**Autonomous Robots manuscript No.**
(will be inserted by the editor)

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

**Kris Hauser**

Received: date / Accepted: date

**Abstract:** The approach of inferring user's intended task and optimizing low-level robot motions has promise for making robot teleoperation interfaces more intuitive and responsive. But most existing methods assume a finite set of candidate tasks, which limits a robot's functionality. This paper proposes the notion of *freeform tasks* that encode an infinite number of possible goals (e.g., desired target positions) within a finite set of types (e.g., reach, orient, pick up). It also presents two technical contributions to help make freeform UIs possible. First, an intent predictor estimates the user's desired task, and accepts freeform tasks that include both discrete types and continuous parameters. Second, a cooperative motion planner continuously updates the robot's trajectories to achieve the inferred tasks by repeatedly solving optimal control problems. The planner is designed to respond interactively to changes in the indicated task, avoid collisions in cluttered environments, handle time-varying objective functions, and achieve high-quality motions using a hybrid of numerical and sampling-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 intended targets faster and to track intended trajectories more closely than comparable techniques.

## 1 Introduction

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. The notion of *assisted teleoperation* proposes to embed greater intelligence in the interface to make robots safer and easier to use [1,3,9,25]. One promising approach is to *predict the user's intent* and *provide task-appropriate assistance* [2,18,24,26,28]. In this approach, the robot estimates the operator's intended task using raw input signals and then chooses its action in order to best achieve the desired task. This strategy is appealing for several reasons. First, it can improve performance in the presence of time delays because the robot can anticipate the desired task in the midst of a partially-issued command [8]. Second, it can also address *asymmetry* between input devices and robots when the robot has many more degrees of freedom than a user can control at once. Task-level control can be simpler that the robot, presents a potential advantage over modal interfaces, which provide users control over subsets of degrees of freedom at once (e.g., [22]), but are quite tedious. Human input is also noisy and error-prone, and intent estimation can filter out noise and assistance can prevent execution of unsafe commands. Finally, this approach decouples the overall problem into two subproblems — intent estimation and motion planning — which can be refined at an algorithmic level using quantitative benchmarks.

This paper proposes algorithmic contributions for extending intent prediction frameworks to handle the notion of *freeform tasks*, such as reaching, pointing, or tracking a trajectory, in which the user's intent ranges over a continuous infinity of possibilities. Compared to other approaches that only consider a finite number of tasks, 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 to implement because the robot must not only estimate the task but also several continuous parameters. Moreover it is not

School of Informatics and Computing, Indiana University
901 E. 10th St.
Bloomington, IN 47408 USA
Tel.: +1(812)856-7496
hauserk@indiana.edu

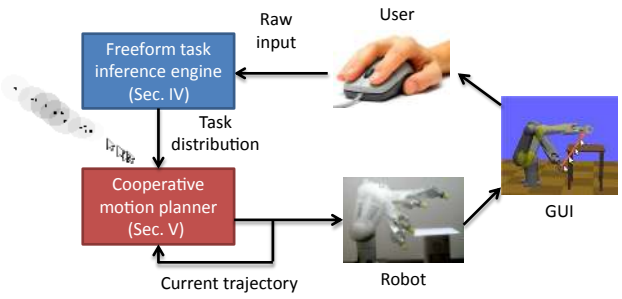


**Fig. 1** The system integrates new contributions for prediction of intended tasks and real-time optimal control.

practical to precompute task-specific motion strategies, and instead the robot must optimize motions on-the-fly.

We present new techniques for estimating freeform tasks as well as planning high quality motions to achieve them (Figure 1). In the intent inference framework, the system deduces a task type and parameters using statistical features of the input time series. Methods are presented for learning and inference with a new time series representation, Gaussian Mixture Autoregression, which has the capability to model nonlinear dynamic processes in the training data. Off-line, the method learns an autoregressive, hybrid discrete/continuous Dynamic Bayesian Network from a database of demonstrations; on-line, task estimation is handled in real-time using Bayesian filtering techniques. This method models and estimates static target and dynamic trajectory estimation in a unified framework, and is able to estimate intended static and dynamic targets with 43% and 13% lower error, respectively, than using the cursor position alone. Experiments demonstrate that it exhibits superior performance to linear models and hidden Markov models in this task.

The second contribution of this paper exploits predictive capabilities to improve trajectory tracking *by anticipating future goal 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. Clutter and complex robot kinematics pose a major problem for local trajectory optimization methods, which are prone to fall into local minima. We present a hybrid planner that combines sampling-based planning techniques [21], to explore globally, with numerical trajectory optimization techniques [5], to refine locally. The planner is responsive and produces higher-quality, collision-free paths than planners that purely rely on global exploration or local optimization.

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 65% compared to systems that either do not perform inference, or that use simpler planners.

## 2 Related Work

**Intent recognition.** As recommended by Goodrich and Olsen (2003) [12] and Green et. al. (2008) [13], one guideline for effective human-robot interaction is to avoid explicit UI mechanisms (e.g., buttons, drop-down menus) for role-switching. It is argued that role-switching unnecessarily burdens the human operator, and instead robots ought to infer their appropriate roles by understanding user intent. Several authors have studied human intent and activity recognition for robot control, primarily in the context of telemanipulation and surgical assist systems [2, 18, 24, 26, 28]. 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. It also employs real-time motion planning techniques to generate behaviors on the fly, rather than pre-defining a motion or control strategy for each task.

**Cursor target prediction.** Another related area in the intelligent user interfaces community is predictive modeling of cursor targets to help select icons in graphical user interfaces. Several methods have been proposed for, such as linear regression from peak velocity [4], directional characteristics [19], kinematic motion models [20], and inverse optimal control techniques [27]. Unlike [19] our work is applicable to a fully continuous set of targets. It also achieves better prediction accuracy than linear models [4, 20, 27] because the Gaussian Mixture Autoregression technique used here is capable of modeling nonlinear characteristics that are observed in user behavior.

**Real-time motion planning.** Sampling-based motion planners such as Probabilistic Roadmaps (PRMs) and Rapidly-Exploring Random Trees (RRTs) are effective at planning collision-free motion for high-dimensional robot systems [21], and have also been successfully applied to hard real-time planing for 2D helicopters [10] and ground vehicles [23]. Recently they have been applied to teleoperation interfaces for robot manipulator arms [14]. But these works have traditionally focused on finding *feasible* paths rather than *optimal* ones. Sampling-based approaches have been recently developed for the optimal motion planning problem [16], 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 [7]. Optimization approaches can achieve optimality with a fast convergence rate, albeit only locally. Our hybrid planner combines the strength of sampling-based and numerical approaches and is designed specifically to

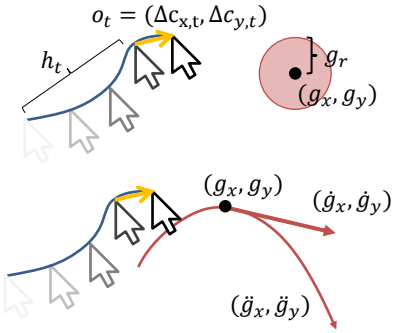


**Fig. 2** Illustrating observation variables $o_t$, history variables $h_t$, and hidden parameters in reach tasks (top) and tracking tasks (bottom).

produce high-quality paths quickly for a broad class of cost functions.

## 3 Problem Overview

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.

The notion of *task* is used 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 $(\ddot{g}_x, \ddot{g}_y)$.

These parameters are illustrated in Figure 2.

In advance, it is worth noting some limitations of the current work. We currently only consider 2D devices because of their widespread use in commodity input devices, but our approach could be extended to other simple and/or noisy input devices. We also assume the environment is known, which means that motion constraints are deterministic and we are able to exploit relatively mature technology for real-time motion planning. It may be possible to extend this work to uncertain environments, but motion planning under uncertainty for many-DOF systems is still an active research topic.

Reach and trajectory following tasks were chosen for proof-of-concept purposes and because intent modeling from user data is tractable; in future work we are interested in extending this approach to more complex tasks, such as depth control, manipulation actions, and end-effector orientation control. It is assumed throughout this paper that the task space is defined such that it covers all of the objectives that the user might wish to accomplish with the robot. As a result, "move joint 3 to angle 55°" is not a goal that can be achieved in the current system. However, if single-joint control were considered to be a relevant task, then it could be encoded into the system and predicted just like any other task. An important question in implementing a task-based approach is whether the human input's has sufficient signal-to-noise ratio and flexibility for the robot to distinguish between tasks.

The system (Figure 1) is composed of two major components: the Freeform Task Inference Engine (FTIE) (Section 4) and the Cooperative Motion Planner (CMP) (Section 5). The FTIE 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.

While the combination of FTIE and CMP is itself novel, we also present new technical contributions in each subsystem that improve performance compared to prior approaches. In particular, we introduce new methods for 1) modeling and estimation of nonlinear probabilistic dynamics for improved task inference, and 2) hybridizing sampling-based planning and numerical optimization for improved real-time motion planning under time-varying objective functions. In Section 6 we describe how these components are integrated to achieve better performance than either component in isolation.

## 4 Freeform Task Inference Engine

The FTIE estimates a time series of unobserved task parameters $(i_t, z_t)$ from a streaming time series of observa-

tions $o_t$, $t = 1,2,\ldots$, obtained from the raw cursor movement. A hybrid discrete/continuous Dynamic Bayesian Network (DBN) is used 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, multimodal transition models. Experiments on our mouse motion datasets suggest that GMA has superior performance to simpler models, such as linear regressions and Kalman filters, as well as comparable nonlinear modeling techniques like Gaussian hidden Markov models (GHMMs) [11] and switching linear dynamic processes [17] (SLDPs).

### 4.1 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 $x_t = (i_t, z_t, h_t)$. 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 3.

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:

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

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:

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

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

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

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

### 4.2 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

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

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

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

where $Z$ is a normalization factor and

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

is the marginal probability that $x$ is drawn from the $c$'th component. Each component in $P(y|x)$ has a mean $\mu_c$ determined by a linear fit $\mu_c = A_c x + b_c$ with a constant covariance $\Sigma_c$. The parameters $A_c$, $b_c$, and $\Sigma_c$ 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 4 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.

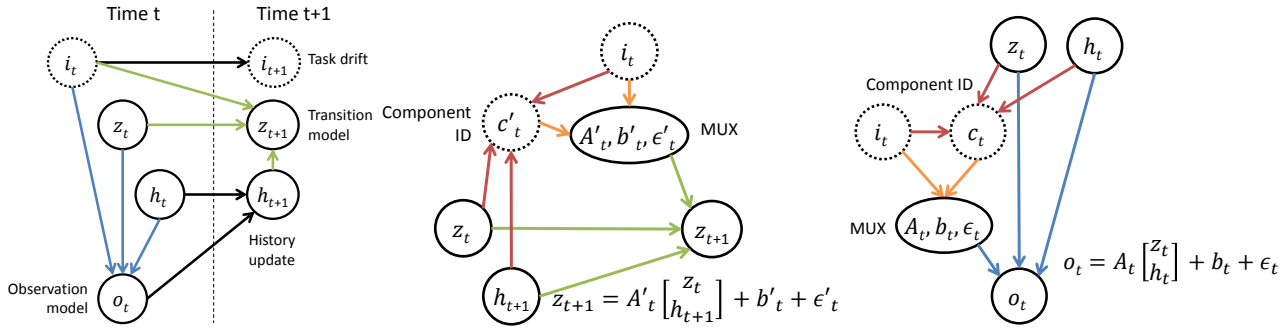


**Fig. 3** Left: the dynamic Bayesian network (DBN) denoting the temporal relationships between hidden task type $i_t$, hidden task parameters $z_t$, history $h_t$, and observation $o_t$. Middle: the transition model represented as a hybrid graphical model with an unknown component $c'_t$ that indexes into a set of linear process models. Right: the observation model. Dashed circles indicate discrete variables while solid circles indicate continuous ones.

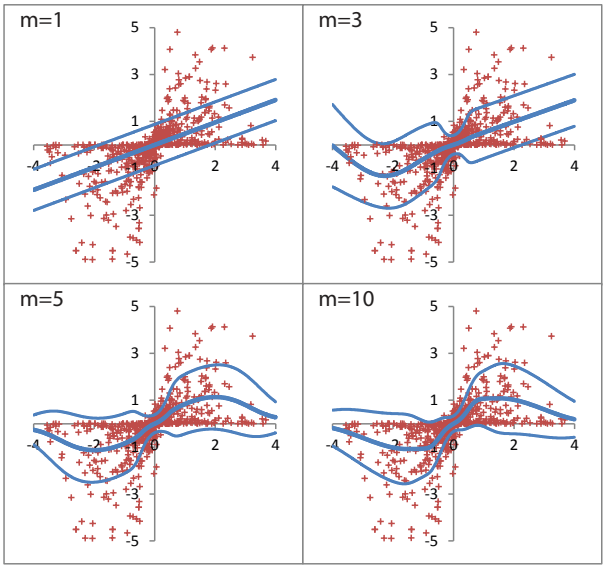


**Fig. 4** Gaussian mixture regressions can model complex nonlinear conditional distributions. This data comes from the reach task training set, with the horizontal target position on the $x$ axis and the horizontal cursor velocity on the $y$ axis (both normalized to unit variance). Curves indicate the mean and standard deviation of the relationship $y(x)$ estimated by GMRs with a varying number $m$ of components.

### 4.3 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 3). 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 $h_{t+1}$.

### 4.4 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:

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

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 $\mu_c$, and covariances $\Sigma_c$, 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, over time the number of components in the belief state would grow exponentially. Hence it is necessary to frequently collapse the belief state representation into a more manageable number of components, in a manner similar to the interacting multiple model (IMM) algorithm that extends the Kalman filter to handle switched linear dynamic processes [6]. 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 $\mu'_i$, and variance $\Sigma'_i$ matched to the subset using the method of moments:

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

**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 $x_{t+1} = Ax_t + b + \varepsilon$ 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.

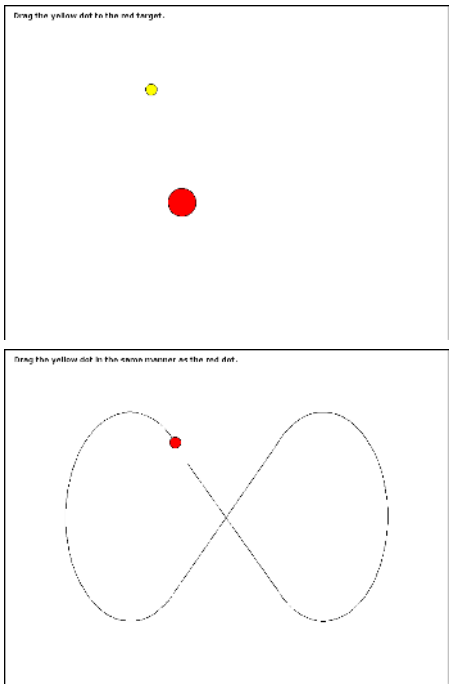


**Fig. 5** Training GUIs for reach tasks (top) and tracking tasks (bottom). Target positions, sizes, trajectories, and speeds are sampled at random.

### 4.5 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 (Figure 5. 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.

Using these GUIs mouse movements and task parameters were gathered from five volunteers at 50Hz, 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.

### 4.6 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.

First, we test study estimation accuracy on single tasks. We compared GMA to the following techniques:

- Cursor only: simply use the cursor position as the estimate.
- Linear Reg.: a linear regression of target position on the last 5 cursor positions, fit using ordinary least-squares.
- Kalman filter: a Kalman filter with target position as hidden state and the last 5 cursor velocities as observations.
- GHMM: a Gaussian Hidden Markov Model (GHMM) with the observation vector containing the target position and last 5 cursor positions, adapted for regression tasks. Fitting was performed using EM techniques [11], and out of 10, 50, 100, 200, and 500 discrete states the 200-state GHMM was chosen because it had the best likelihood on the test set. Target position estimates were calculated using Gaussian mixture regression.
- SLDP: a switching linear dynamic process (SLDP) with target position as hidden state and the last 5 cursor positions as observations. Fitting was performed using approximate EM techniques [17], and out of 5, 10, and 20 discrete states the 10-state SLDP was chosen because it had the best empirical likelihood on the test set.

GMA is similar in spirit to SLDPs and GHMMs, except GMA uses one complex process model rather than many simple process models. One disadvantage of GHMMs is that they do not model the continuous dynamics of hidden parameters. SLDPs have the disadvantage that they must be trained using approximate EM, so fitting is often less robust. Each technique is trained and tested on examples from a single task. The results of testing are shown in Figure 6.

In reach tasks, GMA is 43% better than using the cursor position alone. Figure 7 illustrates how this works. This figure plots the distance-to-target along all test examples for the cursor and the GMA prediction, where each example is normalized to a common initial distance and time. Distance-to-target 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. Future work might look into improving prediction accuracy by modeling this switching phenomenon, combining the strengths of GMA and SLDPs.

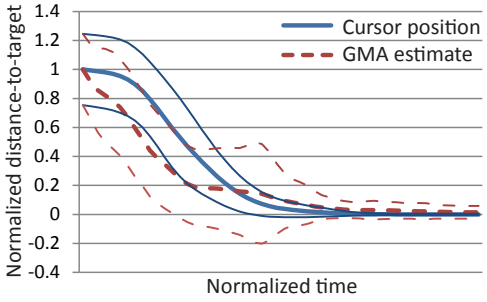


**Fig. 7** Normalized and time-scaled distance-to-target in static target reaching tasks for the cursor position (solid lines) and the GMA estimated target (dotted lines). Mean and standard deviation are plotted.

**Table 1** Cross-task mean squared target prediction errors, normalized to Cursor Only error. Best in column is bolded.

| | Reach | Tracking | Tracking forecast (1s) |
|---|---|---|---|
| Cursor Only | 100% | 100% | 100% |
| GMA (reach only) | **57%** | 151% | 86% |
| GMA (track only) | 112% | **87%** | **60%** |
| GMA (reach+track) | 83% | 108% | 73% |

For trajectory tracking, GMA only estimates the current goal position with 13% lower MSE than simply using the cursor position, which is not a major improvement. But its main strength is its ability to anticipate future target positions. GMA reduces the error in forecasts at 1 s by 40% compared to using the cursor position itself. We also compared two new techniques for forecasting:

- Velocity Extrapolation: extrapolate the cursor's current velocity, as estimated by a moving average with window size 5.
- Linear Reg. (forecast): a linear regression trained specifically to predict 1 s forecasts from the last 5 cursor velocities.

Figure 8 shows some typical trajectory tracking results.

Our second set of experiments evaluates cross-task performance (Table 1). 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 further performance benefits.

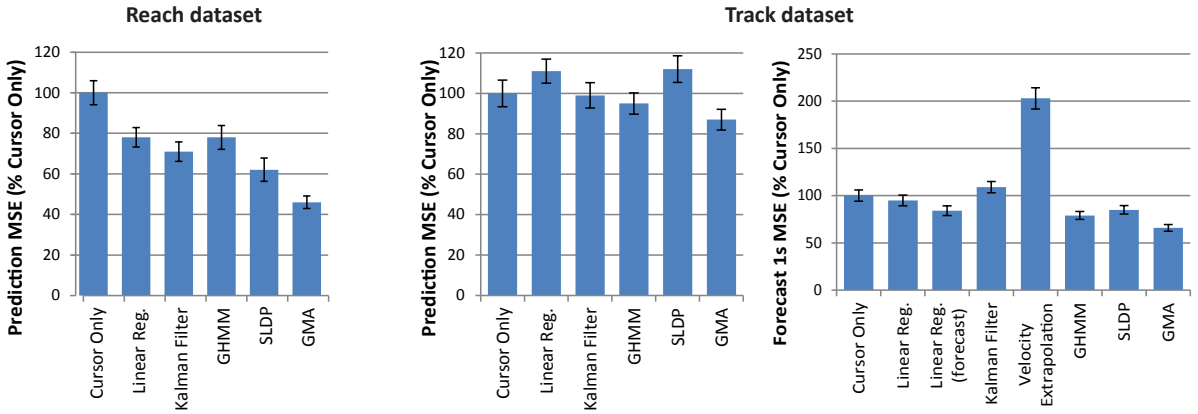


**Fig. 6** Comparing target prediction errors for several models on reach and track datasets. The rightmost plot displays errors in forecasting target position at 1 s in the future. Error bars indicate standard errors, as measured across trials.

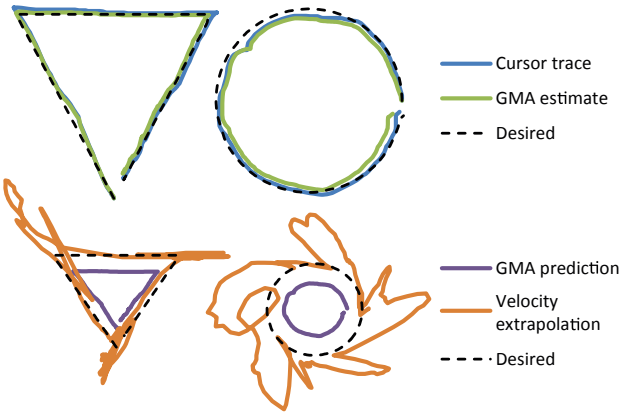


**Fig. 8** Tracking two test trajectories. Top row: target estimates produced by the trajectory tracking model are slightly more accurate than the cursor itself. Bottom row: GMA target forecasts 1 s in the future are substantially more accurate than using the cursor position alone (19% lower MSE on both figures), or velocity extrapolation (42% and 14% lower MSE on the triangle and circle, respectively).

## 5 Cooperative Motion Planner

The second component of our system is a cooperative motion planner that accepts optimal control problems from the task inference engine. Before describing how FTIE and CMP are integrated, this section will first describe the core algorithm that is applicable to general optimal control problems.

CMP accepts optimal control problems with time-varying objective functions and hard constraints like collision avoidance and actuator limits, but incorporates several contributions to make it particularly appropriate for real-time user control. First, 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. Second, as the user-specified goal changes, the trajectory is adjusted by replanning. The replanning framework obeys hard real-time constraints, and tolerates planning and communication delays [14].

### 5.1 Overview

The robot's motion in the configuration space $\mathscr{C}$ 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 $q \in \mathscr{F} \subseteq \mathscr{C}$.

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

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

where $\mathscr{L}$ is an incremental cost and $\Phi$ is a terminal cost. We take $t = 0$ to be the current time.

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.

### 5.2 Hybrid Sampling-Based/Numerical Planning

Our hybrid planner grows a tree of collision-free states $\mathscr{T}$ from the start state forward in time in the configuration / velocity / time space $\mathscr{C} \times \dot{\mathscr{C}} \times \mathbb{R}^+$. Each edge in $\mathscr{T}$ is a relatively short, dynamically-feasible trajectory segment. Each

node $N$ also maintains an accumulated incremental cost $\mathscr{L}(N)$ for the trajectory from the start to $N$.

Like the RRT planner, the tree is grown by sampling a configuration $q_r$ at random and extending a path from an existing node, ending in a configuration $q_d$. 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 [10]). In our case, the steering function constructs a time-optimal acceleration-bounded curve using the method of [15]. Interleaved with random RRT-like expansion, our method also performs numerical optimization to construct trajectory extensions that locally optimize (15).

Pseudocode is listed in Figure 10. Several performance enhancements are used here:

1. We initially seed the tree with the trajectory computed on the prior iteration and attempt to make improvements through local optimization (Lines 1–3). 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 [10] 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 $q_r$, we choose the closest state under the pseudometric that measures the optimal time to reach $q_r$ 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 $\mathscr{L}(N)$ than the current best cost 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 (Line 8). We use the method described in [15], with some minor modifications to ensure that each shortcut makes an improvement in the time-varying cost function.

**Local optimization.** The Local-Optimize subroutine performs numerical optimization of $J$ on the trajectory leading to a node from its parent. The integral in (15) is evaluated using Gaussian quadrature, and the optimization parameters are the target configuration of the steering function $q_d \in \mathscr{C}$ as well as a time scale $\alpha \geq 1$ that extends the time at which $q_d$ is reached. The $\alpha$ 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). If the resulting configuration is infeasible, the optimization is rejected.

**Random expansion.** The Expand-Tree subroutine extends $\mathscr{T}$ at random by sampling a desired configuration $q_r$,

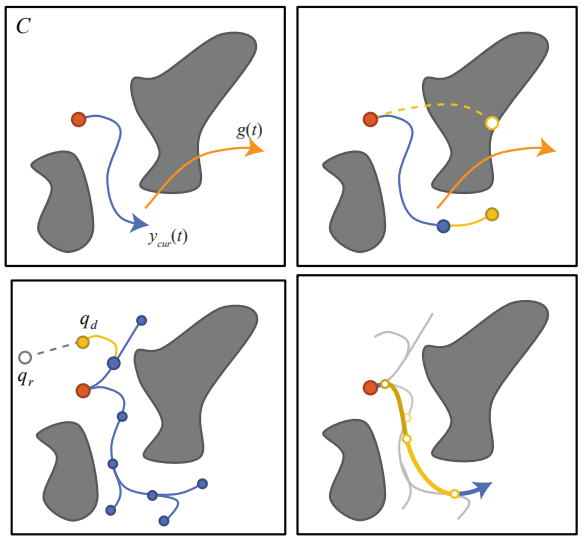


**Fig. 9** Our planner grows a tree in the configuration space $C$ to follow a goal trajectory $g(t)$, using the current trajectory $q_{cur}(t)$ as a seed (top left). The start and end of the trajectory are numerically optimized (top right). The tree is also extended toward randomly selected configurations (bottom left). Finally, the best trajectory is smoothed using a "shortcutting" heuristic (bottom right).

**Hybrid-Planner**$(y_{cur}, J, t_{max})$
1. $\mathscr{T} \leftarrow$ Seed-Tree$(y_{cur})$
2. Let $N_{start}$ and $N_{end}$ be the nodes in $\mathscr{T}$ corresponding to the start and end of the existing path.
3. Do Local-Optimize$(N_{start})$ and Local-Optimize$(N_{end})$.
4. Until $0.5t_{max}$ time elapses, do:
   5. $N \leftarrow$Expand-Tree$(\mathscr{T})$
   6. Local-Optimize$(N)$
7. Let $y$ be the best feasible trajectory in $\mathscr{T}$.
8. Shortcut $y$ until $t_{max}$ time elapses.
9. Return $y$.

**Fig. 10** Pseudocode for the CMP planner.

then generating an extension toward $q_r$ from the "closest" existing node $N$ that satisfies $\mathscr{L}(N) < J_{best}$. This condition ensures that extensions are not attempted from parts of the tree that have no chance of improving upon the current objective value. To measure closeness we use the optimal-travel-time pseudometric $d_t((q(N), \dot{q}(N)), (q_r, 0))$, which uses the the method of [15] to calculate the minimum travel time between two configuration / velocity states under velocity and acceleration bounds. Next, we extend an edge of $\mathscr{T}$, limited by the distance parameter $\delta$. The destination of the edge $q_d$ is set to $q_r$ if $||q_d - q(N)|| \leq \delta$, and otherwise $q_d \leftarrow q(N) + \delta \frac{q_r - q(N)}{||q_r - q(N)||}$. If $q_d$ lies in collision, the process repeats. Otherwise, a new node $N'$ at state $(q_d, 0)$ is added as a child of $N$. As in [10], we also insert a node $N''$ at the midpoint of the trajectory from $N$ to $N'$ so that subsequent steps can proceed from $N''$ without stopping.

**Lazy collision checking.** We use a lazy strategy that delays relatively expensive trajectory collision checking. We

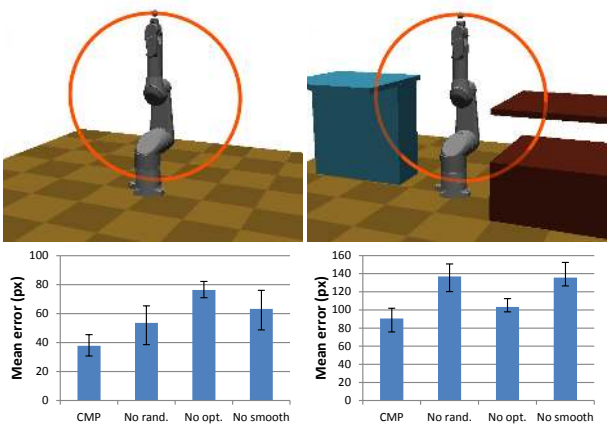


**Fig. 11** Top: Uncluttered and cluttered test scenarios in which the end-effector is instructed to move along a circular trajectory. The trajectory is a priori unknown to the robot. Bottom: Comparing tracking performance when CMP features are disabled. Graphs show the mean, min, and max targeting errors over 10 runs on the uncluttered and cluttered scenarios.

maintain the cost $J_{best}$ of the best *collision-free* path found so far, and is initialized to $J(P_{cur})$. Every time a node $N$ is added to $\mathscr{T}$ we test whether the cost $J(N)$ is lower than $J_{best}$. If so, collision checking is performed on the path leading to $N$. If successful, we set $J_{best} \leftarrow J(N)$. If not, the offending edge and all descendants in $\mathscr{T}$ are deleted.

**Hard real-time constraints.** To obey the hard real-time constraint, the planner is *not* initialized at the robot's current state because once planning is completed, the robot will have already moved. Instead, following [10, 23] the planner is initialized to the predicted state along the current trajectory, propagated forward in time by the planner's time limit $\Delta t$. As in [14] we adapt the time step $\Delta t$ 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 $\Delta t$ when the user changes goals by introducing a scaling factor $e^{-cD}$ where $D$ is the distance between the cursor position on the prior planning iteration and the current iteration. Here $c$ is a sensitivity parameter chosen via a small amount of tuning (in our implementation, $c = 4/h$ where $h$ is the screen height). It is also a simple matter to interrupt a long replanning cycle and start anew if the goal makes a large change.

### 5.3 Experiments

We tested the performance of CMP on a synthetic circle tracking task in both uncluttered and cluttered environments (Figure 11). Experiments suggest the combination of components in the hybrid approach helps our planner perform well across problem variations. The "No rand." case disables randomized exploration, and the planner simply locally optimizes the path with collision checking. Randomized exploration improves performance by 30% and 34% in the uncluttered/cluttered scenarios respectively, because it is able to better circumvent obstacles. "No opt." turns off the optimization step, and performance suffers in the uncluttered scenario because the convergence of random exploration is slow. "No smooth" turns off the postprocessing smoothing step, and performance suffers on both scenarios. Results from pairwise $t$-tests indicates that each component improves performance at significance level $p < 0.001$.

## 6 Integrating FTIE and CMP

To integrate the FTIE and CMP, we convert predictive task estimates into a planning cost function 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.

### 6.1 Expected Cost Functional

Let $g(t)$ denote a screen-coordinate realization of the intended target's trajectory. We want to find a robot trajectory $q(t)$ that minimizes the distance between $g(t)$ and $x(q(t))$, where $x(q)$ is the image-space position of the clicked point at the robot's configuration $q$. When the user clicks and drags a point on the robot, the FTIE provides a time-varying probability distribution $P(g,t)$ on $g(t)$ derived from the inferred task $b_t(z_t)$. Here we estimate $P(g,t)$ for all $t$ from the current time $t = 0$ to some maximum forecasting horizon.

We then optimize the expected cost:

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

where the expectation is taken over goal position $g$. By linearity of expectation, we have

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

Now let the deterministic cost functionals $\mathscr{L}_d$ and $\Phi_d$ measure squared distance to the goal $g(t)$. Specifically, reach and tracking tasks measure the distance between $g(t)$ and the robot's end effector position projected to the screen $x(t) \equiv x(q(t))$. We will also introduce a discount factor $\lambda(t)$ that will be defined below. The resulting formulation is:

$$\begin{aligned} J = &\int_0^T \lambda(t) E_g\left[||x(t) - g(t)||^2\right]dt \\ &+ \int_T^\infty \lambda(t) E_g\left[||x(T) - g(t)||^2\right]. \end{aligned} \tag{18}$$

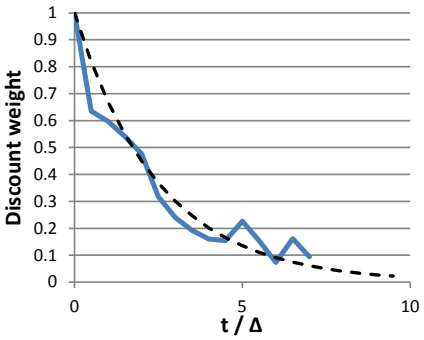


**Fig. 12** The empirical distribution of introspective weights $\lambda(t)$ (solid) and an exponentially decaying fit (dashed).

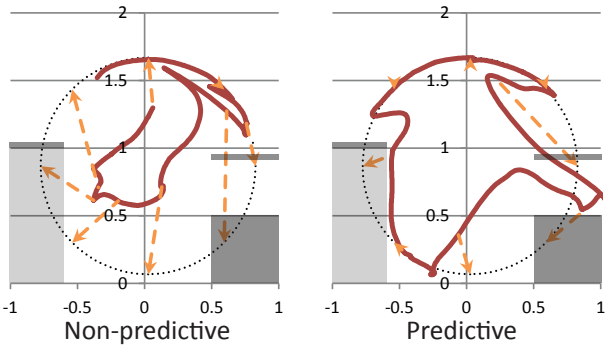


**Fig. 13** Traces of the end effector in the cluttered environment without (left) and with (right) predictive tracking. Because it is planning and executing in real-time, once the non-predictive planner finishes planning the target has already moved a great distance. Predictive tracking anticipates the target's movements and leads to substantially lower error.

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.

### 6.2 Introspective Discounting

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. The rationales for this discount factor is that 1) later planning steps are likely to significantly change the path actually taken, and 2) the belief on $g$ will change while the user input changes. Hence, the planner should spend less effort optimizing the portions of the trajectory in the distant future.

We estimate this expectation by gathering planning statistics on many example 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 $\Delta$ 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 $\lambda$ proportionally to $\Delta$ and estimate a single $b$ that provides the best fit to our training data (Figure 12).

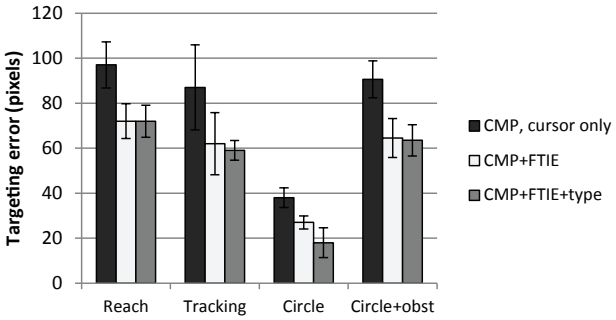


**Fig. 14** Targeting errors with and without intent inference. Results averaged over 20 runs. Error bars indicate one standard deviation.

### 6.3 Experiments

Figure 13 shows an example of how task prediction combined with CMP leads to better trajectory tracking than non-predictive CMP. Overall, our experiments found that prediction improves tracking error by 30% on both the uncluttered and cluttered scenarios, compared to a non-predictive system. Compared to a non-predictive, optimization-based system, our implementation improves tracking by 50% and 54% on these two scenarios, respectively. Compared to a non-predictive, sampling-based system, our implementation improves tracking by 65% and 39%, respectively. All results are statistically significant with $p < 0.001$.

Figure 14 lists mean distance from the robot's end effector to the intended target, with and without task prediction. The Reach and Tracking columns use the natural reach and tracking motions from our testing set as user input, and Circle and Circle+obst use the synthetic circle motions in the uncluttered and cluttered environments, respectively. FTIE improves upon the cursor-only approach by 27% on the natural cursor trajectories and 30% on the circular trajectories. We also examined whether FTIE discriminates well between tasks, and found out that classification performance is in

fact rather poor; the most probable task estimated by GMA is correct only 66% of the time. Surprisingly, this does not have a major effect on overall performance! The rightmost column of Figure 14 suggests that even if perfect task knowledge is introduced into the estimation, targeting error does not change significantly. This indicates that confusing inputs tend to indicate similar intended motions of the end effector. Future work ought to investigate whether similar effects are observed for other tasks besides reaching and trajectory tracking.

## 7 Conclusion

This paper presented algorithms 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.

Future extensions of this work may consider a wider variety of freeform tasks, such as manipulation tasks and richer spatial movements (e.g., depth control and rotations), that cannot be described by screen-space targets. Another interesting line of research might improve task prediction accuracy by considering contextual features of the robot's environment, such as proximity to obstacles and manipulatable objects. Perhaps the most significant challenge in pursuing these avenues is acquiring sufficiently large datasets for learning accurate models of such tasks. Finally, human-robot interaction studies may help determine whether intent inference and cooperative motion planning techniques lead to a better user experience for controlling real robots.

## 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 $y = Ax+b+\varepsilon$ with $A = \Sigma_{xy}\Sigma_x^{-1}$, $b = \mu_y - \Sigma_{yx}\Sigma_x^{-1}\mu_x$, and where $\varepsilon \sim \mathcal{N}(0, \Sigma_{y|x})$ is an error term.

**Kalman Update.** Given a linear observation model $o = Ax+b+\varepsilon$ with $\varepsilon \sim \mathcal{N}(0,Q)$, 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, S. Ekvall, and D. Kragic. Adaptive virtual fixtures for machine-assisted teleoperation tasks. In *Intl. Conf. Rob. Automation*, pages 897 – 903, april 2005.
2. D. Aarno and D. Kragic. Motion intention recognition in robot assisted applications. *Robotics and Autonomous Systems*, 56:692–705, 2008.
3. S. J. Anderson, S. C. Peters, K. D. Iagnemma, and T. E. Pilutti. A unified approach to semi-autonomous control of passenger vehicles in hazard avoidance scenarios. In *Proc. IEEE Int. Conf. on Systems, Man and Cybernetics*, pages 2032–2037, San Antonio, TX, USA, 2009.
4. T. Asano, E. Sharlin, Y. Kitamura, K. Takashima, and F. Kishino. Predictive interaction using the delphian desktop. In *Proc. 18th annual ACM Symposium on User Interface Software and Technology*, page 133141, 2005.
5. J. T. Betts. Survey of numerical methods for trajectory optimization. *Journal of Guidance, Control, and Dynamics*, 21(2):193–207, 1998.
6. 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.
7. J. Bobrow, B. Martin, G. Sohl, E. Wang, F. Park, and J. Kim. Optimal robot motions for physical criteria. *J. of Robotic Systems*, 18(12):785–795, 2001.
8. 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.
9. A. Dragan and S. Srinivasa. Formalizing assistive teleoperation. In *Robotics: Science and Systems*, 2012.
10. E. Frazzoli, M. A. Dahleh, and E. Feron. Real-time motion planning for agile autonomous vehicles. *AIAA J. of Guidance, Control, and Dynamics*, 25(1):116–129, 2002.
11. J.-L. Gauvain and C.-H. Lee. Maximum a posteriori estimation for multivariate gaussian mixture observations of markov chains. *Speech and Audio Processing, IEEE Transactions on*, 2(2):291 – 298, apr 1994.
12. M. A. Goodrich and J. D. R. Olsen. Seven principles of efficient interaction. In *Proc. IEEE International Conference on Systems, Man, and Cybernetics*, pages 3943–3948, October 2003.
13. S. A. Green, M. Billinghurst, X. Chen, and J. G. Chase. Human-robot collaboration: A literature review and augmented reality approach in design. *International Journal of Advanced Robotic Systems*, 5(1):1–18, 2008.
14. K. Hauser. On responsiveness, safety, and completeness in real-time motion planning. *Autonomous Robots*, 32(1):35–48, 2011.
15. K. Hauser and V. Ng-Thow-Hing. Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts. In *Proc. IEEE Int. Conference on Robotics and Automation (ICRA)*, Anchorage, USA, 2010.

16. S. Karaman and E. Frazzoli. Incremental sampling-based algorithms for optimal motion planning. In *Robotics: Science and Systems (RSS)*, Zaragoza, Spain, 2010.
17. C.-J. Kim. Dynamic linear models with markov-switching. *Journal of Econometrics*, 60:1–22, 1994.
18. D. Kragic, P. Marayong, M. Li, A. M. Okamura, and G. D. Hager. Human-machine collaborative systems for microsurgical applications. *The International Journal of Robotics Research*, 24(9):731–741, 2005.
19. D. Lane, S. Peres, A. Sándor, and H. Napier. A process for anticipating and executing icon selection in graphical user interfaces. *International Journal of Human-Computer Interaction*, 19(2):241252, 2005.
20. E. Lank, Y. Cheng, and J. Ruiz. Endpoint prediction using motion kinematics. In *Proc. SIGCHI Conf. on Human Factors in Computing Systems*, pages 637–646, 2007.
21. S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.
22. A. Leeper, K. Hsiao, M. Ciocarlie, L. Takayama, and D. Gossow. Strategies for human-in-the-loop robotic grasping. In *Proc. of Human-Robot Interaction (HRI)*, pages 1–8, Boston, MA, 2012.
23. S. Petti and T. Fraichard. Safe motion planning in dynamic environments. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 3726–3731, 2005.
24. O. Schrempf, D. Albrecht, and U. Hanebeck. Tractable probabilistic models for intention recognition based on expert knowledge. In *Intl. Conf. Intel. Rob. Sys.*, pages 1429 –1434, nov. 2007.
25. J. Shen, J. Ibanez-Guzman, T. C. Ng, and B.-S. Chew. A collaborative-shared control system with safe obstacle avoidance capability. In *Robotics, Automation and Mechatronics, 2004 IEEE Conference on*, volume 1, pages 119–123 vol.1, 2004.
26. 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.
27. B. Ziebart, A. K. Dey, and J. A. Bagnell. Probabilistic pointing target prediction via inverse optimal control. In *Proc. Int'l Conf. on Intelligent User Interfaces*, 2012.
28. 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.