Recognition, Prediction, and Planning for Assisted Teleoperation of Freeform Tasks

Kris Hauser
School of Informatics and Computing, Indiana University at Bloomington
hauserk@indiana.edu

***Abstract*—This paper presents a system for improving the intuitiveness and responsiveness of assisted robot teleoperation interfaces by combining intent prediction and motion planning. Two technical contributions are described. First, an intent predictor estimates the user's desired task, and accepts freeform tasks that include both discrete types and continuous parameters (e.g., desired target positions). Second, a cooperative motion planner uses the task estimates to generate continuously updated robot trajectories by solving optimal control problems with time-varying objective functions. The planner is designed to respond interactively to changes in the indicated task, avoid collisions in cluttered environments, and achieve high-quality motions using a hybrid of numerical and sample-based techniques. The system is applied to the problem of controlling a 6D robot manipulator using 2D mouse input in the context of two tasks: static target reaching and dynamic trajectory tracking. Simulations suggest that it enables the robot to reach static targets faster and to track trajectories more closely than comparable techniques.**

I. Introduction

This paper aims to develop novice-friendly interfaces for teleoperating complex robots using commodity input devices (e.g., computer mice). Safe, intuitive, inexpensive interfaces could bring robotics in touch with a broad user base and lead to new applications in teleoperated household tasks, remote medicine, and space exploration. But a major challenge is the extreme *asymmetry* between input devices and robots: the robot has many more degres of freedom than a user can control at once. Modal interfaces provide users control over subsets of degrees of freedom at once (e.g., [7]), but mode switching makes seemingly simple tasks quite tedious. Moreover, human input is noisy and error-prone, and hence a robot should filter out unsafe commands. Hence, novice-friendly teleoperation systems will need to embed a great deal of intelligence in order to act fluidly, capably, and safely.

One promising approach is to *understand the user's intent* so that the teleoperation system can provide task-appropriate forms of assistance [1, 11, 16, 17, 19]. In this approach, the robot estimates the operator's intent using the raw input signals and then chooses its action in order to best achieve the desired task. This strategy is appealing because it decouples the overall problem into two subproblems — intent estimation and planning — and can also improve performance in the presence of time delays because the robot can anticipate the desired task in the midst of a partially-issued command [5]. In this paper we apply this approach to *freeform tasks*, such as reaching, pointing, or tracking a trajectory, in which the intent ranges over a continuous infinity of possibilities. Freeform tasks broaden the capabilities of intelligent teleoperated robots because they provide an operator with greater flexibility in unstructured scenarios. But they are more challenging because the robot must not only estimate the task but also several continuous parameters. Moreover it is no longer practical to precompute task-specific motion strategies for the robot, and instead the robot must optimize motions on-the-fly.

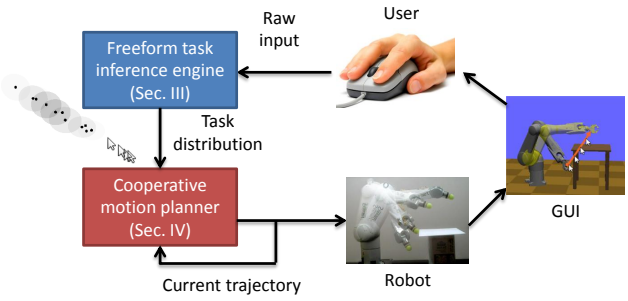

Fig. 1. The system integrates new contributions for prediction of intended tasks and real-time optimal control.

This paper presents new techniques for understanding freeform tasks as well as planning high quality motions to achieve them (Figure 1). We introduce a Bayesian intent inference engine based on a Gaussian Mixture Autoregression framework that deduces a task type and parameters using statistical features of the input time series. Using this method we learn models for estimating static targets and dynamic trajectories in a unified framework, and we predict intended static and dynamic targets with 43% and 13% lower error, respectively, than using the cursor position alone.

Our system is also enables the extrapolation of intended *future time-varying goals*. Our second contribution exploits this capability to improve trajectory tracking *by anticipating future movements*. We present a novel cooperative motion planner that optimizes the robot's trajectory to match the forecasted one, while also handling highly cluttered environments. A combination of sample-based planning [14] and numerical trajectory optimization techniques [4] are used to achieve responsive operation, and to produce high-quality, collision-free paths. The integrated system achieves fluid and real-time operation by interleaving inference, planning, and execution while streaming in user input. Experiments on a simulated 6DOF robot suggest that it improves tracking performance by up to 67% compared to systems that either do not perform

inference, or that use simpler planners.

II. Related Work

Intent and activity recognition for robot teleoperation has been studied in the context of telemanipulation and surgical assist systems [1, 11, 16, 17, 19]. Past work has largely made use of the well-established machinery of Hidden Markov Models, and as a result is limited to a finite number of discrete tasks, such as navigating to a handful of target locations or picking and placing objects in a constrained manner. By contrast, our system learns and estimates models with continuous parameters which makes it applicable to freeform tasks.

Several methods have been proposed for predictive modeling of cursor targets to help select icons in graphical user interfaces, including linear regression from peak velocity [2], directional characteristics [12], kinematic motion models [13], and inverse optimal control techniques [18]. Unlike [12, 18] our work is applicable to a fully continuous set of targets. It also achieves better prediction accuracy compared to linear models [2, 13] because the Gaussian Mixture Autoregression technique used here is capable of modeling nonlinear characteristics that are observed in user behavior.

Sample-based motion planners such as Probabilistic Roadmaps (PRMs) and Rapidly-Exploring Random Trees (RRTs) are successful at planning collision-free motion for high-dimensional robot systems [14], and have also been successfully applied to hard real-time planing for 2D helicopters [6] and ground vehicles [15]. Recently they have been applied to teleoperation interfaces for robot manipulator arms [8]. But these works have traditionally focused on finding feasible paths rather than optimal ones. Sample-based approaches have been recently developed for the optimal motion planning problem [10], but they have not yet been applied to time-varying cost functions and moreover converge too slowly for real-time use. An alternative approach is numerical optimization over a trajectory parameterization [4]. Optimization approaches can achieve optimality with a fast convergence rate, albeit only locally. Our hybrid planner combines the strength of sample-based and numerical approaches and is designed specifically to produce high-quality paths quickly for a broad class of cost functions.

III. Problem Overview

Our system implements an intelligent robot interface for controlling robots with simple and/or noisy input devices. We consider the setting of a user operating a mouse or other 2D input device to control a 6DOF industrial robot arm in a cluttered, known 3D environment. The user issues commands through a GUI that displays the robot and its environment, and can translate, rotate, and zoom the camera freely.

The operator uses a basic "click-and-drag" operation in which he/she clicks on a point to move and then drags it in the manner in which he/she wishes for it to move. This "manner" is stated intentionally vaguely so that the robot's motion is underspecified. For example, a user may wish to reach a target as quickly as possible (e.g., to grasp and move an object), or to perform a gesture, such as a wave hello or to indicate direction for a coworker. Another possibility might use the underspecification to achieve depth control, or actions upon specific objects in the environment. While executing a task, a robot may exploit underspecification to avoid obstacles or to optimize other criteria, such as energy consumption.

We use the notion of *task* to act an intermediary between user input and robot motion. We define a task as *a sufficient representation of the optimality criterion* for a robot to complete an action primitive, such as reaching a target, picking up an object, or pressing a button. In this paper we focus on *freeform tasks*, which include both a discrete task type $i_t \in 1, \ldots, M$ and continuous task parameters $z_t \in \mathbb{R}^N$. (Note that N depends on the type.) Here we consider $M = 2$ task types:

1) *Reach tasks*. Four task parameters include the position of the goal relative to the cursor (g_x, g_y), size g_r, and "urgency" u which is approximated as the average mouse velocity before reaching the goal.
2) *Trajectory following*. Six task parameters include the goal position relative to the cursor (g_x, g_y), its velocity $(\dot{g}_x, \dot{g}_y)$, and its acceleration $(\ddot{g}_x, \ddot{g}_y)$.

The system (Figure 1) is composed of two major components: the Freeform Task Inference Engine (FTIE) (Sec. IV) and the Cooperative Motion Planner (CMP) (Sec. V). The TIE attempts to recover the intended task from the manner in which the mouse is dragged, and infers a probabilistic distribution over tasks. The CMP uses the estimated task and *forecasted evolution of the task in the future* to improve trajectory tracking. The following sections cover new technical contributions in each subsystem that improve performance relative to prior approaches.

IV. Freeform Task Inference Engine

The FTIE estimates a time series of unobserved task parameters (i_t, z_t) from a streaming series of observations o_t, $t = 1, 2, \ldots$, obtained from the raw cursor movement. It uses a hybrid discrete/continuous Dynamic Bayesian Network (DBN) to infer a distribution over freeform task variables. We implement the DBN using a Gaussian mixture autoregression (GMA) technique, which uses Gaussian mixture models to learn and represent complex, multi-modal transition models. It is similar in spirit to the interacting multiple model (IMM) algorithm that extends the Kalman filter to handle multiple switching linear processes [3], but in our case the process models are more complex. Experiments indicate that GMA has superior performance to simpler models, such as linear regressions and Kalman filters.

A. Overview

Given observations o_t of cursor velocities $o_t = (\Delta c_{x,t}, \Delta c_{y,t})$, we wish to infer the distribution over the task parameters i_t and z_t. We model the evolution of the parameters as a DBN, which is autoregressive because we include the observation history $h_t = (o_{t-1}, \ldots, o_{t-k})$ for the prior k time steps as part of the state variable $x_t = (i_t, z_t, h_t)$.

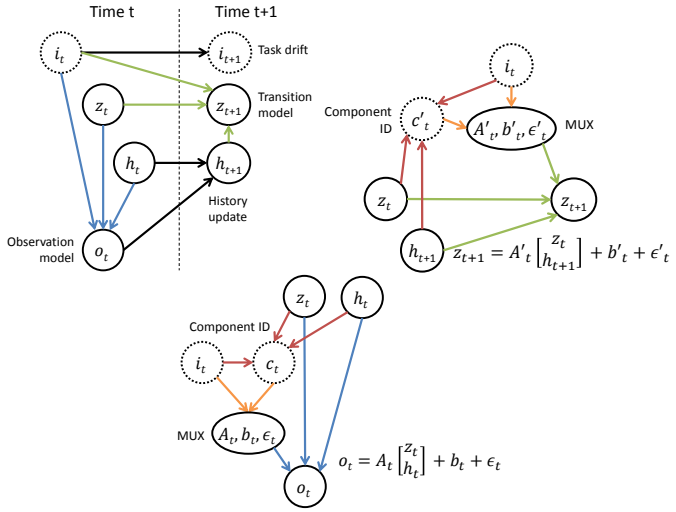

Fig. 2. Top left: the dynamic Bayesian network (DBN) denoting the temporal relationships between hidden task type i_t, hidden task parameters z_t, history h_t, and observation o_t. Top right: the transition model represented as a hybrid graphical model with an unknown component c'_t that indexes into a set of linear process models. Bottom: the observation model. Dashed circles indicate discrete variables while solid circles indicate continuous ones.

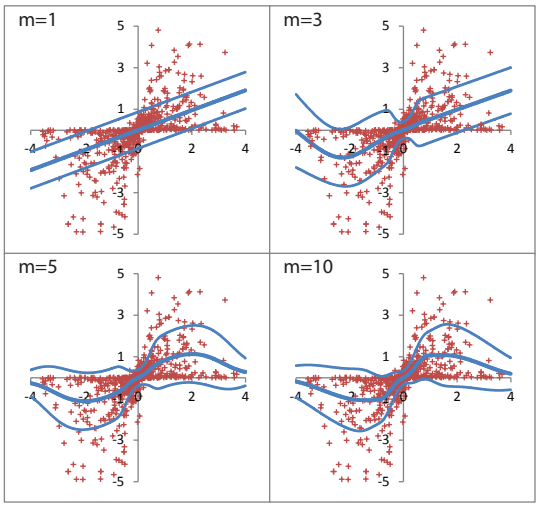

Fig. 3. Gaussian mixture regressions can model complex nonlinear conditional distributions. This data comes from the reach task training set, with the horizontal target position on the x axis and the horizontal cursor velocity on the y axis (both normalized to unit variance). Curves indicate the mean and standard deviation of the relationship $y(x)$ estimated by GMRs with a varying number m of components.

The observation is assumed to be generated according to the probabilistic observation model $P(o_t|x_t)$, while the state evolves according to the transition model $P(x_{t+1}|x_t, o_t)$. Note that in traditional HMMs the transition model is typically not dependent on o_t; this only requires a minor adjustment to the inference procedure. Our model is shown in Figure 2.

As usual in Bayesian filtering, the belief over x_{t+1} is derived from the prior belief over x_t and the observation o_{t+1} in a recursive manner. Specifically, we maintain a belief $b_t(x_t) = P(x_t|o_1, \ldots, o_t)$ and update it using the recursive filtering equation:

$$\begin{aligned} b_{t+1}(x_{t+1}) &= P(x_{t+1}|o_1, \ldots, o_{t+1}) \\ &= \int_{x_t} P(x_{t+1}|x_t, o_{t+1}) P(x_t|o_1, \ldots, o_t) dx_t \\ &= \int_{x_t} P(x_{t+1}|x_t, o_{t+1}) b_t(x_t) dx_t. \end{aligned} \quad (1)$$

This procedure is performed in in two steps:

1) The *predict* step, which computes the unconditioned belief over $b'_{t+1}(x_{t+1})$ independently of the new observation:

$$b'_{t+1}(x_{t+1}) = \int_{x_t} P(x_{t+1}|x_t, o_t) b_t(x_t) dx_t. \quad (2)$$

2) The *update* step, which conditions $b'_{t+1}(x_{t+1})$ on the observation

$$b_{t+1}(x_{t+1}) \propto P(o_{t+1}|x_{t+1}) b'_{t+1}(x_{t+1}). \quad (3)$$

The following sections will describe the GMA implementation of the filter.

B. Gaussian Mixture Regression

First we will provide some preliminaries on Gaussian mixture regression. A Gaussian mixture model (GMM) with m components describes a weighted combination of m Gaussian distributions. If X is distributed with respect to a GMM, the probability that $X = x$ is given by

$$\begin{aligned} P(x) &= GMM(x; w_1, \ldots, w_m, \mu_1, \ldots, \mu_m, \Sigma_1, \ldots, \Sigma_m) \\ &= \sum_{c=1}^{m} w_c \mathcal{N}(x; \mu_c, \Sigma_c). \end{aligned} \quad (4)$$

where $\mathcal{N}(x; \mu, \Sigma)$ denotes the probability that a normally distributed variable $X \sim \mathcal{N}(\mu, \Sigma)$ takes on the value x. The values $w_1, \ldots, w_m$ are component weights that sum to one.

GMMs can be interpreted as introducing an auxiliary hidden *class variable* C taking values in $\{1, \ldots, m\}$ that identifies the component of the GMM from which x is generated. The distribution over x conditional on $C = c$ is $\mathcal{N}(x; \mu_c, \Sigma_c)$, while the prior distribution over C is given by the weights $P(c) = w_c$.

GMMs can be applied to nonlinear regression tasks under an operation known as a *Gaussian mixture regression*, or GMR. It is based on the application of Gaussian conditioning (see Appendix) to each component in the GMM, and reweighting components according to the probability of observing the independent variable. Suppose X and Y are jointly distributed according to a GMM with m components. Given the value of x, $P(y|x)$ is a GMM with weights

$$w_{c|x} = \frac{1}{Z} w_c P(x|c) \quad (5)$$

where Z is a normalization factor and

$$P(x|c) = \mathcal{N}(x; \mu_{c,x}, \Sigma_{c,x}) \quad (6)$$

is the marginal probability that x is drawn from the c'th component. Each component in $P(y|x)$ has a mean μ_c determined by a linear fit $\mu_c = A_c x + b_c$ with a constant covariance Σ_c. The parameters A_c, b_c, and Σ_c are determined in a straightforward manner from the joint GMM using the Gaussian conditioning equation (22). The resulting regression model is:

$$GMR(y|x) = \frac{1}{Z}\sum_{c=1}^{m} w_c \mathcal{N}(x; \mu_{c,x}, \Sigma_{c,x})\mathcal{N}(y; A_c x + b, \Sigma_c). \tag{7}$$

Figure 3 shows that GMRs can model nonlinearities and nonuniform variance in data better than linear regression (which is equivalent to a GMR with $m = 1$). These models are fitted to a 2D projection of our cursor reaching dataset.

C. Transition and Observation Modeling

We use GMRs in both the observation model $P(o_t|i_t, z_t, h_t)$ and transition model $P(x_{t+1}|x_t, o_t)$ (Figure 2). In the observation model, a GMR is estimated for each task type, each of which has independent variables z_t and h_t.

The transition model is factored into three parts: 1) the task type drift, 2) the deterministic history update, and 3) the z transition model as follows:

$$P(x_{t+1}|x_t, o_t) = P(i_{t+1}|i_t)P(h_{t+1}|h_t, o_t)P(z_{t+1}|i_t, z_t, h_{t+1}). \tag{8}$$

The type drift is given by a transition matrix, and in our implementation we simply use a uniform probability of switching $P(i_{t+1} = i_t) = 1 - p$, and $P(i_{t+1} \neq i_t) = p/(M-1)$. The history update simply shifts o_t into the first position of h_t and drops the oldest observation. Finally, $P(z_{t+1}|i_t, z_t, h_{t+1})$ is encoded as a GMR specific to the task type i_t, with the independent variables z_t and h_{t+1}.

D. GMM Filtering and Forecasting

Given GMM belief representations and GMR transition and observation models, the filtering equation (1) has an exact closed form. For computational efficiency we represent $b_t(x_t)$ in factored form as a distribution over task types $P(i_t)$, the history h_t (which is deterministic), and a set of type-conditioned GMMs $b_{z_t|i}$, $i = 1, \ldots, m$ each denoting $P(z_t|i_t = i, o_1, \ldots, o_t)$.

Predict. The predict step evaluates (8) in the context of (2). The history and task type are updated directly as usual, while the task parameters are updated via the propagation of each $b_{z_t|i}$ through the transition GMR. Suppose the transition GMR is given by (7), and $b_{z_t|i}$ has n components: $b_{z_t|i}(z_t) = GMM(z_t; w'_1, \ldots, w'_n, \mu'_1, \ldots, \mu'_n, \Sigma'_1, \ldots, \Sigma'_n)$. It is not hard to show that the distribution of z_{t+1} is a GMM with mn components, given by:

$$b_{z_{t+1}|i}(z_{t+1}) = \frac{1}{Z}\sum_{c=1}^{m}\sum_{d=1}^{n} w_{cd}\mathcal{N}(z_{t+1}; A_c\mu'_d + b, \Sigma_c + A_c\Sigma'_d A_c^T), \tag{9}$$

with

$$w_{cd} = w_c w'_d \mathcal{N}(\mu'_d; \mu_{c,x}, \Sigma_{c,x} + \Sigma'_d) \tag{10}$$

being the probability that the d'th component of $b_{z_t,i}$ is observed in the c'th component of the transition GMR. The last term in this product is the probability that two variables x_1 and x_2, distributed respectively according to $\mathcal{N}(\mu'_d, \Sigma'_d)$ and $\mathcal{N}(\mu_{c,x}, \Sigma_{c,x})$, are equal.

Update. The update step applies the GMR observation model (7) to the update equation (3) via the following derivation. Let the predicted, unconditioned belief $b'_{t+1}(x_{t+1})$ be a GMM with weights w_c, means μ_c, and covariances Σ_c, with $c = 1, \ldots, m$. We also have $P(o_t|x_t)$ as GMR with weights w'_d, x-components $\mathcal{N}(\mu'_{d,x}, \Sigma'_{d,x})$, and linear fits A_d, b_d, $\Sigma'_{d,o}$ with $d = 1, \ldots, n$. We perform a Kalman observation update (23) conditional on each pair of components (c, d) from each of the state and observation models (see Appendix) to determine $P(x_{t+1}|o_{t+1}, c, d)$. Unconditional on the components, we have the update equation:

$$b_{t+1}(x_{t+1}) = \frac{1}{Z}\sum_{c=1}^{m}\sum_{d=1}^{n} w_{cd} P(x_{t+1}|o_{t+1}, c, d) \tag{11}$$

where Z is a normalization term and the weighs w_{cd} indicate the probability that the observation was generated by the c'th component of the state prior and the d'th component of the observation GMR:

$$w_{cd} = w_c w_d \mathcal{N}(o_{t+1}; A_d\mu_c + b_d, \Sigma_{d,o} + A_d\Sigma_c A_d^T). \tag{12}$$

Mixture collapse. Although these equations are exact and polynomial-time computable, their direct application would lead to exponential growth of the number of components in the belief state over time. Hence it is necessary to frequently collapse the belief state representation into a more manageable number of components. We collapse GMMs with $n > k$ components into a constant number k of components after both the predict and update steps ($k = 10$ is used in our experiments). The collapse operation begins by sampling k components $c_1, \ldots, c_k$ without replacement proportionally to their weights. The n original components are partitioned into k subsets $S_1, \ldots, S_k$ by assigning component d to subset i if i is the index for which the KL divergence between $\mathcal{N}(\mu_d, \Sigma_d)$ and $\mathcal{N}(\mu_{c_i}, \Sigma_{c_i})$ is minimized. The output GMM contains one component for each subset S_i, with weight w'_i, mean μ'_i, and variance Σ'_i matched to the subset using the method of moments:

$$\begin{aligned} w'_i &= \frac{1}{Z}\sum_{j\in S_i} w_j \qquad \mu'_i = \frac{1}{Zw'_i}\sum_{j\in S_i} w_j\mu_j \\ \Sigma'_i &= \frac{1}{Zw'_i}\sum_{j\in S_i} w_j[(\mu_j - \mu'_i)(\mu_j - \mu'_i)^T + \Sigma_j]. \end{aligned} \tag{13}$$

Efficient forecasting. Forecasting of future z_t is performed via repeated application of the predict step, without an associated observation update. However our experiments determined that repeated propagation and collapsing for distant forecasts is too expensive for real time use. So, our method only propagates a few steps into the future (5 in our implementation) and for the remaining steps collapses the transition GMR into a single linear Gaussian process $x_{t+1} = Ax_t + b + \epsilon$

TABLE I

MEAN SQUARED TARGET PREDICTION ERRORS OF SEVERAL TECHNIQUES, NORMALIZED TO CURSOR POSITION ERROR. BEST IN COLUMN IS BOLDED.

	Reach	Tracking	Tracking forecast (1s)
Cursor Only	100%	100%	100%
Linear Reg.	78%	111%	98%
Linear Reg. (forecast)	—	—	84%
Kalman Filter	71%	99%	109%
Velocity Extrapolation	—	—	203%
GMA (reach only)	**57%**	151%	86%
GMA (track only)	112%	**87%**	**60%**
GMA (reach+track)	83%	108%	73%

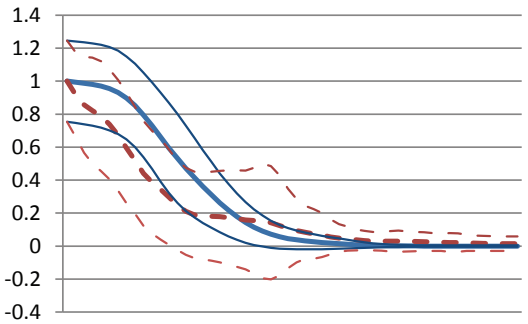

Fig. 4. Normalized and time-scaled distance-to-target in static target reaching tasks for the cursor position (solid lines) and the GMA estimated target (dotted lines). Mean and standard deviation are plotted.

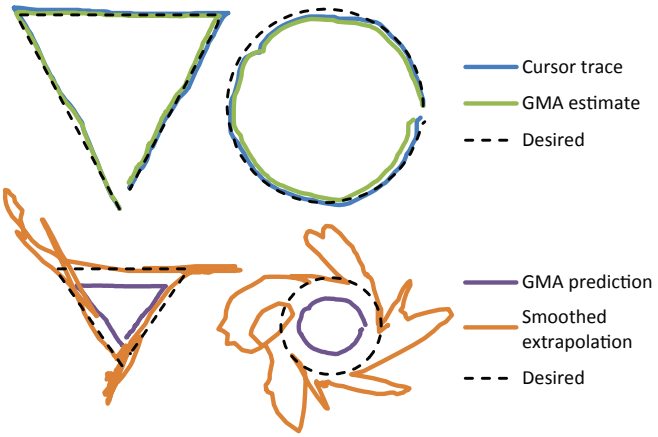

Fig. 5. Tracking two test trajectories. Top row: target estimates produced by the trajectory tracking model are slightly more accurate than the cursor itself. Bottom row: GMA target forecasts 1 s in the future are substantially more accurate than using the cursor position alone (19% lower MSE on both figures), or velocity extrapolation (42% and 14% lower MSE on the triangle and circle, respectively).

linearized around the estimated state distribution. The n'th forecasted state is then computed easily in $O(\log_2 n)$ matrix multiplications through repeated squaring. Memoization is another effective approach if the forecasting horizon is fixed.

E. Gathering Training Data

To acquire training data we constructed a GUI for each task type that instructs users to execute an explicitly given task by clicking and dragging an on-screen widget. In the reach GUI, users are asked to drag a widget to circular targets with center and radius chosen randomly from a uniform range. In the trajectory-tracking GUI, users are asked to drag a widget at the same pace as a reference widget that moved along a trajectory. We pick from triangular, rectangular, circular, and figure-eight patterns that were randomly stretched and compressed in the x and y directions and rotated at an arbitrary angle. The speed of the reference widget was also chosen at random.

These GUIs gathered mouse movements and task parameters at 50Hz from five volunteers resulting in over 830 trials. Trials were mirrored in x and y directions to reduce the effects of asymmetric training data. We noticed that trackpads and mice produce very different cursor movements, so for consistency we gathered all data using a trackpad. The GMM transition and observation models were learned using the standard Expectation-Maximization (EM) algorithm. The history length and number of components were chosen through five-fold cross-validation based model selection, and learning was completed in approximately 10 hours on a 2.8 GHz PC.

F. Experiments

We find that the resulting task models infer desired goals in qualitatively different ways. The reach model predicts the goal location essentially as an extrapolation of the cursor velocity, with greater variance in the direction of travel. The trajectory following model essentially performs smoothing, and evens out jerkiness in the cursor motion.

Table I demonstrates that GMA dramatically reduces MSE of reach tasks compared to simply using the cursor position. It also performs better than simpler techniques of a linear regression and Kalman filter, both fit to the last 5 cursor velocities. Figure 4 plots the distance-to-target along all test examples for the cursor and the GMA prediction. It drops sharply as GMA extrapolates from the current mouse velocity. There is a curious jump in prediction variance once the cursor reaches approximately 20% of the original distance. It appears that this may be an artifact of the input device: in some of our training examples, users made an initial coarse motion toward the target, paused and possibly lifted their finger off of the trackpad, and then approached the target with a fine-grained motion. The pause causes GMA to be significantly thrown off, if only temporarily.

For trajectory tracking, GMA only estimates the current goal position with 13% lower MSE than simply using the cursor position. But its main strength is its ability to anticipate future target positions. GMA reduces the error in forecasts at 1 s by 40%, which is a significant improvement on Kalman filtering or velocity extrapolation. It also performs better than a linear regression trained specifically to perform 1 s forecasts.

This data also indicates that when GMA does not know the task type (reach+track row), its performance decreases as compared to when it is given perfect type information (reach only and track only rows). This suggests that better task classification accuracy would yield significant performance benefits. We leave this issue for future work.

V. Cooperative Motion Planner

The second component of our system is a cooperative motion planner that accepts predictive task estimates and avoids collision. As the task inference engine updates the task estimate in real-time, the trajectory is adjusted by replanning. Our planner incorporates several contributions to make it appropriate for real-time user control of robot arms. It handles time-varying probabilistic distributions over tasks as well as hard constraints like collision avoidance and actuator limits. A hybrid sampling-based and optimization-based scheme is used to quickly generate high-quality motions while also avoiding the local minima problems of optimization approaches. It also is implemented in a hard real-time framework that tolerates planning and communication delays [8].

A. Overview

The robot's motion is subject to joint, velocity, and acceleration limits:

$$\begin{aligned} q_{min} \leq q \leq q_{max} \\ |\dot{q}| \leq \dot{q}_{max} \\ |\ddot{q}| \leq \ddot{q}_{max} \end{aligned} \tag{14}$$

where all inequalities are taken element-wise. A trajectory $q(t) : [t_1, t_2]$ is said to be *dynamically feasible* if each of these constraints is met for all t. Collision constraints furthermore limit the set of valid configurations to the collision-free subset of the configuration space $q \in \mathcal{F} \subseteq \mathcal{C}$.

When the user clicks and drags a point on the robot, the motion defines a time-varying potential field $P(x(q), t)$ where $x(q)$ is the image-space position of the clicked point at the robot's configuration q and $P(x, t)$ is a screen-coordinate target distribution from the inferred task $b_t(z_t)$. In this section we take $t = 0$ to be the current time, and estimate the distribution $P(x, t)$ using the FTIE for all t from 0 to some maximum forecasting horizon.

At each time step the planner searches for a dynamically feasible, collision-free trajectory $q(t) : [0, T] \rightarrow \mathcal{F}$ that optimizes a cost functional of the form

$$J = \int_0^T \mathcal{L}(q(t), t)dt + \Phi(q(T), T) \tag{15}$$

where $\mathcal{L}$ is an incremental cost and Φ is a terminal cost. Rather than optimizing to convergence, the planner stops when it finds a trajectory with lower cost than the robot's current trajectory. If planning is successful then the new trajectory is sent to the robot. To guarantee safety, the trajectory is ensured to be completely collision free and to terminate in velocity zero. If planning fails, the planner simply begins anew on the next step. Before turning to the specification of the cost function in the teleoperation setting we will first describe the planner, which is more general purpose.

B. Hybrid Sample-Based/Numerical Planning

Our hybrid planner grows a tree of states from the start state forward in time in the configuration/velocity/time space $\mathcal{C} \times \dot{\mathcal{C}} \times \mathbb{R}^+$. Like the RRT planner, the tree is grown by sampling a configuration q_d at random and picking an existing node to extend a new edge toward q_d. Here, each edge is a relatively short collision-free trajectory, and we utilize a *steering function* to ensure that each extension is dynamically feasible and terminates at q_d with zero velocity (a similar strategy was used in [6]). In our case, the steering function constructs a time-optimal acceleration-bounded curve using the method of [9].

Interleaved with random RRT-like expansion, our method also performs numerical optimization to construct trajectory extensions that locally optimize (15). The integral in (15) is evaluated using Gaussian quadrature, and the optimization parameters are the target configuration of the steering function $q_d \in \mathcal{C}$ as well as a time scale $\alpha \geq 1$ that extends the time at which q_d is reached. This parameter helps the planner follow slowly time-varying objectives more closely. A boundary constrained quasi-Newton optimization is run for a handful of iterations (10 in our implementation).

Several performance enhancements are used in our planner.

1) We initially seed the tree with the trajectory computed on the prior iteration and attempt to make improvements through local optimization. This approach helps the robot cope better with suboptimal paths caused by terminating queries early because subsequent iterations are likely to improve the path further.
2) Following [6] we produce more fluid paths by bisecting each edge in the tree to produce intermediate states with nonzero velocity.
3) To expand the tree toward the the random state q_d, we choose the closest state under the pseudometric that measures the optimal time to reach q_d in the absence of obstacles. The combination of this metric and the prior bisection technique makes the planner less likely to find inefficient paths that repeatedly start and stop.
4) We prune branches of the tree that have greater incremental cost than the current best path found so far.
5) Lazy collision checking is used to delay expensive edge feasibility checks until the planner finds a path that improves the cost.
6) We devote 50% of each time step to trajectory smoothing using a shortcutting heuristic described in [9], with some minor modifications to ensure that each shortcut makes an improvement in the time-varying cost function.

To obey the hard real-time constraint, the planner should not be initialized at the current state of the robot because once planning is completed, the robot will have already moved. Instead, following [6, 15] the planner is initialized to the predicted state along the current trajectory, propagated forward in time by the planner's time limit Δt. As in [8] we adapt the time step Δt to the difficulty of problems by increasing it if the planner failed on the prior time step (indicating a hard plan), or reducing it if the planner succeeded on the prior time step (indicating an easy plan). To improve responsiveness we also reduce Δt when the user changes goals by introducing a scaling factor e^{-cD} where D is the distance between the

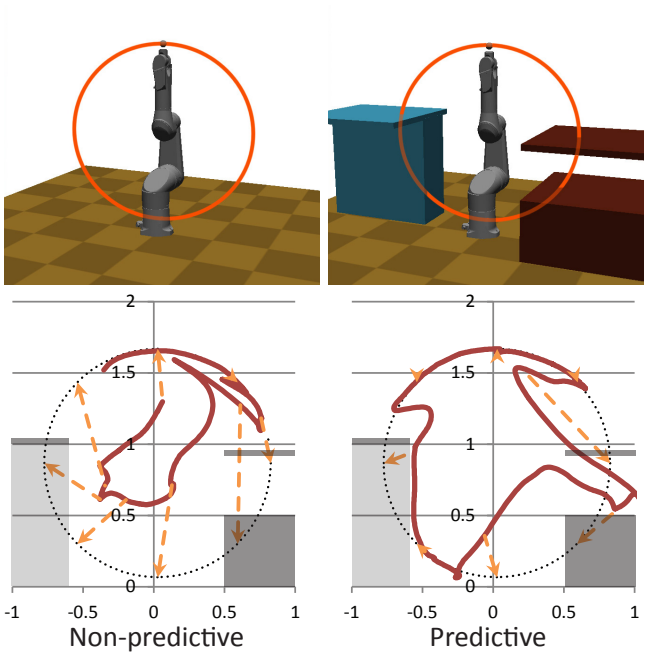

Fig. 6. Top: Uncluttered and cluttered test scenarios in which the end-effector is instructed to move along a circular trajectory. The trajectory is a priori unknown to the robot. Bottom: Traces of the end effector in the cluttered environment without (left) and with (right) predictive tracking. Because it is planning and executing in real-time, once the non-predictive planner finishes planning the target has already moved a great distance. Predictive tracking anticipates the target's movements and leads to substantially lower error.

cursor position on the prior planning iteration and the current iteration. Here c is a sensitivity parameter chosen via a small amount of tuning.

C. Expected Cost Functionals and Introspective Discounting

This section converts the predictive task probability distribution $P(x(q), t)$ to a cost functional of proper deterministic form (15). We also introduce an *introspective cost functional* that monitors both the task distribution and the likelihood of a successful replan. Let g denote a realization of the user-defined, time-dependent objective distributed according to the current belief. We optimize the expected cost:

$$J = E_g \left[\int_0^T \mathcal{L}_d(q(t), t|g(t))dt + \Phi_d(q(T), T|g) \right]. \quad (16)$$

By linearity of expectation, we have

$$J = \int_0^T E_{g(t)} \left[\mathcal{L}_d(q(t), t|g(t)) \right] dt + E_g \left[\Phi_d(q(T), T|g) \right]. \quad (17)$$

Now let the deterministic cost functionals $\mathcal{L}_d$ and Φ_d measure squared distance to the goal $g(t)$. Specifically, reach and tracking tasks require that the robot's end effector position projected to the screen $x(t) \equiv x(q(t))$ reaches the goal $g(t)$:

$$J = \int_0^T \lambda(t) E_g \left[||x(t) - g(t)||^2 \right] dt + \int_T^\infty \lambda(t) E_g \left[||x(T) - g(t)||^2 \right] \quad (18)$$

TABLE II
MEAN TARGETING ERRORS FOR VARIOUS TEST TRAJECTORIES, IN PIXELS. RESULTS AVERAGED OVER 20 RUNS.

	Reach	Tracking	Circle	Circle+Obst
IK+S, cursor only	88	87	56	141
CMP, cursor only	97	87	39	78
CMP+FTIE	**72**	62	**15**	75
CMP+FTIE+type	**72**	**59**	18	**63**

where $\lambda(t)$ is a discount factor. Using the fact that $E[||X||^2] = tr(Var[X]) + ||E[X]||^2$, this expression simplifies to

$$J = \int_0^\infty tr(Var[g(t)])dt + \int_0^T \lambda(t)||x(t) - E[g(t)]||^2 dt + \int_T^\infty \lambda(t)||x(T) - E[g(t)]||^2 dt. \quad (19)$$

Since the first term is independent of x it can be dropped from the optimization, resulting in an expression in the form of (15) as desired.

The introspective discount function weighs the contribution of each point in time to reflect its expected cost *taking into account the fact that the path will be replanned in the future*:

$$\lambda(t) = E_{x_t} \left[\frac{||X_t(t) - E[g(t)]||^2}{||x(t) - E[g(t)]||^2} \right] \quad (20)$$

where X_t indicates the unknown trajectory actually executed by the robot, taking future replans into account. We estimate this expectation by gathering planning statistics on prior problems. We find that for a given problem, an exponentially decreasing fit e^{-bt} provides a good fit to the empirical data. However, the rate parameter b is highly sensitive to the difficulty of the problem. Fortunately the current time step Δ provides a first order approximation of problem difficulty. So, we adaptively discount less in harder regions of the configuration space by scaling the t axis of λ proportionally to Δ and estimate a single b that provides the best fit to our training data.

D. Experiments

Table II lists mean distance from the robot's end effector to the intended target for several planners and scenarios. The Reach and Tracking columns use the natural reach and tracking motions from our testing set as user input. The Circle column indicates a synthetic circle tracking task, while Circle+Obst introduces clutter into the environment (Figure 6).

For comparison we tested an inverse kinematics controller with a safety filter (IK+S), which prevents infeasible motions by simple rejection. We also tested a non-predictive "cursor only" implementation that simply optimizes the end effector's distance to the cursor at every step. CMP improves upon IK+S by 47% in clutter because it uses trajectory optimization to circumvent obstacles. Furthermore, FTIE improves upon the cursor-only approach by 27% on the natural cursor trajectories and 23% on the circular trajectories.

We also examined whether FTIE discriminates well between tasks, and found out that classification performance is in fact

quite poor; the most likely task estimated by GMA is correct only 66% of the time. Surprisingly, this does not have a major effect on overall performance! The data in the last row of Table II suggest that even if perfect task knowledge is introduced into the estimation, targeting error does not change significantly.

VI. Conclusion

This paper presented a system for estimating, predicting, and planning freeform tasks for assisted teleoperation of a 6DOF robot manipulator using 2D cursor input. Our contributions are twofold. First, a freeform intent inference technique based on Gaussian mixture autoregression (GMA) was used to predict static targets, dynamic targets, and to distinguish between the two. Second, a cooperative motion planner was used generate higher quality trajectories by anticipating users' desired tasks. Results in simulation show improved task performance, and suggest that better task discrimination may yield even further benefits. We are interested in extending our work to handle freeform tasks that are contextualized to the robot's environment, and to support object manipulation tasks. Finally, in the near future we intend to study the subjective experience of novice users using our system to control a real robot.

Appendix

Gaussian Conditioning. If X and Y are jointly normally distributed as follows:

$$\begin{bmatrix} X \\ Y \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}, \begin{bmatrix} \Sigma_x & \Sigma_{xy} \\ \Sigma_{yx} & \Sigma_y \end{bmatrix} \right), \tag{21}$$

then the conditional distribution over Y given the value of X is another Gaussian distribution $\mathcal{N}(\mu_{y|x}, \Sigma_{y|x})$ with

$$\begin{aligned} \mu_{y|x} &= \mu_y + \Sigma_{yx}\Sigma_x^{-1}(x - \mu_x) \\ \Sigma_{y|x} &= \Sigma_y - \Sigma_{yx}\Sigma_x^{-1}\Sigma_{xy}. \end{aligned} \tag{22}$$

This form is in fact equivalent to the ordinary least-squares fit $y = Ax + b + \epsilon$ with $A = \Sigma_{xy}\Sigma_x^{-1}$, $b = \mu_y - \Sigma_{yx}\Sigma_x^{-1}\mu_x$, and where $\epsilon \sim \mathcal{N}(0, \Sigma_{y|x})$ is an error term.

Kalman Update. Given a linear observation model $o = Ax + b + \epsilon$ with $\epsilon \sim \mathcal{N}(0, Q)$, and prior $x \sim \mathcal{N}(\mu, \Sigma)$, the posterior $P(x|o)$ is a Gaussian with mean and covariance

$$\begin{aligned} \mu_{x|o} &= \mu - \Sigma A^T C^{-1}(o - A\mu) \\ \Sigma_{x|o} &= \Sigma - \Sigma A^T C^{-1} A\Sigma \end{aligned} \tag{23}$$

where $C = A\Sigma A^T + Q$.

References

[1] D. Aarno and D. Kragic. Motion intention recognition in robot assisted applications. *Robotics and Autonomous Systems*, 56:692–705, 2008.

[2] T. Asano, E. Sharlin, Y. Kitamura, K. Takashima, and F. Kishino. Predictive interaction using the delphian desktop. In *18th ACM Symposium on User Interface Software and Technology*, page 133141, 2005.

[3] H. A. P. Blom and Y. Bar-Shalom. The interacting multiple model algorithm for systems with markovian switching coefficients. *IEEE T. on Automatic Control*, 33:780–783, 1988.

[4] J. E. Bobrow, B. Martin, G. Sohl, E. C. Wang, F. C. Park, and Junggon Kim. Optimal robot motions for physical criteria. *J. of Robotic Systems*, 18(12):785–795, 2001.

[5] R. R. Burridge and K. A. Hambuchen. Using prediction to enhance remote robot supervision across time delay. In *Intl. Conf. Intel. Rob. Sys.*, pages 5628–5634, 2009. ISBN 978-1-4244-3803-7.

[6] E. Frazzoli, M. A. Dahleh, and E. Feron. Real-time motion planning for agile autonomous vehicles. In *American Control Conf.*, volume 1, pages 43 – 49, 2001.

[7] D. Gossow, A. Leeper, D. Hershberger, and M. Ciocarlie. Interactive markers: 3-d user interfaces for ros applications. *Robotics and Automation Magazine*, 18(4):14 – 15, 2012.

[8] K. Hauser. On responsiveness, safety, and completeness in real-time motion planning. *Autonomous Robots*, 32 (1):35–48, 2012.

[9] K. Hauser and V. Ng-Thow-Hing. Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts. In *Intl. Conf. Rob. Automation*, 2010.

[10] S. Karaman and E. Frazzoli. Incremental sampling-based algorithms for optimal motion planning. In *Robotics: Science and Systems (RSS)*, Zaragoza, Spain, 2010.

[11] D. Kragic, P. Marayong, M. Li, A. M. Okamura, and G. D. Hager. Human-machine collaborative systems for microsurgical applications. *Int. J. of Robotics Research*, 24(9):731–741, 2005.

[12] D. Lane, S. Peres, A. Sándor, and H. Napier. A process for anticipating and executing icon selection in graphical user interfaces. *Int. J. of Human-Computer Interaction*, 19(2):241252, 2005.

[13] E. Lank, Y. Cheng, and J. Ruiz. Endpoint prediction using motion kinematics. In *SIGCHI Conf. on Human Factors in Computing Systems*, pages 637–646, 2007.

[14] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[15] S. Petti and T. Fraichard. Safe motion planning in dynamic environments. *Intl. Conf. Intel. Rob. Sys.*, 2 (6):2210 – 2215, 2005.

[16] O.C. Schrempf, D. Albrecht, and U.D. Hanebeck. Tractable probabilistic models for intention recognition based on expert knowledge. In *Intl. Conf. Intel. Rob. Sys.*, pages 1429 –1434, nov. 2007.

[17] W. Yu, R. Alqasemi, R. Dubey, and N. Pernalete. Telemanipulation assistance based on motion intention recognition. In *Intl. Conf. Rob. Automation*, pages 1121–1126, 2005.

[18] B. Ziebart, A. K. Dey, and J. A. Bagnell. Probabilistic pointing target prediction via inverse optimal control. In *Int. Conf. on Intelligent User Interfaces*, 2012.

[19] R. Zollner, O. Rogalla, R. Dillmann, and M. Zollner. Understanding users intention: programming fine manipulation tasks by demonstration. In *Intl. Conf. Intel. Rob. Sys.*, volume 2, pages 1114 – 1119, 2002.