Supervisory teleoperation with online learning and optimal control

Ioannis Havoutis[1,2] and Sylvain Calinon[1]

***Abstract*— We present a general approach for online learning and optimal control of manipulation tasks in a supervisory teleoperation context, targeted to underwater *remotely operated vehicles* (ROVs). We use an online Bayesian nonparametric learning algorithm to build models of manipulation motions as *task-parametrized hidden semi-Markov models* (TP-HSMM) that capture the spatiotemporal characteristics of demonstrated motions in a probabilistic representation. Motions are then executed autonomously using an optimal controller, namely a *model predictive control* (MPC) approach in a receding horizon fashion. This way the remote system locally closes a high-frequency control loop that robustly handles noise and dynamically changing environments. Our system automates common and recurring tasks, allowing the operator to focus only on the tasks that genuinely require human intervention. We demonstrate how our solution can be used for a *hot-stabbing* motion in an underwater teleoperation scenario. We evaluate the performance of the system over multiple trials and compare with a state-of-the-art approach. We report that our approach generalizes well with only a few demonstrations, accurately performs the learned task and adapts online to dynamically changing task conditions.**

I. Introduction

Many useful robotics applications require performing tasks in environments that are not accessible to humans. One typical example is underwater activities, ranging from inspection and maintenance of underwater cables and pipelines, to underwater archeology and marine biology. To this end there has been a boom in underwater *remotely operated vehicles* (ROVs) over the past few years. Nonetheless the cost of using ROVs is still prohibitively high for wider adoption, as currently ROV usage still requires substantial off-shore support.

One of the main limiting factors is that a large off-shore crew is required to supervise and teleoperate the ROV directly from the support vessel. This is mainly due to the need of online teleoperation, i.e. the operator receives visual feedback from an array of cameras on the ROV and accordingly uses a set of buttons, knobs and joysticks to guide the motion of all degrees-of-freedom (DoF) of the ROV, including body and arm(s). This cost can be reduced by moving the support and teleoperation team to an on-shore facility and communicating with the ROV remotely. Such a change introduces delays to the teleoperation control, making direct control less efficient and increasing the cognitive load on the operator.

[1]Idiap Research Institute, Martigny, Switzerland. {ioannis.havoutis, sylvain.calinon}@idiap.ch

[2]Oxford Robotics Institute, Department of Engineering Science, University of Oxford, United Kingdom. ihavoutis@robots.ox.ac.uk

This work was in part supported by the DexROV project through the EC Horizon 2020 programme (Grant #635491).

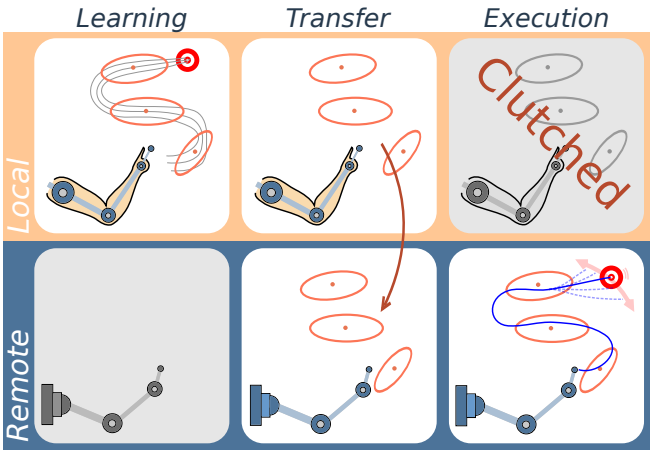


Fig. 1. Overview of supervisory teleoperation within our approach. *Top:* Local system where a teleoperator is using an exoskeleton to demonstrate motions to the system and to perform direct teleoperation whenever necessary. *Bottom:* Remote side where a robotic manipulator receives a learned model of the task at hand and can then execute the task autonomously using local feedback. During autonomous execution, the overall system is clutched, i.e. the state of the local system does not affect the state evolution on the remote side.

Within the DexROV project [1], we are developing a novel teleoperation paradigm where the operator can shift between direct teleoperation of the ROV manipulator arms (when unstructured interactions with the environment are required) and supervisory control (for the ROV to autonomously perform common and recurring tasks). In our approach, such tasks are learned by demonstration directly from the operator's input. This way, our system can build up a library of tasks over a number of missions, that is in turn exploited to automate the ROV tasks.

In this paper, we are interested in *incrementally* building such models from demonstrated motions as *task-parametrized hidden semi-Markov models* (TP-HSMM), using an online Bayesian nonparametric learning algorithm [2]. This model is then transferred to the remote system and used to perform the task autonomously, using local feedback, with an optimal control method, *model predictive control* (MPC), that is robust to noise and perturbations.

The contributions of this paper are: (*i*) a method for online, incremental learning of tasks from demonstrations in a probabilistic, non-parametric manner; (*ii*) an MPC-based controller robust to noise and perturbations that generates control commands online based on learned task models; (*iii*) a system of supervisory teleoperation that uses previously learned task models to autonomously complete tasks; (*iv*) a learning by demonstration approach in the context of teleoperation with supervisory control, targeted to underwater

ROVs.

With our approach we are able to incrementally learn motions in a piecewise fashion, without the need to re-train the model in batch or to set the number of components by hand. Consequently we do not need to keep any demonstration data (datapoints are discarded after observation) while the model automatically grows as needed. We show how ROV skills can be learned on the local side (the operator's site), and how this model is then used to autonomously execute the task on the remote side (ROV/robot's side). With our approach, the model parameters need to be communicated from the operator's side to the teleoperated system, while all feedback relevant to task execution is local to the remote side. This makes the overall method robust to noise and perturbations, while significantly reducing the workload of the operator.

II. Motivation & Related work

Teleoperation is one of the earliest applications of robotics as it allows human operators to reach and interact with environments that are inaccessible or potentially dangerous. Teleoperation approaches fall under three broad categories; direct control, shared control and supervisory control. Within a *direct control* approach, all aspects of the remote system are controlled by the human operator, i.e. the remote system needs no intelligence and no autonomy.

Shared control approaches seek to offload part of the control effort to the remote system. These approaches have a broad application domain ranging from medical robotics [3] to factory floor automation [4]. In many applications, shared control is most encountered as virtual fixtures [5]. These are virtual elements that model knowledge about some aspect of the task and are used to guide, limit, or assist the operator when performing the task [6], [7]. Recently [8] demonstrated how virtual fixtures can be learned from data and how new fixtures can be added to the system so that it can adapt to new manipulation examples. In our recent work in shared control [9], we showed how a teleoperation system can be designed to rely on learned probabilistic models of manipulation tasks in combination with online operator's input.

Supervisory control approaches imply that the remote system possesses some degree of intelligence and can operate autonomously to a given extent. Supervisory control was introduced in [10] and was initially developed for space robotics where the communication delay could exceed several minutes [11]. Ideally within a supervisory control framework the operator needs only to provide high-level task goals to the remote system, which in turn uses local sensory feedback directly and closes local control loops to achieve the commanded task.

Our system is a combination of direct control, where the operator assumes complete command to handle unknown tasks, and supervisory control, where the system can autonomously execute tasks that have been previously learned, relying on local sensory feedback and automatically compensating for possible noise and perturbations.

For the encoding and retrieval of movements, Hidden Markov Models (HMMs) are often used but are limited by the simplistic state duration modeling that they provide. Other signal processing related disciplines, such as speech synthesis, have developed a number of models that seek to model state duration information more explicitly (for an overview see [13]). One such model is the Hidden Semi-Markov Models (HSMM) [14]. Recently we experimented with the use of HSMM in robot applications, by contrasting it to a set of different graphical model based approaches [15]. HSMMs are relevant for robot motion generation because they model transitions and durations of states, providing a relative time instead of a global time representation. In [16], we exploited this local duration representation for autonomously learning and reproducing the tasks of opening a valve and avoiding obstacles.

The approach that we propose in this paper for online HSMM estimation draws parallels to the DP-means extension of HMM presented in [17]. There the authors present a small-variance asymptotic analysis of the HMM and its infinite-state Bayesian nonparametric extension. Our solution is based on [2] and a small-variance asymptotic analysis of the Dirichlet Process clustering algorithm, that leads to very efficient online model learning.

An alternative *learning from demonstration* (LfD) approach is to use Probabilistic Movement Primitives (ProMP) [12], that we will use in our experiments for comparison. ProMP uses a model-free approach to encode a distribution over trajectories and analytically derive a stochastic feedback controller to reproduce the given trajectory distribution. This allows for flexibility over the possible motion generation such as spatial and temporal rescaling, combination and blending of the modeled motion primitives (MPs).

III. Approach

Here we present the online TP-HSMM formulation used throughout this work. This model leverages the spatial representation that the TP-GMM [18] provides, combined with an HSMM that captures the temporal evolution of the motion. Our model can natively handle changing task parametrizations and is trained online and incrementally from demonstration data.

The task parameters can be regarded as candidate coordinate systems that are relevant to the task at hand. For example, consider the task of reaching for a handle with a robotic manipulator. One task parameter can be the pose of the manipulator base, while a second task parameter can be the pose of the handle. Varying the pose of the handle and/or the pose of the base of the manipulator require adaptation of the motion to reach the handle. Task parameters in our model are represented by P candidate coordinate systems, defined at time step t by $\{\boldsymbol{b}_{t,j}, \boldsymbol{A}_{t,j}\}_{j=1}^{P}$, representing an affine transformation for each frame of reference.

Each demonstration datapoint $\{\boldsymbol{\xi}_t \in \mathbb{R}^D\}$ is observed from the perspective of each of the different frames, resulting in P samples that can be collected from each frame individually or computed with

$$\boldsymbol{\xi}_t^{(j)} = \boldsymbol{A}_{t,j}^{-1}(\boldsymbol{\xi}_t - \boldsymbol{b}_{t,j}). \quad (1)$$

For example, a demonstrated motion is provided in a global coordinate system and is simultaneously projected to the

local coordinate systems of the manipulator base and the handle. Note that Eq. (1) can handle both positions and orientations, see [19] where Riemannian manifolds are exploited in task-parametrized models.

The parameters of a TP-HSMM with K components are

$$\theta = \left\{\pi_i,\ \{a_{i,l}\}_{l=1}^{K},\ \{\boldsymbol{\mu}_i^{(j)},\ \boldsymbol{\Sigma}_i^{(j)}\}_{j=1}^{P},\ \mu_i^{\mathcal{D}},\ \Sigma_i^{\mathcal{D}}\right\}_{i=1}^{K},$$

where π_i are the initial state distributions, $\boldsymbol{\mu}_i^{(j)}$ and $\boldsymbol{\Sigma}_i^{(j)}$ are the center and covariance matrix of the i-th Gaussian component in frame j, $\mu_i^{\mathcal{D}}$ and $\Sigma_i^{\mathcal{D}}$ are univariate Gaussian distributions that model the duration of each state, and $a_{i,l}$ the transition probabilities between states.

A. Online model learning

Learning of the TP-HSMM parameters is performed incrementally and online. The advantages of our method are: *(a)* it does not require an initialization step (typical of EM approaches), *(b)* the number of components grows as new data is available and does not need an explicit definition of K, *(c)* the model is trained online with piecewise demonstrations, allowing incremental additions, adaptations and refinements.

To learn the model online we use a variant of Dirichlet Process clustering called DP-means [2]. Dirichlet Process is a well known Bayesian non-parametric approach to cluster data that automatically estimates the number of components required for the model. This process uses Gibbs sampling so it cannot be generally used to learn a model online. The DP-Means variant of Dirichlet process that we use was introduced to simplify the learning process by making a parallel with the K-Means algorithm by small variance asymptotics analysis. The idea is to avoid Gibbs sampling by substituting the evaluation of the posterior probabilities of each component with the distance of datapoints from the center of each component. This can be done by specifying a maximum distance between two components determining when to add a new component to the model.

From a current TP-HSMM configuration with K components, each point $\boldsymbol{\xi}_t$ is assigned to the nearest component. If the distance from any component is higher than the maximum distance λ, then the point is assigned to a new component with $\boldsymbol{\mu}_i = \boldsymbol{\xi}_t$ and $\boldsymbol{\Sigma}_i = \hat{\boldsymbol{\Sigma}}$, a preassigned fixed covariance (at start-up, the first demonstration datapoint is used to initialize the model). If the datapoint is added to an existing component, then $\boldsymbol{\mu}_i$ and $\boldsymbol{\Sigma}_i$ are updated according to the MAP estimate [20], taking the previous values of center and covariance as priors. This results in a very fast clustering algorithm, that only requires a maximum distance λ and a minimum covariance $\hat{\boldsymbol{\Sigma}}$ (Algorithm 1).

For the duration aspect of the online-TP-HSMM, we need to estimate the center and covariance for each state duration as a distribution with parameters $\{\boldsymbol{\mu}_i^{\mathcal{D}}, \boldsymbol{\Sigma}_i^{\mathcal{D}}\}_{i=1}^{K}$, as well as the transition probabilities $a_{i,l}$ that can be arranged as a $K \times K$ transition probability matrix, where each element represents the probability to move to state q_l, while currently being in state q_i. For all demonstrations, given the spatial model that was learned above, we can compute the state probabilities of every datapoint. This way, for each datapoint $\boldsymbol{\xi}_t$ we can estimate the state q_j and the previous state q_i. To build up the transition probabilities $a_{i,l}$, we keep a trace of $c_{i,j} \in \mathbb{R}^{K \times K}$, counting the number of state transitions that are not self-transitory, by performing a pass through each demonstration.

When computing the transition probabilities we only need to keep track of the non self-transitory instances as we are modeling the relative time during which the system will stay in each state. We used here a univariate Gaussian distribution $\mathcal{N}(\mu_i^{\mathcal{D}}, \Sigma_i^{\mathcal{D}})$ to model this duration, but other distributions are possible, including lognormal and gamma distributions. Hence, we bypass the computationally expensive HSMM batch training procedure and replace it with an iterative approach keeping statistics over the state transitions. This way, as we add each datapoint, we keep track of each state duration and accordingly update the statistics at each state. This is done using a running statistics method to compute the mean and variance for each state duration. This requires that we only keep track of the total number of samples (N) while we incrementally add new datapoints. Our online learning algorithm is summarized in Algorithm 1.

Algorithm 1 Online TP-HSMM model learning, using the DP-means algorithm. Note, superscript (j) for each frame is dropped for readability. Superscript (o) stands for old value.

Input: max distance λ, min. covariance $\hat{\boldsymbol{\Sigma}}$,
Input: $\boldsymbol{\xi}_t$ datapoint at time t
  Initialize $K = 1, N = 1 (\#datapoints), \boldsymbol{\mu}_1 = \boldsymbol{\xi}_0, \boldsymbol{\Sigma}_1 = \hat{\boldsymbol{\Sigma}}$
  while $\boldsymbol{\xi}_t$ **do**
    for $j = 1$ **to** P **do**
      $d_{t,i} = \|\boldsymbol{\xi}_t - \boldsymbol{\mu}_i\|_2,\ i = 1 \dots K,\ j = 1 \dots P$
      if $\min(d_{t,i}) > \lambda$ **then**
        $K = K + 1,\ \pi_K = \frac{1}{t},\ \boldsymbol{\mu}_K = \boldsymbol{\xi}_t,\ \boldsymbol{\Sigma}_K = \hat{\boldsymbol{\Sigma}}$
      else
        $q_t = \text{argmin}_i\, d_{t,i}$
        $\pi = \frac{1}{N} + \pi_{q_t},\ \pi_{q_t} = \frac{1}{K},$
        $\boldsymbol{\mu}_{q_t} = \frac{1}{\pi}(\pi_{q_t}\boldsymbol{\mu}_{q_t}^{(o)} + \frac{\boldsymbol{\xi}_t}{N})$
        $\boldsymbol{\Sigma}_{q_t} = \frac{\pi_{q_t}}{\pi}(\boldsymbol{\Sigma}_{q_t}^{(o)} + (\boldsymbol{\mu}_{q_t}^{(o)} - \boldsymbol{\mu}_{q_t})(\boldsymbol{\mu}_{q_t}^{(o)} - \boldsymbol{\mu}_{q_t})^\top)$
        $\quad + \frac{1}{N\pi}(\hat{\boldsymbol{\Sigma}} + (\boldsymbol{\xi}_t - \boldsymbol{\mu}_{q_t})(\boldsymbol{\xi}_t - \boldsymbol{\mu}_{q_t})^\top)$
      end if
      if $q_t = q_{t-1}$ **then** # $(t > 1)$
        $d = d + 1$
      else
        $c_{q_{t-1},q_t} = c_{q_{t-1},q_t} + 1$
        $a_{q_{t-1},q_t} = c_{q_{t-1},q_t} / \sum_{k=1}^{K} c_{q_{t-1},k}$
        $\mu_{q_{t-1}}^{D} = \mu_{q_{t-1}}^{D^{(o)}} + \frac{(d - \mu_{q_{t-1}}^{D^{(o)}})}{N_{q_t}}$
        $e = e + (d - \mu_{q_{t-1}}^{D^{(o)}})(d - \mu_{q_{t-1}}^{D})$
        $\Sigma_{q_{t-1}}^{D} = \frac{e}{(N_{q_t} - 1)}$ # $(N_{q_t} > 1)$
        $d = 0, \quad N_{q_t} = N_{q_t} + 1, \quad \mu_{q_{t-1}}^{D^{(o)}} = \mu_{q_{t-1}}^{D}$
      end if
    end for
    $N = N + 1$
  end while
  return $\theta^* = \left\{\pi_i,\ \{a_{i,l}\}_{l=1}^{K},\ \{\boldsymbol{\mu}_i^{(j)},\ \boldsymbol{\Sigma}_i^{(j)}\}_{j=1}^{P},\ \mu_i^{\mathcal{D}},\ \Sigma_i^{\mathcal{D}}\right\}_{i=1}^{K}$

B. Reproduction

We use a model predictive controller (MPC) to compute control commands, based on the current state of the system and the learned TP-HSMM model. MPC uses a model of the system and computes control commands based on the predicted state evolution, allowing the controller to anticipate

future events. We use a linear unconstrained MPC formulation, using a discrete unit mass double integrator system as the system model. Accordingly the system dynamics have the form

$$\boldsymbol{\xi}_{t+1} = \boldsymbol{A}\boldsymbol{\xi}_t + \boldsymbol{B}\boldsymbol{u}_t,\ \boldsymbol{x}_t = \boldsymbol{C}\boldsymbol{\xi}_t, \tag{2}$$

where $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$ are the system dynamics, input and output matrices, respectively. Given the linear system above, N_p state predictions $\boldsymbol{\xi}_r$ with $r \in \{t+1, \dots, t+N_p\}$ are generated, given the current state $\boldsymbol{\xi}_t$ (position and velocity) and N_c control commands $\boldsymbol{u}_r$ with $r \in \{t, \dots, t+N_c-1\}$. The state predictions can be written in matrix-vector form as

$$\underbrace{\begin{bmatrix} \boldsymbol{A} \\ \boldsymbol{A}^2 \\ \boldsymbol{A}^3 \\ \vdots \\ \boldsymbol{A}^{N_p} \end{bmatrix}}_{\boldsymbol{S}^{\boldsymbol{\xi}}} \boldsymbol{\xi}_t + \underbrace{\begin{bmatrix} \boldsymbol{B} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{A}\boldsymbol{B} & \boldsymbol{B} & \cdots & \boldsymbol{0} \\ \boldsymbol{A}^2\boldsymbol{B} & \boldsymbol{A}\boldsymbol{B} & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{A}^{N_p-1}\boldsymbol{B} & \boldsymbol{A}^{N_p-2}\boldsymbol{B} & \cdots & \boldsymbol{A}^{N_p-N_c}\boldsymbol{B} \end{bmatrix}}_{\boldsymbol{S}^{\boldsymbol{u}}} \underbrace{\begin{bmatrix} \boldsymbol{u}_t \\ \boldsymbol{u}_{t+1} \\ \boldsymbol{u}_{t+2} \\ \vdots \\ \boldsymbol{u}_{t+N_c-1} \end{bmatrix}}_{\boldsymbol{u}}, \tag{3}$$

$$\boldsymbol{x} = (\boldsymbol{I}_{N_p} \otimes \boldsymbol{C})(\boldsymbol{S}^{\boldsymbol{\xi}}\boldsymbol{\xi}_t + \boldsymbol{S}^{\boldsymbol{u}}\boldsymbol{u}) \tag{4}$$

where $\boldsymbol{\xi} = [\boldsymbol{\xi}_{t+1}^\top, \boldsymbol{\xi}_{t+2}^\top, \dots, \boldsymbol{\xi}_{t+N_p}^\top]^\top$, $\boldsymbol{x} = [\boldsymbol{x}_{t+1}^\top, \boldsymbol{x}_{t+2}^\top, \dots, \boldsymbol{x}_{t+N_p}^\top]^\top$, $\boldsymbol{I}_{N_p}$ is an $N_p \times N_p$ identity matrix and $\otimes$ is the Kronecker product.

We compute the control command $\boldsymbol{u}_t$ at time t, by minimizing a quadratic cost function over the prediction horizon. The cost function has the form

$$J = \sum_{r=t+1}^{t+N_p} (\hat{\boldsymbol{\xi}}_r - \boldsymbol{\xi}_r)^\top \boldsymbol{Q}_r (\hat{\boldsymbol{\xi}}_r - \boldsymbol{\xi}_r) + \sum_{r=t}^{t+N_c-1} \boldsymbol{u}_r^\top \boldsymbol{R}_r \boldsymbol{u}_r, \tag{5}$$

where $\hat{\boldsymbol{\xi}}_r$ and $\boldsymbol{\xi}_r$ are the desired and the current state of the system. $\boldsymbol{Q}_r$ is the tracking cost matrix and $\boldsymbol{R}_r$ is the control cost matrix. The same cost function rewritten in batch form is

$$J = (\hat{\boldsymbol{\xi}} - \boldsymbol{\xi})^\top \boldsymbol{Q} (\hat{\boldsymbol{\xi}} - \boldsymbol{\xi}) + \boldsymbol{u}^\top \boldsymbol{R} \boldsymbol{u}, \tag{6}$$

where $\boldsymbol{Q} = \text{blockdiag}(\boldsymbol{Q}_{t+1}, \boldsymbol{Q}_{t+2}, \dots, \boldsymbol{Q}_{t+N_p})$ and $\boldsymbol{R} = \text{blockdiag}(\boldsymbol{R}_t, \boldsymbol{R}_{t+1}, \dots, \boldsymbol{R}_{t+N_c-1})$.

We compute the vector of control commands $\boldsymbol{u}$ by substituting (3) into (6) and minimizing with respect to $\boldsymbol{u}$,

$$\boldsymbol{u} = (\boldsymbol{S}^{\boldsymbol{u}\top}\boldsymbol{Q}\boldsymbol{S}^{\boldsymbol{u}} + \boldsymbol{R})^{-1} \boldsymbol{S}^{\boldsymbol{u}\top} \boldsymbol{Q} (\hat{\boldsymbol{\xi}} - \boldsymbol{S}^{\boldsymbol{\xi}}\boldsymbol{\xi}_t). \tag{7}$$

State sequence: The learned TP-HSMM model is used to generate a state sequence given the current set of coordinate systems $\{\boldsymbol{b}_{t,j}, \boldsymbol{A}_{t,j}\}_{j=1}^P$. First we generate a GMM with parameters $\{\pi_i, \hat{\boldsymbol{\mu}}_{t,i}, \hat{\boldsymbol{\Sigma}}_{t,i}\}_{i=1}^K$ where

$$\mathcal{N}\Big(\hat{\boldsymbol{\mu}}_{t,i}, \hat{\boldsymbol{\Sigma}}_{t,i}\Big) \propto \prod_{j=1}^{P} \mathcal{N}\Big(\hat{\boldsymbol{\mu}}_{t,i}^{(j)},\ \hat{\boldsymbol{\Sigma}}_{t,i}^{(j)}\Big), \text{ with}$$
$$\hat{\boldsymbol{\mu}}_{t,i}^{(j)} = \boldsymbol{A}_{t,j}\boldsymbol{\mu}_i^{(j)} + \boldsymbol{b}_{t,j}\ ,\quad \hat{\boldsymbol{\Sigma}}_{t,i}^{(j)} = \boldsymbol{A}_{t,j}\boldsymbol{\Sigma}_i^{(j)}\boldsymbol{A}_{t,j}^\top, \tag{8}$$

where the result of the Gaussian product is given by

$$\hat{\boldsymbol{\Sigma}}_{t,i} = \Big(\sum_{j=1}^{P} \hat{\boldsymbol{\Sigma}}_{t,i}^{(j)-1}\Big)^{-1},\quad \hat{\boldsymbol{\mu}}_{t,i} = \hat{\boldsymbol{\Sigma}}_{t,i} \sum_{j=1}^{P} \hat{\boldsymbol{\Sigma}}_{t,i}^{(j)-1} \hat{\boldsymbol{\mu}}_{t,i}^{(j)}. \tag{9}$$

Next we generate the state sequence by recursively computing the probability of the datapoint $\boldsymbol{\xi}_t$ to be in state i at time step t, given the partial observation $\{\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \dots, \boldsymbol{\xi}_t\}$, using the forward variable $\alpha_{i,t}^{\text{HSMM}}$ as in [13] with

$$\alpha_{i,t}^{\text{HSMM}} = \sum_{j=1}^{K} \sum_{d=1}^{d^{\max}} \alpha_{j,t-d}^{\text{HSMM}}\, a_{j,i}\, \mathcal{N}_{d,i}^{\mathcal{D}} \prod_{s=t-d+1}^{t} \mathcal{N}_{s,i}, \quad \text{where}$$
$$\mathcal{N}_{d,i}^{\mathcal{D}} = \mathcal{N}(d|\mu_i^{\mathcal{D}}, \Sigma_i^{\mathcal{D}}) \text{ and } \mathcal{N}_{s,i} = \mathcal{N}\big(\boldsymbol{\xi}_s|\ \hat{\boldsymbol{\mu}}_{s,i}, \hat{\boldsymbol{\Sigma}}_{s,i}\big),$$

that is computed with an iterative procedure. This generative process can be constructed by setting equal observation probability $\mathcal{N}_{s,i} = 1\ \forall i$. This yields a step-wise state reference wiht labels $\boldsymbol{s} = \{s_1, \dots, s_{N_p}\}$, that we use as control targets $\hat{\boldsymbol{\xi}}_r$ and cost matrices $\boldsymbol{Q}_r$ in the MPC formulation described earlier.

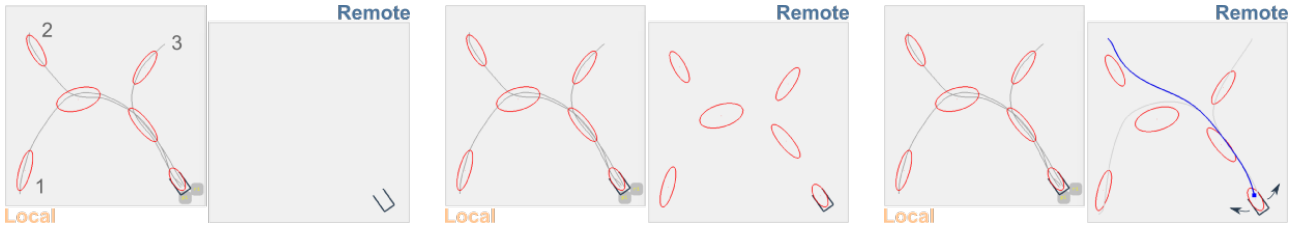
IV. Planar example

We use a planar example to illustrate the steps of our approach. The task is to reach the "U" shape, drawn in Fig. 2(a) at the bottom right corners of both the local and remote sides. The operator uses a mouse cursor to interact with the local side and provide motion demonstrations. Snapshots are presented in Fig. 2 and in the accompanying video.

We begin by learning the task in the local side, where the operator demonstrates a number of motions that are encoded online. In the example of Fig. 2(a), 3 motions are demonstrated incrementally, resulting in a model with 6 Gaussian components. Note that components are added to the model as motions are being demonstrated. Fig. 3 shows the transition matrix and duration probabilities of the learned motion model. Note that the 3 demonstrations resulted in 3 paths as shown by the graph connectivity.

In Fig. 2(b) the learned model is transferred to the remote side. In practice, we do this step either once the operator is satisfied with the learned model or just whenever the model changes. For an ROV mission targeting multiple tasks, the transfer of a general model –or task library– can be performed at the start of each mission, while after an initial learning period most task models would already belong to the learned task library.

Now the remote side can autonomously execute the motion that the operator has demonstrated. Note here that the operator can directly teleoperate the system when the controller is not active, but once the MPC controller initializes the state evolution of the remote side is decoupled from the operator's input, i.e. the system is *clutched*. Execution of the motion begins with a command from the operator's side. The controller generates control commands according to the learned task model and the current task parameters that are local to the remote system Fig. 2(c). This way the position of the U-shaped reference that this motion is reaching can change at each time step, while the controller continuously adjusts to this dynamically changing environment. In our example we use a simple oscillation around the position of the target shape to show how the controller adapts to this changing task parametrization (see accompanying video).



(a) Learning (b) Transfer (c) Execution

Fig. 2. Planar example of learning and control with the TP-HSMM. Note that in each plot the left panel represents the local system (operator's side) while the right panel represents the remote system (robot's side). See Sec. IV for a detailed explanation.

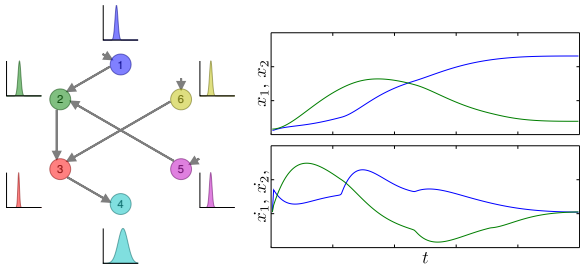


Fig. 3. (*Left*:) Graphical representation of the transition matrix and duration probabilities for each state. Note the three paths formed according to our 3 demonstrations in (Fig. 2(a)). (*Right*:) Executed motion on the remote side, using the learned TP-HSMM and the MPC controller.

V. Experimental Setup

We use the two-armed Baxter robot as a working example of a teleoperation system. Each of Baxter's arms has 7 DoFs, actuated by series-elastic actuators, enabling force/torque control of the joints. We use the left arm as the local system (operator's side) and the right arm as the remote system (robot's side). The left arm is used for kinesthetic demonstrations of the motion in the local environment of the operator, by using a controller compensating the effect of gravity on the arm. We use markers and an RGB-D sensor to track task-relative reference frames both in the local and the remote systems. Fig. 5 provides an overview of the experimental setup.

This setup is used as a running mock-up platform within the DexROV project, where the remote system will later in the project be replaced by the underwater ROV manipulator, and the local system will be replaced by an arm exoskeleton worn by the operator.

Fig. 4. Example of a standardised hot-stab and receptacle [21].

To demonstrate our approach, we chose one of the most frequently executed tasks in underwater ROVs [22]. This is the task of inserting a hot-stab plug into a hot-stab receptacle (Fig. 4), which is used to provide hydraulic actuation to most of the tools employed in underwater facilities. Hot-stabbing also shares similarities with the standard peg-in-the-hole task in robotics applications (see Fig. 5 for our hot-stabbing mock-up).

VI. Evaluation

We begin with the kinesthetic demonstration of the hot-stabbing task. The operator uses the arm to perform the motion targeting a reference in his environment. This reference provides the task parameters for projecting the demonstrated data to the task-local frame of reference. In this example we use one task-parametrized frame ($P = 1$), attached to the hot-stab receptacle. The TP-HSMM is being built online and incrementally, adding Gaussian components as needed. A short comparison between an online and a batch learned TP-HSMM is available in [23]. In our examples we train a model with 5 demonstrations. Note that the TP-HSMM is learned online and does not need any record of previous demonstration data (we keep here the data only for visualization and subsequent comparison). Fig. 6 (*left*) Shows the demonstrated end-effector motions to the local reference, while Fig. 6 (*middle*) shows the demonstrations projected to the task-local frame (Eq. (1)). Fig. 6 (*right*) shows the resulting TP-HSMM. Note the paths that evolve from the start of the motions and how these converge to the hot-stab receptacle reference. Datapoints that are added away from the existing model are modeled by distinct Gaussians, while in areas where the trajectories converge, i.e. at the final approach towards the receptacle, components are shared among datapoints. In this example we used a distance λ of 10cm. In setting λ, one needs to take into consideration the size of the workspace and the accuracy needed for the task at hand. Intuitively, setting a large λ results in a system with fewer Gaussians and smoother response, while a smaller λ results in a system with more components and a closer fit of the demonstrated motions.

Fig. 7 (*left*) presents 5 examples of generated motions

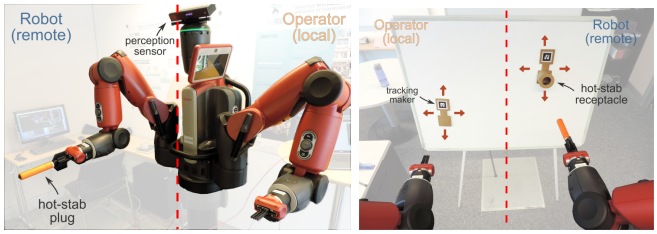


Fig. 5. Experimental setup of teleoperation mock-up with the Baxter robot. The operator uses the left arm to provide demonstrations and directly teleoperate the remote (right) arm. Once the TP-HSMM model is learned, the remote side (robot) can autonomously execute the hot-stabbing motion, successfully adapting to changing receptacle locations.

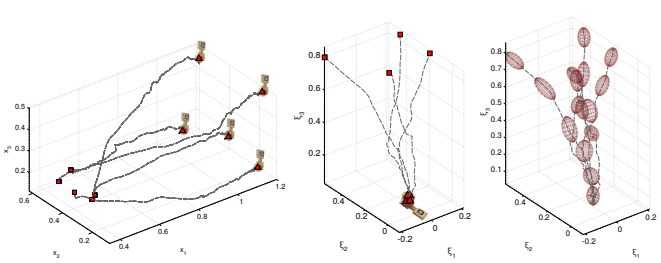


Fig. 6. *Left*: Demonstrations of the motion in the global frame. *Middle*: The motions projected to the task-parametrized frame of the receptacle. *Right*: The TP-HSMM model learned online from the demonstrations in the task-parametrized frame of the receptacle.

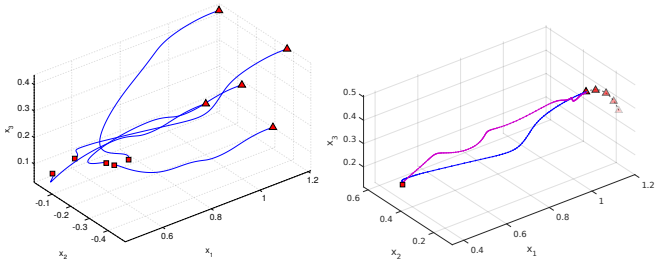


Fig. 7. *Left*: 5 examples of generated motions from our system. Red squares are random starting states and red triangles are random target states. *Right*: Example of generated motions where the target is moved while the motion is performed. Our approach (in blue) can smoothly adapt to the change of target. The ProMP generated motion (in purple) also reaches the set target but with an incorrect approach path.

on the remote side, from random starting points to random targets. Here the starting position is the current end-effector position of the robot arm as directly teleoperated. Once the motion generation begins, the local and remote systems are clutched (states can evolve separately) and the robot executes the task autonomously. Our approach generalizes well even when starting at points far away from the learned model. Typical behavior of the system is to first reach the closest component and then continue through the path outlined by the model. In this aspect the behavior of the system closely resembles that of a virtual fixture/guide. The demonstrations and generated motions are very close from spatial and temporal perspective. Note how the system always keeps a steady vertical/perpendicular approach to the receptacle, as demonstrated. Quantitatively, comparing the system output to the operator's demonstrations resulted to an RMSE of 1.2 ± 0.2cm while the overall success rate of autonomously performing the hot-stabbing motion was 87% over 15 trials. We believe that the unsuccessful trials can be attributed to noise in the marker tracking process (calibration and estimation).

One thing to highlight is that the remote system (robot side) performs the task autonomously. The remote system relies on local feedback and keeps track of the task-relevant variables. This way when the receptacle is moved the system can adapt online to the environment change and successfully execute the task. An example of this scenario is shown in Fig. 7 (*right*). In the context of the underwater ROV, this could correspond to noise in the position of the ROV or sudden sways due to currents or other dynamically changing conditions.

A. Comparison with ProMP

We compare the behavior of our approach with ProMP [12] by analyzing the generated motions in terms of generalization capability, and suitability of generated motions in static and changing conditions. We use the 5 demonstrated motions collected in our trials to train a ProMP for the hot-stabbing task. Training the ProMP model is done in a batch fashion and involves dynamic-time-warping (DTW) of the demonstration and the setting of a regularization term (when estimating Σ_w) that was empirically set to 10^{-3} for our trials to obtain the best performance. The number of the basis functions used to represent the trajectory distribution was set to 15 components (equal to the number of components of the TP-HSMM for fairness of comparison).

We compare the generated motions against the 5 collected demonstrations, performing the regression step conditioned on the given start and target points. This is shown in Fig. 8(*left*) where we see that the generated motions closely follow the demonstrations, resulting in an RMSE of 1.1 ± 0.3cm. Next, we test the ProMP model against the predictions of our approach, i.e., performing the regression conditioned on the same starting and target points as for the situations generated with our framework. We observe that the resulting motions are not as smooth as that of our method (spurious motions sometimes appearing in the start or end of the generated trajectories), which is principally due to the small number of demonstrations used to estimate the ProMP parameters (an inverse Wishart distribution was employed here as prior for better performance). More importantly, we can observe that samples from the ProMP do not exhibit the final approach phase behavior that existed in the demonstrated motions, i.e. the motions reach the target from different –not perpendicular– directions, a crucial step in successfully executing the hot-stabbing motion. The overall impression from the performance of ProMP is that it would require more demonstrations to generalize better as the generated motions degraded substantially when tested further away from the training data. It is important to highlight here that Gaussian conditioning fulfils its role, by starting and ending at the desired position with a continuous transition, but that the adaptation is not well adapted to the changing task requirements, sometimes resulting in an inappropriate approaching phase to the target and an incorrect insertion of the hot-stab. In contrast, the task-parameterized formulation shows better extrapolation behaviors.

Last, we compare the behavior of the ProMP with our approach when the target changes during execution. Our approach can –by design– smoothly handle dynamically changing task parameters, while for ProMP this would amount to changing the target on which the regression is conditioned as the motion is being generated. Fig. 7 (*right*) shows an example of this comparison. Both methods reach the target point, but ProMP reaches the target in a manner that does not result in a successful trial. Indeed, the hot-stab

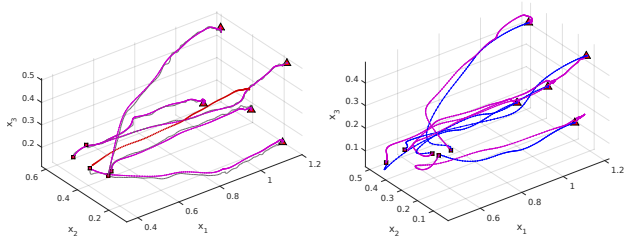


Fig. 8. *Left*: Demonstrations (in grey) used to learn the ProMP model along with reproductions (in purple) using ProMP on the start and target points taken from the demonstrations. The red dotted line represents the mean of the trajectory distribution. *Right*: Motions generated with our system (in blue) and with the ProMP method (in purple), based on the same start and target points.

is not aligned with the receptacle during the later part of the motion.

B. Discussion

The difficulties that we encountered with the ProMP approach is common to all regression-based approaches and not specific to ProMP. By using a probabilistic task-parametrized formulation, our method leverages significant generalization benefits, as learning is performed in a task-local manner, resulting to more efficient use of the demonstration datapoints for generalization. Another benefit of our approach is that it learns online and allows incremental learning of tasks in piecewise steps. This results in a minimal set of model parameters needing to be set in advance, only 2 – the max distance λ and the minimum variance $\hat{\Sigma}$, that are intuitive to set. In addition, no record of data needs to be kept as model building and all estimations are performed online. This makes the proposed approach flexible and suitable for building up a task library during multiple missions. In practice, every time a task is performed by the operator, the task library can be growing and/or refining the ROV skill repertoire.

VII. Conclusion

We presented a general approach for online learning and optimal control of manipulation tasks in a supervisory teleoperation context. We used an online Bayesian nonparametric learning algorithm to build models of manipulation motions encoded as TP-HSMMs that accurately capture the spatiotemporal characteristics of demonstrated motions. We showed how the probabilistic representation can be combined with an MPC controller to generate online control commands in a receding horizon manner that is both robust to noise and changes in the environment. We presented how this framework can be used to automate common and recurring tasks, allowing the operator to focus only on the aspects of the tasks that genuinely require human intervention. We compared against a state-of-the-art approach and demonstrated how our method leverages the task-parametrized formulation to increase generalization, both in terms of extrapolation and online adaptability.

References

[1] J. Gancet, P. Weiss, G. Antonelli, M. F. Pfingsthorn, S. Calinon, A. Turetta, C. Walen, D. Urbina, S. Govindaraj, P. Letier, X. Martinez, J. Salini, B. Chemisky, G. Indiveri, G. Casalino, P. Di Lillo, E. Simetti, D. De Palma, A. Birk, A. Tanwani, I. Havoutis, A. Caffaz, and L. Guilpain, "Dexterous undersea interventions with far distance onshore supervision: the DexROV project," in *IFAC CAMS*, Trondheim, Norway, September 2016, pp. 414–419.

[2] B. Kulis and M. I. Jordan, "Revisiting k-means: New algorithms via Bayesian nonparametrics," in *Proc. ICML*, Edinburgh, Scotland, UK, 2012, pp. 1–8.

[3] S. Park, R. D. Howe, and D. F. Torchiana, "Virtual fixtures for robotic cardiac surgery," in *Proc. MICCAI*, W. J. Niessen and M. A. Viergever, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 1419–1420.

[4] L. B. Rosenberg, "Virtual fixtures: Perceptual tools for telerobotic manipulation," in *Proc. IEEE VRAIS*, Sep 1993, pp. 76–82.

[5] J. J. Abbott, P. Marayong, and A. M. Okamura, "Haptic virtual fixtures for robot-assisted manipulation," in *Proc. ISRR*. Springer Berlin Heidelberg, 2007, pp. 49–64.

[6] F. Rydn, A. Stewart, and H. J. Chizeck, "Advanced telerobotic underwater manipulation using virtual fixtures and haptic rendering," in *2013 OCEANS - San Diego*, Sept 2013, pp. 1–8.

[7] J. Bohren, C. Paxton, R. Howarth, G. D. Hager, and L. L. Whitcomb, "Semi-autonomous telerobotic assembly over high-latency networks," in *Proc. ACM/IEEE HRI*, March 2016, pp. 149–156.

[8] G. Raiola, X. Lamy, and F. Stulp, "Co-manipulation with multiple probabilistic virtual guides," in *Proc. IEEE/RSJ IROS*, Sept 2015, pp. 7–13.

[9] I. Havoutis and S. Calinon, "Learning assistive teleoperation behaviors from demonstration," in *Proc. IEEE SSRR*, Lausanne, Switzerland, October 2016.

[10] W. R. Ferrell and T. B. Sheridan, "Supervisory control of remote manipulation," *IEEE Spectrum*, vol. 4, no. 10, pp. 81–88, Oct 1967.

[11] G. Hirzinger, J. Heindl, K. Landzettel, and B. Brunner, "Multisensory shared autonomy - a key issue in the space robot technology experiment rotex," in *Proc. IEEE/RSJ IROS*, vol. 1, Jul 1992, pp. 221–230.

[12] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proc. IEEE*, vol. 77:2, pp. 257–285, February 1989.

[13] S.-Z. Yu and H. Kobayashi, "Practical implementation of an efficient forward-backward algorithm for an explicit-duration hidden Markov model," *IEEE Trans. on Signal Processing*, vol. 54, no. 5, pp. 1947–1951, 2006.

[14] S. Calinon, A. Pistillo, and D. G. Caldwell, "Encoding the time and space constraints of a task in explicit-duration hidden Markov model," in *Proc. IEEE/RSJ IROS*, San Francisco, CA, USA, September 2011, pp. 3413–3418.

[15] A. Tanwani and S. Calinon, "Learning robot manipulation tasks with task-parameterized semitied hidden semi-Markov model," *IEEE RA-L*, vol. 1, no. 1, pp. 235–242, Jan 2016.

[16] A. Roychowdhury, K. Jiang, and B. Kulis, "Small-variance asymptotics for hidden Markov models," in *Proc. NIPS*, 2013, pp. 2103–2111.

[17] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, "Probabilistic movement primitives," in *Proc. NIPS*. Curran Associates, Inc., 2013, pp. 2616–2624.

[18] S. Calinon, "Robot learning with task-parameterized generative models," in *Proc. ISRR*, 2015.

[19] M. J. A. Zeestraten, I. Havoutis, J. Silvério, S. Calinon, and D. G. Caldwell, "An approach for imitation learning on Riemannian manifolds," *IEEE RA-L*, 2017.

[20] J. Gauvain and C.-H. Lee, "Maximum a posteriori estimation for multivariate gaussian mixture observations of Markov chains," *IEEE Transactions on Speech and Audio Processing*, vol. 2, no. 2, pp. 291–298, Apr 1994.

[21] Total Marine Technology, Website, 2015. [Online]. Available: http://www.tmtrov.com.au/tooling/standard-tooling/tmt-single-port-high-flow-hot-stab

[22] A. Bleicher, "Gulf spill one year later: Lessons for robotics," Website, 2015. [Online]. Available: http://spectrum.ieee.org/robotics/industrial-robots/gulf-spill-one-year-later-lessons-for-robotics#

[23] I. Havoutis, A. Tanwani, and S. Calinon, "Online incremental learning of manipulation tasks for semi-autonomous teleoperation," in *IEEE/RSJ IROS 2016, Workshop on Closed-loop Grasping and Manipulation: Challenges and Progress*, October 2016.