = b(g) = p(g|x^{0:t}) \).

Let \( Q_g(x,a,u) \) correspond to the action-value for goal \( g \), estimating the cost-to-go of taking action \( a \) when in state \( x \) with user input \( u \), and acting optimally for goal \( g \) thereafter. The QMDP approximation is \([18]\):

\[ Q(b,a,u) = \sum_g b(g)Q_g(x, a, u) \]

Finally, as we often cannot calculate \( \arg \max_a Q(b, a, u) \) directly, we use a first-order approximation, which leads to us following the gradient of \( Q(b, a, u) \).

We now discuss two methods for approximating \( Q_g \):

1) **Robot and user both act**: Estimate \( u \) with \( \pi^u_{g} \) at each time step, and utilize \( C^{\text{rob}}(\{x,g\},a,u) \) for the cost. Using this cost, we could run Q-learning algorithms to compute \( Q_g \). This would be the standard QMDP approach for our POMDP.

2) **Robot takes over**: Assume the user will stop supplying inputs, and the robot will complete the task. This enables us to use the cost function \( C^{\text{rob}}(s,a,u) = C^{\text{rob}}(s,a,0) \). Unlike the user, we can assume the robot will act optimally. Thus, for many cost functions we can analytically compute the value, e.g. cost of always moving towards the goal at some velocity.

An additional benefit of this method is that it makes no assumptions about the user policy \( \pi^u_{g} \), making it more robust to modelling errors. We use this method in our experiments.

VI. Multi-Goal MDP

There are often multiple ways to achieve a goal. We refer to each of these ways as a target. For a single goal (e.g. object to grasp), let the set of targets (e.g. grasp poses) be \( \kappa \in K \). We assume each target has robot and user cost functions \( C^{\text{rob}}_\kappa \) and \( C^{\text{usr}}_\kappa \), from which we compute the corresponding value and action-value functions \( V^\kappa_\kappa \) and \( Q^\kappa_\kappa \), and soft-value functions \( V^\kappa_\kappa^\infty \) and \( Q^\kappa_\kappa^\infty \). We derive the quantities for goals, \( V_g, Q_g, V^\infty_g, Q^\infty_g \), as functions of these target functions.
A. Multi-Target Assistance

For simplicity of notation, let \( C_g(x, a) = C^{\text{act}}(\{x, g\}, a, 0) \) and \( C_\kappa(x, a) = C^{\text{act}}(x, a, 0) \). We assign the cost of a state-action pair to be the cost for the target with the minimum cost-to-go after this state:

\[
C_g(x, a) = C_{\kappa^*}(x, a) \quad \kappa^* = \arg \min_{\kappa} V_\kappa(x')
\]

Where \( x' \) is the robot state when action \( a \) is applied at \( x \).

**Theorem 1:** Let \( V_\kappa \) be the value function for target \( \kappa \). Define the cost for the goal as above. For an MDP with deterministic transitions, the value and action-value functions \( V_g \) and \( Q_g \) can be computed as:

\[
Q_g(x, a) = C_{\kappa^*}(x, a) + V_{\kappa^*}(x') \quad \kappa^* = \arg \min_{\kappa} V_\kappa(x')
\]

\[
V_g(x) = \min_{\kappa} V_\kappa(x)
\]

**Proof:** We show how the standard value iteration algorithm, computing \( Q_g \) and \( V_g \) backwards, breaks down at each time step. At the final timestep \( T \), we get:

\[
Q_g^T(x, a) = C_g(x, a)
\]

\[
V_g^T(x) = \min_a C_{\kappa^*}(x, a)
\]

Since \( V_g^T(x) = \min_a C_{\kappa^*}(x, a) \) by definition. Now, we show the recursive step:

\[
Q_g^{t-1}(x, a) = C_g(x, a) + V_g^t(x')
\]

\[
= C_{\kappa^*}(x, a) + \min_{\kappa} V_{\kappa^*}(x') \quad \kappa^* = \arg \min_{\kappa} V_{\kappa^*}(x')
\]

\[
V_g^{t-1}(x) = \min_a Q_g^{t-1}(x, a)
\]

\[
= \min_a C_{\kappa^*}(x, a) + \min_{\kappa} V_{\kappa^*}(x') \quad \kappa^* = \arg \min_{\kappa} V_{\kappa^*}(x')
\]

\[
\geq \min_{\kappa} \left( C_g(x, a) + \min_{\kappa} V_{\kappa^*}(x') \right)
\]

\[
= \min_{\kappa} V_g^{t-1}(x)
\]

Additionally, we know that \( V_g(x) \leq \min_\kappa V_\kappa(x) \), since \( V_\kappa(x) \) measures the cost-to-go for a specific target, and the total cost-to-go is bounded by this value for a deterministic system. Therefore, \( V_g(x) = \min_\kappa V_\kappa(x) \).

B. Multi-Target Prediction

Here, we don’t assign the goal cost to be the cost of a single target \( C_g \), but instead use a distribution over targets.

**Theorem 2:** Define the probability of a trajectory and target as \( p(\xi, \kappa) \propto \exp(-C_\kappa(\xi)) \). Let \( V_\kappa^\approx \) and \( Q_\kappa^\approx \) be the soft-value functions for target \( \kappa \). The soft value functions for goal \( g \), \( V_g^\approx \) and \( Q_g^\approx \), can be computed as:

\[
V_g^\approx(x) = \text{softmin}_{\kappa} V_\kappa^\approx(x)
\]

\[
Q_g^\approx(x, u) = \text{softmin}_{\kappa} Q_\kappa^\approx(x, u)
\]

**Proof:** As the cost is additive along the trajectory, we can expand out \( \exp(-C_\kappa(\xi)) \) and marginalize over future inputs to get the probability of an input now:

\[
\pi^\text{usr}(u_t, \kappa|x_t) = \frac{\exp(-C_\kappa(x_t, u_t)) \int \exp(-C_\kappa(\xi^t_{t+1})))}{\int \exp(-C_\kappa(\xi^t_{t+1})))}
\]

Where the integrals are over all trajectories. By definition, \( \exp(-V_\kappa^\approx(x_t)) = \int \exp(-C_\kappa(\xi^t_{t+1}))) \):

\[
\pi^\text{usr}(u_t|x_t) = \frac{\exp(-C_\kappa(x_t, u_t)) \exp(-V_\kappa^\approx(x_t, u_t))}{\sum_{\kappa} \exp(-V_\kappa^\approx(x_t, u_t))}
\]

Marginalizing out \( \kappa \) and simplifying:

\[
\pi^\text{usr}(u_t|x_t) = \frac{\sum_{\kappa} \exp(-Q_\kappa^\approx(x_t, u_t))}{\sum_{\kappa} \exp(-V_\kappa^\approx(x_t, u_t))}
\]

As \( V_\kappa^\approx \) and \( Q_\kappa^\approx \) are defined such that \( \pi^\text{usr}(u|x, g) = \exp(V_g^\approx(x) - Q_g^\approx(x, u)) \), our proof is complete.

VII. USER STUDY

We compare two methods for shared autonomy in a user study: our method, referred to as policy, and a conventional predict-then-blend approach based on Dragan and Srinivas [7], referred to as blend.
Both systems use the same prediction algorithm, based on the formulation described in Sec. [IV] For computational efficiency, we follow Dragan and Srinivasa [1] and use a second order approximation about the optimal trajectory. They show that, assuming a constant Hessian, we can replace the difficult to compute soft-min functions \( V_\kappa \) and \( Q_\kappa \) with the min value and action-value functions \( V_\kappa \) and \( Q_\kappa \).

Our policy approach requires specifying two cost functions, \( C^\text{usr}_\kappa \) and \( C^\text{rob}_\kappa \), from which everything is derived. For \( C^\text{usr}_\kappa \) we use a simple function based on the distance \( d \) between the robot state \( x \) and target \( \kappa \):

\[
C^\text{usr}_\kappa(x, u) = \begin{cases} 
\alpha & d > \delta \\
\frac{\alpha}{d} & d \leq \delta 
\end{cases}
\]

That is, a linear cost near a goal \((d \leq \delta)\), and a constant cost otherwise. This is by no means the best cost function, but it does provide a baseline for performance. We might expect, for example, that incorporating collision avoidance into our cost function may enable better performance [26].

We set \( C^\text{rob}_\kappa(x, a, u) = C^\text{usr}_\kappa(x, u) + (a - D(u))^2 \), penalizing the robot from deviating from the user command while optimizing their cost function.

The predict-then-blend approach of Dragan and Srinivasa requires estimating how confident the predictor is in selecting the most probable goal. This confidence measure controls how autonomy and user input are arbitrated. For this, we use the distance-based measure used in the experiments of Dragan and Srinivasa [27], \( \text{conf} = \max(0,1-d^2/d^2) \), where \( d \) is the distance to the nearest target, and \( D \) is some threshold past which confidence is zero.

A. Hypotheses

Our experiments aim to evaluate the task-completion efficiency and user satisfaction of our system compared to the predict-then-blend approach. Efficiency of the system is measured in two ways: the total execution time, a common measure of efficiency in shared teleoperation [6], and the total user input, a measure of user effort. User satisfaction is assessed through a survey.

This leads to the following hypotheses:

**H1** Participants using the policy method will grasp objects significantly faster than the blend method

**H2** Participants using the policy method will grasp objects with significantly less control input than the blend method

**H3** Participants will agree more strongly on their preference for the policy method compared to the blend method

B. Experiment setup

We recruited 10 participants (9 male, 1 female), all with experience in robotics, but none with prior exposure to our system. To counterbalance individual differences of users, we chose a within-subjects design, where each user used both systems.

We setup our experiments with three objects on a table - a canteen, a block, and a cup. See Fig. 5. Users teleoperated a robot arm using two joysticks on a Razer Hydra system.

The right joystick mapped to the horizontal plane, and the left joystick mapped to the height. A button on the right joystick closed the hand. Each trial consisted of moving from the fixed start pose, shown in Fig. 5, to the target object, and ended once the hand was closed.

At the start of the study, users were told they would be using two different teleoperation systems, referred to as “method1” and “method2”. Users were not provided any information about the methods. Prior to the recorded trials, users went through a training procedure: First, they teleoperated the arm directly, without any assistance or objects in the scene. Second, they grasped each object one time with each system, repeating if they failed the grasp. Third, they were given the option of additional training trials for either system if they wished.

Users then proceeded to the recorded trials. For each system, users picked up each object one time in a random order. Half of the users did all blend trials first, and half did all policy trials first. Users were told they would complete all trials for one system before the system switched, but were not told the order. However, it was obvious immediately after the first trail started, as the policy method assists from the start pose and blend does not. Upon completing all trials for one system, they were told the system would be switching, and then proceeded to complete all trials for the other system. If users failed at grasping (e.g. they knocked the object over), the data was discarded and they repeated that trial. Execution time and total user input were measured for each trial.

Upon completing all trials, users were given a short survey. For each system, they were asked for their agreement on a 1-7 Likert scale for the following statements:

1) “I felt in control”
2) “The robot did what I wanted”
3) “I was able to accomplish the tasks quickly”
Fig. 6. Task completion times and total input for all trials. On the left, means and standard errors for each system. On the right, the time and input of blend minus policy, as a function of the time and total input of blend. Each point corresponds to one trial, and colors correspond to different users. We see that policy was faster and resulted in less input in most trials. Additionally, the difference between systems increases with the time/input of blend.

Fig. 7. On the left, means and standard errors from survey results from our user study. For each system, users were asked if they felt in control, if the robot did what they wanted, if they were able to accomplish tasks quickly, and if they would like to use the system. Additionally, they were asked which system they prefer, where a rating of 1 corresponds to blend, and 7 corresponds to policy. On the right, the like rating of policy minus blend, plotted against the prefer rating. When multiple users mapped to the same coordinate, we plot multiple dots around that coordinate. Colors correspond to different users, where the same user has the same color in Fig. 6.

4) “If I was going to teleoperate a robotic arm, I would like to use the system”

They were also asked “which system do you prefer”, where 1 corresponded to blend, 7 to policy, and 4 to neutral. Finally, they were asked to explain their choices and provide any general comments.

C. Results

Users were able to successfully use both systems. There were a total of two failures while using each system - once each because the user attempted to grasp too early, and once each because the user knocked the object over. These experiments were reset and repeated.

We assess our hypotheses using a significance level of $\alpha = 0.05$, and the Benjamini–Hochberg procedure to control the false discovery rate with multiple hypotheses.

Trial times and total control input were assessed using a two-factor repeated measures ANOVA, using the assistance method and object grasped as factors. Both trial times and total control input had a significant main effect. We found that our policy method resulted in users accomplishing tasks more quickly, supporting H1 ($F(1, 9) = 12.98, p = 0.006$). Similarly, our policy method resulted in users grasping objects with less input, supporting H2 ($F(1, 9) = 7.76, p = 0.021$). See Fig. 6 for more detailed results.

To assess user preference, we performed a Wilcoxon signed-rank test on the survey question asking if they would like to use each system, and a Wilcoxon rank-sum test on the survey question of which system they prefer against the null hypothesis of no preference (value of 4). There was no evidence to support H3.

In fact, our data suggests a trend towards the opposite - that users prefer blend over policy. When asked if they would like to use the system, there was a small difference between methods (Blend: $M = 4.90, SD = 1.58$, Policy: $M = 4.10, SD = 1.64$). However, when asked which system they preferred, users expressed a stronger preference for blend ($M = 2.90, SD = 1.76$). While these results are not statistically significant according to our Wilcoxon tests and $\alpha = 0.05$, it does suggest a trend towards preferring blend. See Fig. 7 for results for all survey questions.

We found this surprising, as prior work indicates a strong correlation between task completion time and user satisfaction, even at the cost of control authority, in both shared autonomy [7, 11] and human-robot teaming [9] settings. Not only were users faster, but they recognized they could accomplish tasks more quickly (see quickly in Fig. 7). One user specifically commented that “(Policy) took more practice to learn. . . but once I learned I was able to do things a little faster. However, I still don’t like feeling it has a mind of it’s own”.

As shown in Fig. 7 users agreed more strongly that they felt in control during blend. Interestingly, when asked if the robot did what they wanted, the difference between methods was less drastic. This suggests that for some users, the robot’s autonomous actions were in-line with their desired motions, even though the user was not in control.

Users also commented that they had to compensate for policy in their inputs. For example, one user stated that “(policy) did things that I was not expecting and resulted in unplanned motion”. This can perhaps be alleviated with user-specific policies, matching the behavior of particular users.

Some users suggested their preferences may change with specific policies, matching the behavior of particular users. 1In prior works where users preferred greater control authority, task completion times were indistinguishable [13].
learning its behavior. Perhaps I would prefer it more strongly with more experience”. It is possible that with more training, or an explanation of how policy works, users would have preferred the policy method. We leave this for future work.

D. Examining trajectories

Users with different preferences had very different strategies for using each system. Some users who preferred the assistance policy changed their strategy to take advantage of the constant assistance towards all goals, applying minimal input to guide the robot to the correct goal (Fig. 9). In contrast, users who preferred blending were often opposing the actions of the autonomous policy (Fig. 9). This suggests the robot was following a strategy different from their own.

VIII. CONCLUSION AND FUTURE WORK

We presented a framework for formulating shared autonomy as a POMDP. Whereas most methods in shared autonomy predict a single goal, then assist for that goal (predict-then-blend), our method assists for the entire distribution of goals, enabling more efficient assistance. We utilized the MaxEnt IOC framework to infer a distribution over goals, and Hindsight Optimization to select assistance actions. We performed a user study to compare our method to a predict-then-blend approach, and found that our system enabled faster task completion with less control input. Despite this, users were mixed in their preference, trending towards preferring the simpler predict-then-blend approach.

We found this surprising, as prior work has indicated that users are willing to give up control authority for increased efficiency in both shared autonomy [7, 11] and human-robot teaming [9] settings. Given this discrepancy, we believe more detailed studies are needed to understand precisely what is causing user dissatisfaction. Our cost function could then be modified to explicitly avoid dissatisfying behavior. Additionally, our study indicates that users with different preferences interact with the system in very different ways. This suggests a need for personalized learning of cost functions for assistance.

Implicit in our model is the assumption that users do not consider assistance when providing inputs - and in particular, that they do not adapt their strategy to the assistance. We hope to alleviate this assumption in both prediction and assistance by extending our model as a stochastic game.

ACKNOWLEDGMENTS

This work was supported in part by NSF GRFP No. DGE-1252522, NSF Grant No. 1227495, the DARPA Autonomous Robotic Manipulation Software Track program, the Okawa Foundation, and an Office of Naval Research Young Investigator Award.
REFERENCES


