Superhuman Performance of Surgical Tasks by Robots using Iterative Learning from Human-Guided Demonstrations

Jur van den Berg, Stephen Miller, Daniel Duckworth, Humphrey Hu,
Andrew Wan, Xiao-Yu Fu, Ken Goldberg, Pieter Abbeel

***Abstract*— In the future, robotic surgical assistants may assist surgeons by performing specific subtasks such as retraction and suturing to reduce surgeon tedium and reduce the duration of some operations. We propose an apprenticeship learning approach that has potential to allow robotic surgical assistants to autonomously execute specific trajectories with superhuman performance in terms of speed and smoothness. In the first step, we record a set of trajectories using human-guided backdriven motions of the robot. These are then analyzed to extract a smooth reference trajectory, which we execute at gradually increasing speeds using a variant of iterative learning control. We evaluate this approach on two representative tasks using the Berkeley Surgical Robots: a figure eight trajectory and a two handed knot-tie, a tedious suturing sub-task required in many surgical procedures. Results suggest that the approach enables (i) rapid learning of trajectories, (ii) smoother trajectories than the human-guided trajectories, and (iii) trajectories that are 7 to 10 times faster than the best human-guided trajectories.**

I. Introduction

Robotic surgical assistants, such as Intuitive Surgical™'s (Sunnyvale, CA) da Vinci® system are increasingly being accepted in hospitals for endoscopic surgery. Existing hardware is capable of performing delicate surgical procedures. In fact, robotic surgical assistants enable surgeons to overcome major barriers and limitations regarding scaling, accessibility, distance and teamwork. However, many surgical tasks performed with these robotic surgical assistants remain tedious and time consuming because they are tele-operated in a master-slave mode. Intelligent robotic surgical assistants that enable the automation of selected surgical motions and skills could improve patient health by enhancing surgeon performance, reduce tedium and thus medical errors, and reduce costs by reducing operation time.

This paper presents a first step towards the ultimate goal of intelligent surgical assistants. In particular, we present an approach that learns a task from multiple human demonstrations, and learns to execute such tasks with superhuman performance. Specifically, we aim to maximize the *smoothness* and the *speed* with which the tasks are performed. Increased smoothness reduces damage to human tissue and the robotic mechanism, and increased speed reduces operation time.

We assume that the task can be described by a trajectory that the robot should follow through its state space. It is often difficult to mathematically define a unique optimal trajectory for a particular task or even to characterize the set of effective trajectories. However it is possible to record *demonstrations* of effective trajectories by monitoring the trajectories generated by human experts operating the robotic system. Even though a demonstrated trajectory may deviate slightly from the trajectory the expert intended, we assume that *multiple* demonstrations deviate in different ways, and as such implicitly encode the intended trajectory. Using an apprenticeship learning approach [11], combined with the prior knowledge that the trajectory must be smooth, we extract the intended trajectory from the human demonstrations and use it as the *reference trajectory* specifying the particular task.

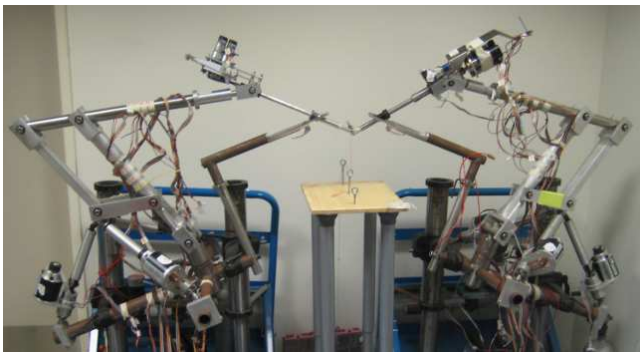


Fig. 1. The Berkeley Surgical Robots performing a knot-tie.

Subsequently, the obtained reference trajectory is a target trajectory for our robots and our approach is able to increase the execution speed significantly while maintaining accuracy. Enabling the robots to execute the target trajectory faster is challenging as it would require an accurate model of the robots' dynamics in the vicinity of the sped-up target trajectory—whereas demonstrations are only available at the original speed. One reasonable approach would be to build a detailed model of the robots, which for accurate execution would need to include trajectory specific properties such as stiction and hysteresis effects [8]. In this paper we show that, rather than going through this detailed modeling phase, it is possible to *learn* to perform faster trajectories through gradually speeding up the trajectory. This seems to work well even with only an approximation of the true robot dynamics.

We implemented our approach on the Berkeley Surgical Robots (see Fig. 1), and applied it to two representative tasks, among which knot-tying. Our experimental results indicate that we can increase the speed to up to 10 times the average speed of the demonstrations, and perform executions with a quality exceeding that of the demonstration trajectories.

This work was supported in part by NSF Award 0905344 and NIH Award 1R01EB-006435-01A1.

The authors are with the University of California at Berkeley, Berkeley, CA, USA. E-mail: {berg, sdavidmiller, duckworthd, humphrey.hu, andrewwan, xyfu, goldberg}@berkeley.edu, pabbeel@cs.berkeley.edu.

II. Related Work

There is a large body of literature on modeling human movement and extracting human strategies for use by robots (see [7] for a literature review). Human skill has been modeled using hidden Markov models [14], [15], neural networks [4], [18], and fuzzy sets [26], [31]. In [11], an apprenticeship learning approach is introduced that has enabled a quadruped robot to traverse challenging, previously unseen terrain [19] and helicopters to autonomously perform aerobatic maneuvers. Also, there has been some foundational work on the modeling of surgical skill [21], [29], [22]. In [24], [23], Mayer et al. demonstrated a surgical robot learning knot-tying from a demonstration.

Other techniques plan medical motions without learning, and have the robot perform them autonomously [2], [5], [32]. In the work of [33], a motion is carefully tuned to enable a three-fingered robot to tie a knot at an extremely high speed. In [8], the time-optimal motion along a specified path is computed given an accurate model of the robot dynamics.

Our work extends the above learning-by-demonstration approaches, as we aim to increase the execution speed even though we do not have an accurate dynamics model of the robot and the demonstrations are only available at the original speed. We complement the work of [33], as our approach can teach the robot new tasks without the need for precise specification of the task trajectory.

Our approach builds off of [11] to learn a reference trajectory specifying a task from human demonstrations. To increase the execution speed, we use a variant of iterative learning control [9]. ILC has been studied widely in robotics [25]. The work of [20] suggests that only global knowledge of the robot dynamics is necessary to be able to iteratively improve control. The approach of [1] learns a local correction to the dynamics model by adding a time-dependent bias term. Our approach borrows from ideas of [9] by iteratively adapting the target trajectory based on the observed error.

III. Problem Definition and Global Approach

In this section, we formally define the problem we discuss in this paper, and sketch the outline of our approach.

A. Problem Definition

The problem we discuss in this paper is formally defined as follows. We are given a robotic system whose *state* is described by a vector $\mathbf{x}$ and whose *control input* is described by a vector $\mathbf{u}$. We assume that the *dynamics model* of the robot, which describes the state of the robot in the next time step as a function of the state and the control input in the current time step, is given in the linear form

$$\mathbf{x}_{t+1} = A\mathbf{x}_t + B\mathbf{u}_t + \mathbf{w}_t, \quad \mathbf{w}_t \sim \mathcal{N}(\mathbf{0}, P), \tag{1}$$

where A and B and P are matrices of the appropriate dimension, and $\mathbf{w}_t$ is an independent noise term drawn from a Gaussian distribution with zero mean and covariance P that captures process noise and unmodeled physical effects on the robot. The precise composition of the state, control input and the matrices for the robotic system we use for our experiments is discussed in Section VI.

Further, we are given M demonstration trajectories (obtained by a human controlling the robot) of the robotic task we wish to perform. Each trajectory $\mathbf{y}^j$, with $j \in 1 \dots M$, has duration T^j, and is a function of time $t \in [0, T^j]$ to a state $\mathbf{x}^j(t)$ and control input $\mathbf{u}^j(t)$:

$$\mathbf{y}^j(t) = \begin{bmatrix} \mathbf{x}^j(t) \\ \mathbf{u}^j(t) \end{bmatrix}, \tag{2}$$

Our goal is to "learn" a *reference trajectory* $\mathbf{z}$ from the given demonstration trajectories that is consistent with the dynamics model of Equation (1). We define this trajectory as a discrete sequence of states $\mathbf{x}^*$ and control inputs $\mathbf{u}^*$ of length $N+1$ (such that there are N "steps"), and denote it

$$\mathbf{z}_t = \begin{bmatrix} \mathbf{x}^*_t \\ \mathbf{u}^*_t \end{bmatrix}, \tag{3}$$

where $t \in 0 \dots N$. Initially, we set the *time step* Δt, which denotes the actual difference in time between $\mathbf{z}_{t+1}$ and $\mathbf{z}_t$, to some desired value and set $N = \frac{1}{M\Delta t} \sum_{j=1}^{M} T^j$, such that the duration $N\Delta t$ of the reference trajectory $\mathbf{z}$ is equal to the average of the durations of the example trajectories.

Our objective is to (i) learn a smooth trajectory $\mathbf{z}$ from the demonstrations, and then (ii) decrease Δt (while keeping N constant) to have the robot successfully execute trajectory $\mathbf{z}$ at superhuman speeds.

B. Outline of our Approach

Our approach to the above problem can roughly be divided into two steps, outlined as follows:

1. Learning from human demonstrations. First, we use the demonstration trajectories $\mathbf{y}$ as *observations* to the (unobserved) reference trajectory $\mathbf{z}$ in a *Kalman smoother* [28], [17], which will produce (Gaussian) distributions of the states along the reference trajectory $\mathbf{z}$. The Kalman smoother is used as part of the *EM-algorithm* [13], [17], which iteratively and alternatingly infers the distributions of the reference trajectory given the current model parameters, and updates the model parameters by maximizing their likelihood given the current distributions of the reference trajectory. As the demonstration trajectories may not be perfectly temporally aligned, we include a *time-warping* step [30], [27] in the EM-algorithm that time-aligns the demonstration trajectories as well as possible with the current reference trajectory. In Section IV, we discuss learning from human demonstrations in detail.

2. Trajectory execution at superhuman speeds. Secondly, we aim to execute the reference trajectory $\mathbf{z}$ on the robot, and speed it up. We use an *LQR controller* [3], [6], [16] to execute the trajectory, for it gives the optimal control policy given a linear dynamics model and a quadratic cost function (which penalizes deviations from the trajectory and non-smoothnesses in its execution). However, due to the inaccuracy of our dynamics model, the resulting execution may not be perfect. Therefore, using a variant of *iterative learning control* [9], [25], we adapt the target trajectory based

on the observed error, and re-execute the LQR controller on this adjusted target trajectory. We repeat this until convergence. Then, the execution speed is slightly increased, and the aforementioned process repeats. Our algorithm continues to increase the execution speed as long as the quality of the execution remains above a pre-defined threshold. In Section V, we discuss in detail how we achieve superhuman execution speeds.

IV. Learning from Human Demonstrations

In this section we describe how we learn the reference trajectory $\mathbf{z}$ from the demonstration trajectories $\mathbf{y}$.

A. Stochastic Model

In our stochastic model, we the use the following dynamics model for $\mathbf{z}$:

$$\mathbf{z}_{t+1} = \begin{bmatrix} A & B \\ 0 & I \end{bmatrix} \mathbf{z}_t + \mathbf{w}_t^*, \quad \mathbf{w}_t^* \sim \mathcal{N}(\mathbf{0}, \begin{bmatrix} P & 0 \\ 0 & Q \end{bmatrix}), \quad (4)$$

where matrices A, B and P are as given in Section III-A, and $\mathbf{w}_t^*$ is a noise term drawn from a zero-mean Gaussian distribution. This model extends the model of Equation (1), and assumes that the control input $\mathbf{u}^*$ does not change between time steps. The only fluctuation allowed comes from the noise term and is determined by matrix Q. The smaller the fluctuation in control input, the *smoother* the trajectory. Hence, to force the trajectory $\mathbf{z}$ to be smooth, we set Q as a diagonal matrix with small (positive) values on the diagonal.

We use the demonstration trajectories $\mathbf{y}$ as (noisy) observations of the reference trajectory $\mathbf{z}$, giving the following observation model:

$$\begin{bmatrix} \mathbf{y}^1(\tau_t^1) \\ \vdots \\ \mathbf{y}^M(\tau_t^M) \end{bmatrix} = \begin{bmatrix} I \\ \vdots \\ I \end{bmatrix} \mathbf{z}_t + \mathbf{v}_t, \mathbf{v}_t \sim \mathcal{N}(\mathbf{0}, \begin{bmatrix} R^1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & R^M \end{bmatrix}), \quad (5)$$

where $\mathbf{v}_t$ a noise term drawn from a zero-mean Gaussian distribution, and τ_t^j is the *mapping* of time t in trajectory $\mathbf{z}$ to the corresponding time in trajectory $\mathbf{y}^j$.

We use the time mappings τ_t^j to *temporally align* the demonstration trajectories to the reference trajectory $\mathbf{z}$, as the given demonstrations may contain temporal variations; for instance not all parts of the trajectory are executed at the same pace among the demonstration trajectories. The time mappings τ_t^j are initially unknown, but we assume:

$$\tau_0^j = 0, \quad \tau_N^j = T^j, \quad \frac{1}{v} \leq \frac{\tau_{t+1}^j - \tau_t^j}{T^j} N \leq v, \quad (6)$$

where T^j is the duration of trajectory $\mathbf{y}^j$, N the number of steps in the reference trajectory $\mathbf{z}$, and v (with $v > 1$) is a bound on how fast time progresses along the demonstration trajectory *relative* to the reference trajectory. That is, the time mapping is done such that the endpoints of the trajectories match, and such that at any given point along the demonstration trajectory, time moves at least v times as slow and at most v times as fast as along the reference trajectory.

The covariance matrices R^j encode the "weight" that demonstration trajectory $\mathbf{y}^j$ has in shaping the reference trajectory $\mathbf{z}$, and are initially unknown as well.

B. EM Algorithm

In the above model, the reference trajectory $\mathbf{z}$, the matrices R and the time-mappings τ are unknown. We compute them using the EM (Expectation Maximization)-algorithm by optimizing their joint likelihood. The EM-algorithm alternatingly computes Gaussian distributions Z of the reference trajectory using a *Kalman smoother* given the current values for R and τ, and then updates R and τ by maximizing their *likelihood* with respect to the most recently computed distributions Z of the reference trajectory (see Algorithm 1). The Kalman smoother constitutes the E-step of the EM-algorithm, and updating R and τ constitutes the M-step.

Let Z_t denote the unconditional Gaussian distribution of the state $\mathbf{z}_t$ of the reference trajectory $\mathbf{z}$ at time t (which can be computed recursively using Equation (4) given an initial distribution Z_0). Then, assuming R and τ are given, the Kalman smoother will compute the *posterior* distributions $Z_t|\mathbf{y}$ of the reference trajectory given the demonstration trajectories $\mathbf{y}$. This is done using Equation (5) in an efficient two pass linear-time algorithm [28]. We will not describe the Kalman smoother in full detail here, but refer to [17] for an excellent treatise. For notational simplicity, we refer to the posterior distributions computed by the Kalman smoother as Z_t in the remainder of this paper. The mean of posterior distribution Z_t is denoted $\bar{\mathbf{z}}_t$. Note that the distributions computed by the Kalman smoother depend on the values of R and τ.

In the M-step, the matrices R and time mappings τ are updated by maximizing the *expectation* of the (log-) likelihood of R and τ, given the demonstration trajectories $\mathbf{y}$ and the distributions Z computed by the Kalman smoother:

$$\max_{R,\tau} \mathbb{E}_Z(\ell(R, \tau | Z, \mathbf{y})) = \max_{R,\tau} \mathbb{E}_Z(\log \mathbb{P}(Z, \mathbf{y})) = \max_{R,\tau} \sum_{j=1}^{M} \sum_{t=0}^{N} \mathbb{E}_Z(\log \mathbb{P}(\mathbf{y}^j(\tau_t^j) | Z_t)). \quad (7)$$

As the "log" is eliminated against the "exp" in the Gaussian probability-density function, the expectations can be brought inside, and a closed form update rule for R (with fixed τ) can be derived (see [17] for more details). To optimize τ (with fixed R), we use *dynamic time warping*, which is a dynamic programming algorithm that we will discuss in the next subsection.

The above process repeats until convergence. The EM-algorithm is *guaranteed* to converge to a local optimum for any initialization of R, and for any initialization of τ. As we have no prior knowledge about the relative quality of the given demonstration trajectories, all R^j should initially be equal. So, before performing the first E-step (the Kalman smoother), we set R^j (for $j \in 1 \dots M$) equal to the identity matrix I. We can initialize τ by assuming that the time along the reference trajectory $\mathbf{z}$ corresponds proportionally to the time along the demonstration trajectories $\mathbf{y}^j$, i.e. $\tau_t^j = \frac{t}{N} T^j$.

C. Dynamic Time Warping

As part of the EM-algorithm, the time mappings τ are updated by optimizing their likelihood given distributions

Algorithm 1 $\mathbf{z} \leftarrow$ LEARNINGFROMDEMONSTRATIONS$(\mathbf{y})$

1: Initialize $R^j = I$, and $\tau_t^j = t\frac{T^j}{N}$.
2: **while** not converged **do**
3: $\quad Z \leftarrow$ KALMANSMOOTHER$(\mathbf{y}, R, \tau)$.
4: $\quad R \leftarrow \arg\max_R \mathbb{E}_Z(\ell(R|Z, \mathbf{y}))$.
5: $\quad \tau^j \leftarrow \arg\max_{\tau^j} \mathbb{E}_Z(\ell(\tau^j|Z, \mathbf{y}))$. // dynamic time warping
6: **return** modes $\bar{\mathbf{z}}$ of distributions Z as reference trajectory.

Z of the reference trajectory, and fixed matrices R (see Equation (7)). We compute the new τ_t^j for each trajectory $\mathbf{y}^j$ independently using *Dynamic Time Warping*, which is a dynamic programming algorithm that has been used as well for speech recognition [30], biological sequence alignment [27], etc.

For the algorithm, the time-axis of the demonstration trajectory $\mathbf{y}^j$ is discretized into kN steps of duration $\Delta t^j = \frac{T^j}{kN}$, where $k > 1$ is an integer specifying the resolution of the algorithm. Now, for each $t \in 0 \dots N-1$, we let

$$\tau_{t+1}^j = \tau_t^j + a_t \Delta t^j, \qquad a_t \in \{\lceil k/v \rceil, \dots, \lfloor kv \rfloor\}, \tag{8}$$

where a_t determines how fast time progresses along the demonstration trajectory relative to the reference trajectory at time t; when $a_t = k/v$, the demonstration trajectory moves "v times as slow" as the reference trajectory; when $a_t = k$ both trajectories have equal pace; and when $a_t = kv$, the demonstration trajectory moves "v times as fast". In general, the demonstration trajectory moves "a_t/k times as fast" as the reference trajectory at time t.

Equation (8) defines a graph in a 2-D grid of points (t, τ_t^j) through which we have to find a "path" from point $(0, 0)$ to point (N, T^j) that maximizes:[1]

$$\max_{\tau^j} \sum_{t=0}^{N} \mathbb{E}_Z(\log \mathbb{P}(\mathbf{y}^j(\tau_t^j)|Z_t)) = \max_{\tau^j} \sum_{t=0}^{N} \log \mathbb{P}(\mathbf{y}^j(\tau_t^j)|\bar{\mathbf{z}}_t). \tag{9}$$

As the inner probability can be computed by evaluating point $\mathbf{y}^j(\tau_t^j)$ in the probability-density function of Gaussian $\mathcal{N}(\bar{\mathbf{z}}_t, R^j)$ (this follows from Equation (5)), the optimal τ^j can be found using dynamic programming (for instance Dijkstra's shortest path algorithm).

The time-complexity of the algorithm is $O(N \cdot kN \cdot (\lfloor kv \rfloor - \lceil k/v \rceil)) = O(N^2 k^2 v)$. To save computation time, we use the smallest possible values of $k = 2$ and $v = 2$, which, as experiments indicate, gives surprisingly good results.

V. EXECUTION AT SUPERHUMAN SPEEDS

In this section we describe how we execute the reference trajectory $\mathbf{z}$ as learned above at superhuman speeds.

A. LQR Controller

Given a target trajectory $\hat{\mathbf{x}}$, we aim to control the robot to follow trajectory $\hat{\mathbf{x}}$ as closely as possible. For this, we use an *LQR controller*, which is a standard control technique that is ideally suited for our context; given a linear(ized) dynamics model and a quadratic cost function, an LQR controller gives the *optimal* control policy.[2]

As we wish the robot to closely follow the given target trajectory $\hat{\mathbf{x}}$, and at the same time achieve *smooth* control, the LQR controller should penalize the *deviation* from the target trajectory, the *magnitude* of the applied control input, as well as the *fluctuation* in the control input. To achieve this using the standard LQR derivations, we adapt the dynamics model of Equation (4) into an *error* dynamics model [3]: The state vector of the adapted model is comprised of the state *error* $\mathbf{x}_t - \hat{\mathbf{x}}_t$, the control input $\mathbf{u}_t$, and the number 1 to allow for intercept terms. And the control input vector of the model is the *change* $\Delta\mathbf{u}_t$ in the actual control input applied between the current time step and the next. We thus define:

$$\tilde{\mathbf{z}}_t = \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t \\ 1 \end{bmatrix}, \qquad \Delta\mathbf{u}_t = \mathbf{u}_{t+1} - \mathbf{u}_t, \tag{10}$$

where $\mathbf{x}_t$ and $\mathbf{u}_t$ are the actual states observed and control inputs applied, respectively, during control at time t. This gives the following dynamics model:

$$\tilde{\mathbf{z}}_{t+1} = \begin{bmatrix} A & B & \mathbf{c}_t \\ 0 & I & \mathbf{0} \\ \mathbf{0}^T & \mathbf{0}^T & 1 \end{bmatrix} \tilde{\mathbf{z}}_t + \begin{bmatrix} 0 \\ I \\ \mathbf{0}^T \end{bmatrix} \Delta\mathbf{u}_t, \tag{11}$$

where $\mathbf{c}_t$ is a time-dependent intercept term defined as $\mathbf{c}_t = A\hat{\mathbf{x}}_t - \hat{\mathbf{x}}_{t+1}$.[3]

The optimal controls $\Delta\mathbf{u}_t$ are found by minimizing the quadratic cost function

$$\min_{\Delta\mathbf{u}} \left(\tilde{\mathbf{z}}_N^T G \tilde{\mathbf{z}}_N + \sum_{t=0}^{N-1} (\tilde{\mathbf{z}}_t^T G \tilde{\mathbf{z}}_t + \Delta\mathbf{u}_t^T H \Delta\mathbf{u}_t) \right), \tag{12}$$

for cost matrices G and H. This gives the control policy $\Delta\mathbf{u}_t = L_t\tilde{\mathbf{z}}_t$, for *feedback matrix* L_t that is (pre-)computed using standard LQR derivations (see [3], [6] for more details). Note that $\tilde{\mathbf{z}}_t$ is fully observable during execution, as long as the state $\mathbf{x}_t$ is observable.

B. Improving Quality using Iterative Learning Control

As our dynamics model (see Equations (4) and (11)) is a linear approximation of the true dynamics, the above LQR controller, when run on the reference trajectory $\mathbf{x}^*$ learned from the demonstrations, may not give convincing results. We see that the observed trajectory $\mathbf{x}$, even though it globally follows the reference trajectory $\mathbf{x}^*$, sometimes significantly deviates from the reference trajectory. To improve the quality of the observed trajectory $\mathbf{x}$ (i.e. to make it more similar to the reference trajectory $\mathbf{x}^*$), we iteratively manipulate the *target* trajectory $\hat{\mathbf{x}}$ on which the LQR controller is executed. Initially, the target trajectory is the reference trajectory.

In particular, if the observed trajectory $\mathbf{x}$ deviates from the reference trajectory $\mathbf{x}^*$ by a vector $\mathbf{x} - \mathbf{x}^*$, we rerun the LQR controller on a new target trajectory $\hat{\mathbf{x}} - (\mathbf{x} - \mathbf{x}^*)$, such that

[1] The equality holds as Z only appears in linear form in the derivation.

[2] We also experimented with a PID controller, but as it lacks lookahead along the trajectory, we were unable to achieve good performance.

[3] Spelling out Equation (11) gives $\left[\begin{smallmatrix} \mathbf{x}_{t+1} \\ \mathbf{u}_{t+1} \end{smallmatrix}\right] = \left[\begin{smallmatrix} A & B \\ 0 & I \end{smallmatrix}\right] \left[\begin{smallmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{smallmatrix}\right] + \left[\begin{smallmatrix} 0 \\ I \end{smallmatrix}\right] \Delta\mathbf{u}_t$.

if the deviation with respect to the target trajectory would remain the same at each point along the trajectory, the new observed trajectory would match the reference trajectory $\mathbf{x}^*$. In other words, we "mirror" the deviation of the observed trajectory from the reference trajectory in the current target trajectory to form a new target trajectory. We repeat this process a number of times until the observed trajectory $\mathbf{x}$ converges to the reference trajectory $\mathbf{x}^*$. Algorithmically, this looks as follows (with parameter $0 < \alpha \leq 1$):

Algorithm 2 $\hat{\mathbf{x}} \leftarrow$ ITERATIVELEARNINGCONTROL$(\mathbf{x}^*, \hat{\mathbf{x}})$

1: **repeat**
2: $\quad \mathbf{x} \leftarrow$ EXECUTELQR$(\hat{\mathbf{x}})$.
3: $\quad \hat{\mathbf{x}} \leftarrow \hat{\mathbf{x}} - \alpha(\mathbf{x} - \mathbf{x}^*)$.
4: **until** $\mathbf{x} \approx \mathbf{x}^*$
5: **return** $\hat{\mathbf{x}}$

The input of this algorithm is the reference trajectory $\mathbf{x}^*$ and an initial target trajectory $\hat{\mathbf{x}}$ (which should be the reference trajectory $\mathbf{x}^*$ if one has no prior information); the output is the optimized target trajectory. The parameter α determines the "learning rate" of the algorithm. The larger α (i.e. the closer to 1), the faster the algorithm converges. However, if α is too large, the resulting trajectories $\mathbf{x}$ may oscillate around the reference trajectory $\mathbf{x}^*$. Our experiments suggest that a value of $\alpha = 0.4$ works well in our case.

C. Speeding-up the Trajectory Execution

The execution speed of the trajectory $\mathbf{x}^*$ is fully determined by the value of the time step Δt, if we keep the number of steps N in the trajectory constant. Let us denote the speed-up factor by s. For $s = 1$, the time step Δt has its initial value for which the reference trajectory $\mathbf{x}^*$ was learned using Algorithm 1. Let us denote this initial time step by Δt_0. So, to increase the execution speed s, the time step is decreased (note that the trajectory $\mathbf{x}^*$ itself, which is a constant-length sequence of states, does *not* change when the execution speed is increased):

$$\Delta t = \Delta t_0 / s. \tag{13}$$

When the time step is decreased, the matrices A and B of the dynamics model of Equation (1) change. If the entries of these matrices are clear functions of Δt, these matrices can be directly updated. Otherwise, let A_0 and B_0 be the given initial matrices of the dynamics model for the initial time step Δt_0. Then, one can compute A and B for increased execution speed s as:

$$\begin{bmatrix} A & B \\ 0 & I \end{bmatrix} = \begin{bmatrix} A_0 & B_0 \\ 0 & I \end{bmatrix}^{1/s}. \tag{14}$$

Further, as we may expect the control input fluctuations to increase (proportionally) with the execution speed, the cost matrix H in the LQR penalty function of Equation (12) should be updated as well. Let H_0 be the initial cost matrix for the normal speed trajectory, then for increased speed s:

$$H = H_0 / s^2. \tag{15}$$

We use the above update rules to increase the execution speed. In principle, we could set the execution speed at any desired high value, and run the iterative learning control algorithm (Algorithm 2) on trajectory $\mathbf{x}^*$. However, as we will show in Section VII, our experiments indicate that this causes Algorithm 2 to converge poorly. Much better results are obtained if we *gradually* increase the execution speed in an iterative process:

Initially, the reference trajectory $\mathbf{x}^*$ is used as target trajectory, and the execution speed is set to the average of the demonstration trajectories (i.e. $s = 1$). Then, Algorithm 2 is executed, which returns the optimal target trajectory $\hat{\mathbf{x}}$ for $s = 1$. In each subsequent iteration, the speed is slightly increased and the optimal target trajectory $\hat{\mathbf{x}}$ of the previous iteration is used as *initial* target trajectory for the current iteration. This process continues as long as Algorithm 2 converges and produces trajectories that are of sufficient quality when executed (see Algorithm 3).

Algorithm 3 SUPERHUMANEXECUTION$(\mathbf{x}^*)$

1: $\hat{\mathbf{x}} \leftarrow \mathbf{x}^*$.
2: **do**
3: $\quad \hat{\mathbf{x}} \leftarrow$ ITERATIVELEARNINGCONTROL$(\mathbf{x}^*, \hat{\mathbf{x}})$.
4: $\quad$ Increase execution speed: $s \leftarrow s + 1$.
5: $\quad$ Update matrices A, B, and H and Δt for new speed s.
6: **while** resulting trajectory is of sufficient quality.

VI. IMPLEMENTATION DETAILS

In this section we describe how we implemented the above approach on our specific robotic setup.

A. Robotic Setup

In our setup we use two identical Berkeley Surgical Robots [10] that each provide an "arm" in a surgical workspace. For each robot, the surgical arm is kinematically constrained to pivot around a fixed point, which can be imagined as the point where the robot enters a patient's body. The tip of the arm can freely be moved in 3-D space, and has a fully controllable gripper attached to it (see Fig. 2 for the kinematic model).

Each robot is controlled by seven servo motors; four motors are mounted on the arm itself and control the pitch, roll and gross rotation, as well as the jaws of the gripper. Another three motors, called the base motors, control – through a kinematic chain– the 3-D position in space of the tip of the robot arm. The motors contain accurate sensors to determine their current configuration, and each of the motors can individually be controlled by applying a voltage to it. On a higher level, the robots can be manually operated using a pair of master controllers. The robots then copy the motions that are applied to the masters.

B. Inferring a Dynamics Model

In order for our algorithm to work, we need a dynamics model of the robot as given in Equation (1). As is the case with any real-world system, a truly accurate dynamics

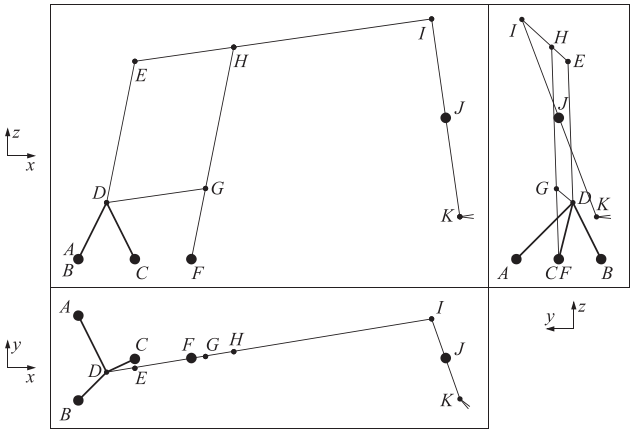


Fig. 2. The kinematic model of our robots (side view, front view and top view – gravity points in the negative z-direction). Points A, B, C, F and J are fixed points in space. Three base motors control the length of the sides AD, BD and CD of the tetrahedron $ABCD$ such that point D can freely be positioned in space. The points D, E, F, G, H and I lie in a common plane parallel to the z-axis. The point J is a fixed pivot point that can be imagined as the point where the robot enters a patient's body for surgical tasks. The segment IK has constant length and can slide through point J. Four motors placed at point I control the roll, pitch and gross rotation of the tip K, as well as the jaws of the gripper. As a result the 6-D position and orientation of point K can fully be controlled by the robot.

model is complex. Apart from the input voltage, many other (external) effects influence the motion of the robot [1].

In our case it appeared that particularly the effect of *gravity* is of substantial significance. Gravity is pulling at all parts of the robot, which, through hinges and pivots in the kinematics, has complex effects on the (base) motors of the robot. In most configurations of the robot, voltages need to be applied to the (base) motors in order to keep the robot in its current configuration. We call this voltage the *gravity voltage* and denote it $\mathbf{g}(\mathbf{q})$ for configuration $\mathbf{q}$. To approximate this function, we measured the voltages applied to the motors in a dense sample of configurations of the robot, and fitted a quadratic polynomial to the data using the least mean squared error technique (we note that there are other techniques for compensating gravity, see e.g. [12]).

With gravity factored out, it turns out that a linear dynamics model suffices for our purposes: The state $\mathbf{x}$ of the robot is the vector $\mathbf{q}$ of the encoder positions of its motors (i.e. the speed is not included in the state), and the control input $\mathbf{u}$ (as used by the LQR-controller) is the vector of the *gravity-adjusted* voltages applied to each motor:

$$\mathbf{x} = \mathbf{q}, \qquad \mathbf{u} = \mathbf{v} - \mathbf{g}(\mathbf{q}), \tag{16}$$

where $\mathbf{v}$ is the vector of the *actual* voltages applied to the motors. Further, we treat each motor independently with the following dynamics:

$$x_{t+1}^i = x_t^i + b^i \Delta t u_t^i + w_t^i, \quad w_t^i \sim \mathcal{N}(0, \sigma_i^2) \tag{17}$$

where x_t^i is the state (i.e. configuration) of motor i at time t, u_t^i the gravity-adjusted voltage applied to motor i at time t, and w_t^i a noise term. That is, we let the applied voltage directly (and linearly) relate to the speed of the motor through the coefficient b^i. This means that in the form of Equation (1), A is the identity matrix I, B is a diagonal matrix with entry $B_{ii} = b^i \Delta t$, and P is a diagonal matrix with entry $P_{ii} = \sigma_i^2$.

In order to estimate the parameters b^i and σ_i^2, we recorded a trajectory $\left[\begin{smallmatrix}\mathbf{x}_t \\ \mathbf{u}_t\end{smallmatrix}\right]$ (for $t \in 0 \ldots T$) that is representative of the robot's dynamics, through human operation of the master controls of the robot. Then, the coefficients b^i are computed by fitting the model as closely as possible to the data from the recorded trajectory using the least mean squared error technique, i.e. by minimizing

$$\min_B \sum_{t=0}^{T-1} \left\| \mathbf{x}_{t+1} - (A\mathbf{x}_t + B\mathbf{u}_t) \right\|^2. \tag{18}$$

Subsequently, the variances σ_i^2 are computed as:

$$P = \sum_{t=0}^{T-1} \frac{(\mathbf{x}_{t+1} - (A\mathbf{x}_t + B\mathbf{u}_t))(\mathbf{x}_{t+1} - (A\mathbf{x}_t + B\mathbf{u}_t))^T}{T}. \tag{19}$$

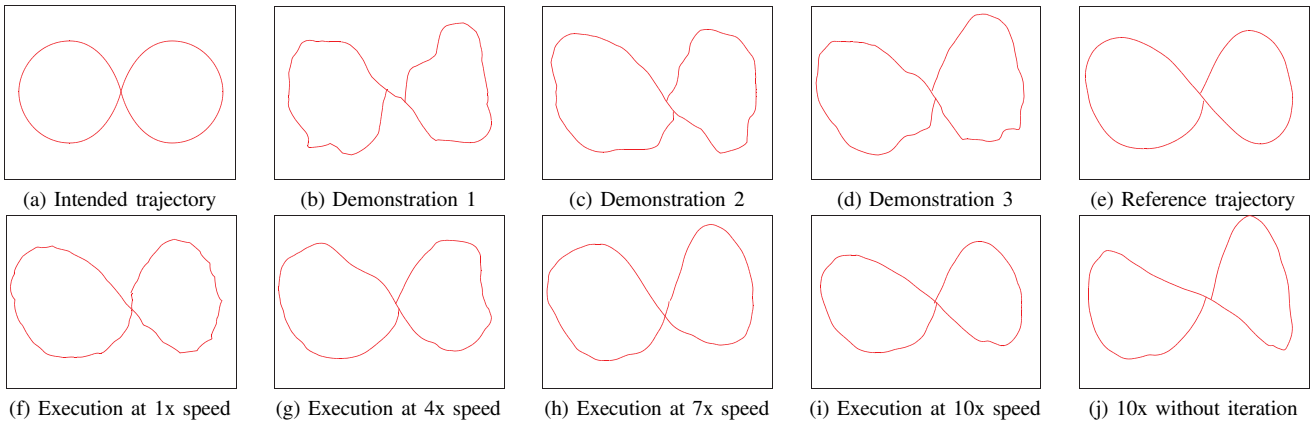
VII. Experimental Results

We implemented the approach on our robots and experimented with it on two representative tasks. In the first experiment, a magnetic pen is attached to the tip of the arm of one of the robots such that the robot can write on a magnetic write-board. The second experiment involves tying a knot in a surgical thread. Knot-tying is a task that is frequently performed during surgical operations.

A. Drawing Figures on a Magnetic Write-Board

To have the robot draw figures on a magnetic write-board, we attached a magnetic pen to the tip of the robot. The pen is attached on a slider such that gravity can pull the pen down until it hits the writing surface. This relaxes the constraints on the z-position of the tip of the robot arm.

The objective in this experiment was to write a perfectly symmetric figure '8' on the writing surface (see Fig. 3(a)). Through manual control of the robot, we created 3 demonstration trajectories, which had an average duration of 12.5 seconds. Because manually controlling the robot is not easy, these demonstration trajectories were far from perfect (see Figs. 3(b-d)). Nonetheless, our learning from human demonstrations algorithm is able to capture the essence of the demonstration trajectories, and learns a trajectory that would smoothly write the figure '8' on the board (see Fig. 3(e)). Subsequently, we executed the learned trajectory at gradually increasing execution speeds using our approach of Section V (the execution speed was increased by one factor in each iteration, i.e. $s = s + 1$). For each execution speed, it took on average 3 iterations of Algorithm 2 to converge to an optimal target trajectory. As can be seen in Figs. 3(f-i), even at 10 times the normal (i.e. human) execution speed, the robot is able to draw a better figure than most of the human demonstrations. If, on the other hand, we would not gradually increase the speed, but run Algorithm 2 directly at $s = 10$, the quality of the resulting trajectory is hardly better than that of the demonstrations (see Fig. 3(j)).



(a) Intended trajectory (b) Demonstration 1 (c) Demonstration 2 (d) Demonstration 3 (e) Reference trajectory

(f) Execution at 1x speed (g) Execution at 4x speed (h) Execution at 7x speed (i) Execution at 10x speed (j) 10x without iteration

Fig. 3. Results of the first experiment. (a) The objective is to draw a perfectly symmetric figure '8' on a magnetic write-board. (b, c, d) Three human demonstrations. (e) The drawing that would result from the learned reference trajectory if executed perfectly. (f, g, h, i) The drawing resulting from executing the reference trajectory for gradually increasing execution speeds — 1x, 4x, 7x, and 10x the average speed of the demonstrations, respectively. (j) The drawing resulting when the execution speed is not gradually increased, but directly initialized at 10x.

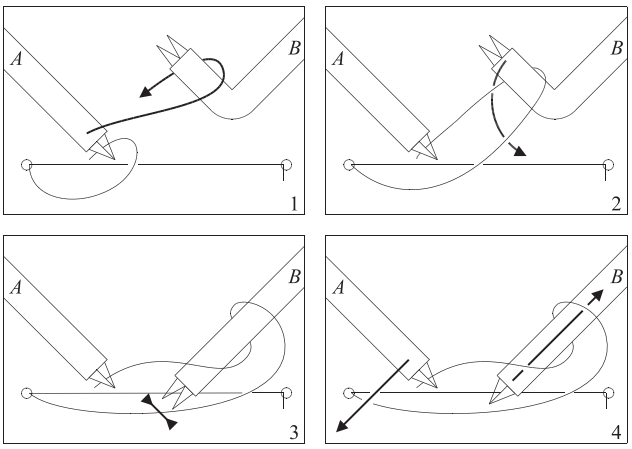


Fig. 4. The knot tie consists of three stages. In the first stage (1), robot A loops the thread around the gripper of robot B. In the second stage (2, 3), robot B grasps the thread and closes its grippers. In the third stage (4), both robot arms are moved away from each other to tighten the knot.

B. Knot Tying

In the second experiment we let the robots tie a knot in a thread around a ring. A similar procedure needs to be performed frequently during surgical operations to affix a suture to human tissue. In our experiment, the ring takes the role of the tissue to which the thread is tied. Knot-tying is a challenging task that involves both robots (let us call them A and B). We assume that initially the thread is going through the ring, and that robot A is holding one end of the thread (see Fig. 4). The knot-tie task can be broken down into three stages: in the first stage robot A loops the thread around the gripper of robot B; in the second stage robot B grasps the other end of the thread; and in the third stage the arm tips of both robots are moved away from each other to tighten the knot around the ring (unfortunately, in our experiments the limited reach of each of the robot arms prohibited us from fully tightening the knot). As we do not model the physical behavior of the thread in our approach, we use another ring and a weight to make sure that the thread is taut between the two rings, such that the position where robot B must grasp the thread is predictable.

TABLE I

RESULTS OF ITERATIVE LEARNING FOR THE KNOT-TIE EXPERIMENT

Deviation in mm compared to reference trajectory

speed	iteration 1	iteration 2	iteration 3
1x	1.143	1.306	1.226
2x	3.342	n/a	n/a
3x	6.402	5.408	4.765
4x	7.685	6.615	6.442
5x	9.192	8.004	7.090
6x	9.772	8.161	6.920
7x	8.601	7.878	n/a

Given that the grasp position is predictable, the motion of robot B is simple compared to the motion of robot A; it only needs to move back and forth. Therefore, we pre-programmed the motion of robot B, and have it executed by a PID controller. For the motion of robot A we generated 5 demonstrations through human control of the robot. Each demonstration consists of two trajectories; one for the first stage and one for the third stage. The average duration of each of the parts of the demonstrations was 7.6 and 5.6 seconds, respectively. On each of the two parts we ran our algorithm to learn a smooth reference trajectory, and stitched them together afterwards with a pause in between during which B can perform the second stage of the knot-tie. Subsequently, we executed the learned trajectory at gradually increasing execution speeds (the motion of B is not sped-up, so the duration of the second stage is constant). As above, we increase the speed by one factor in every iteration, and it took on average 3 iterations of Algorithm 2 to produce an optimal target trajectory for each execution speed.

In Table I, we show the average deviation (the square root of the mean squared error) in mm of the resulting executions with respect to the reference trajectory for each execution speed and for each iteration of Algorithm 2. The results suggest that the quality of the trajectory improves with each iteration of Algorithm 2. After 3 iterations, the quality improvement turned out to be insignificant. The quality degrades when the execution speed is increased. However, we note that the quality of the trajectories was better than

the numbers suggest; small temporal errors (the execution lags a fraction of a second behind the reference trajectory) inflate the apparent spatial deviation, particularly at high speeds. In experiments with execution speeds up to 7x the speed of the demonstrations, the robots were able to successfully execute the knot-tie task. In the video accompanying this paper, we show footage of knot-ties performed at 1x and 5x the average speed of the demonstrations (see also `http://rll.eecs.berkeley.edu/surgical/icra10/`).

VIII. Conclusion and Future Work

In this paper, we have presented a first step towards autonomous surgical assistants. We have shown that using human demonstrations, the robots can learn to execute challenging tasks with a speed and quality beyond those of the human demonstrations. Although our current results were obtained in controlled and predictable environments, we are continuing our research with more generally set-up experiments. On the longer term, we plan to generalize our approach by incorporating visual feedback such that it can be applied to perform tasks in environments that may be different each time. We envision that we can build an extensive library of learned task "primitives", which can be executed in series to fully autonomously perform complicated and lengthy procedures.

Also, we have shown that our approach works well even with a linear approximation of the true dynamics of the robots—augmented with an adjustment for gravity. However, we believe that a more accurate dynamics model, which also considers the surgical thread and the tissue, could further improve the performance of our approach.

Acknowledgment

We thank Ron Alterovitz, Ruzena Bajcsy, Mike Branicky, M. Cenk Cavusoglu, Noah Cowan, Simon DiMaio, Vincent Duindam, Amy Fu, Blake Hannaford, Wyatt Newman, Paul Oh, Allison Okamura, Shankar Sastry, and Lee Weinstein for their inspiring ideas and valuable advice.

References

[1] P. Abbeel, M. Quigley, A. Ng. Using inaccurate models in reinforcement learning. *Proc. Int. Conf. on Machine Learning*, 2006.

[2] J. Adler, S. Chang, M. Murphy, J. Doty, P. Geis, S. Hancock. The cyberknife: a frameless robotic system for radiosurgery. *Stereotactic and Functional Neurosurgery* 69(1-4):124–8, 1997.

[3] B. Anderson, J. Moore. *Optimal Control: Linear Quadratic Methods*. Prentice-Hall, 1989.

[4] H. Asada, S. Liu. Transfer of human skills to neural net robot controllers. *Proc. IEEE Int. Conf. on Robotics and Automation*, 1991.

[5] A. Benabid, P. Cinquin, S. Lavalleè, J. Le Bas, J. Demongeot, and J. de Rougemont. Computer-driven robot for stereotactic surgery connected to CT scan and magnetic resonance imaging: technological design and preliminary results. *Applied Neurophysiology* 50:153–154, 1987.

[6] D. Bertsekas. *Dynamic programming and optimal control*. Athena Scientific, 2001.

[7] C. Birkhimer. *Extracting Human Strategies for Use in Robotic Assembly*. PhD thesis, Case Western Reserve University, 2005.

[8] J. Bobrow, S. Dubowsky, J. Gibson. Time-optimal control of robotic manipulators along specified paths. *Int. J. on Robotics Research* 4(3):3–17, 1985.

[9] D. Bristow, M. Tharayil, A. Alleyne. A survey of iterative learning control; a learning-based method for high-performance tracking control. *IEEE Control Systems Magazine*, June 2006.

[10] M. Çavuşoğlu, F. Tendick, M. Cohn, S. Sastry. A laparoscopic telesurgical workstation. *IEEE Trans. on Robotics and Automation* 15(4):728–739, 1999.

[11] A. Coates, P. Abbeel, A. Ng. Learning for control from multiple demonstrations. *Proc. Int. Conf. on Machine Learning*, 2008.

[12] A. de Luca, S. Panzieri. A simple iterative scheme for learning gravity compensation in robot arms. *Proc. Ann. Conf. ANIPLA*, 1992.

[13] A. Dempster, N. Laird, D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 1977.

[14] B. Hannaford, P. Lee. Hidden markov model analysis of force/torque information in telemanipulation. *Lecture Notes in Control and Information Sciences* 139:135–149, 1989.

[15] K. Itabashi, K. Hayakawa, T. Suzuki, S. Okuma, F. Fujiwara. Modeling of the peg-in-hole task based on impedance parameters and HMM. *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 1997.

[16] D. Jacobson, D. Mayne. *Differential Dynamic Programming*. Elsevier Publishers, 1970.

[17] M. Jordan. *An introduction to probabilistic graphical models*. In preparation, 2009.

[18] M. Kaiser, R. Dillmann. Building elementary robot skills from human demonstration. *Proc. IEEE Int. Conf. on Robotics and Automation*, 1996.

[19] Z. Kolter, P. Abbeel, A. Ng. Hierarchical apprenticeship learning with application to quadruped locomotion. *Neural Information Processing Systems* 20, 2008.

[20] Z. Kolter, A. Ng. Policy search via the signed derivative. *Proc. Robotics: Science and Systems*, 2009.

[21] T. Kowalewski, J. Rosen, L. Chang, M. Sinanan, B. Hannaford. Optimization of a vector quantization codebook for objective evaluation of surgical skill. *Studies in health technology and Informatics* 98:174–179, 2004.

[22] H. Lin, I. Shafran, D. Yuh, G. Hager. Towards automatic skill evaluation: detection and segmentation of robot-assisted surgical motions. *Computer Aided Surgery* 11(5):220-230, 2006.

[23] H. Mayer, I. Nagy, D. Burschka, A. Knoll, E. Braun, R. Lange, R. Bauernschmitt. Automation of manual tasks for minimally invasive surgery. *Int. Conf. on Autonomic and Autonomous Systems*, 2008.

[24] H. Mayer, I. Nagy, A. Knoll, E. Braun, R. Bauernschmitt. Adaptive control for human-robot skill transfer: trajectory planning based on fluid dynamics. *Proc. IEEE Int. Conf. on Robotics and Automation*, 2007.

[25] K. Moore. *Iterative learning control for deterministic systems*. Springer-Verlag, New York, 1993.

[26] D. Myers. An approach to automated programming of industrial robots. *Proc. IEEE Int. Conf. on Robotics and Automation*, 1999.

[27] S. Needleman, C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 1970.

[28] H. Rauch, F. Tung, C. Striebel. Maximum likelihood estimates of linear dynamic systems. *AIAA Journal* 3:1445–1450, 1965.

[29] J. Rosen, B. Hannaford, C. Richards, M. Sinanan. Markov modeling of minimally invasive surgery based on tool/tissue interaction and force/torque signatures for evaluating surgical skills. *IEEE Trans. on Biomedical Engineering* 48(5):579–591, 2001.

[30] H. Sakoe, S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 1978.

[31] M. Skubic, R. Volz. Acquiring robust, force based assembly skills from human demonstration. *IEEE Trans. on Robotics and Automation* 16(6):772–781, 2000.

[32] R. Taylor, B. Mittelstadt, H. Paul, W. Hanson, P. Kazanzides. An image-directed robotics system for precise orthopaedic surgery. *IEEE Trans. on Robotics and Automation* 10(3):261–275, 1994.

[33] Y. Yamakawa, A. Namiki, M. Ishikawa, M. Shimojo. One-handed knotting of a flexible rope with a high-speed multifingered hand having tactile sensors. *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2007.