2007 IEEE International Conference on
Robotics and Automation
Roma, Italy, 10-14 April 2007

Adaptive Control for Human-Robot Skilltransfer: Trajectory Planning Based on Fluid Dynamics

H. Mayer, I. Nagy and A. Knoll
Robotics and Embedded Systems
Technical University Munich
{mayerh|nagy|knoll}@in.tum.de

E.U. Braun, R. Lange and R. Bauernschmitt
Department of Cardiovascular Surgery
German Heart Center Munich
{brauneva|lange|bauernschmitt}@dhm.mhn.de

***Abstract*— A popular method for an easy and also flexible programming of robots is learning by demonstration. An intelligent controller learns a task from several examples carried out by an experienced user. Afterwards, the task can be adapted to new, formerly unknown environments. One particular challenge arising with this technique is generalization of demonstrations in order to get a generic description of the task. In this paper a new methodology for solving this problem is proposed. The main part of the algorithm exploits principles known from fluid dynamics.**

***Index Terms*— learning by demonstration, imitation learning, one-shot learning, fluid dynamics**

I. Introduction

Learning by demonstration (aka. imitation learning) has proven to be an useful technique for instructing robots by end-users. I.e. a user which is an expert in a domain different from robotics (e.g. assembly, welding or even surgery) can easily upgrade the controller of the robot with new capabilities without caring about details of robot programming. Most implementations of this technique require a sufficient number of demonstrations in order to generalize the task. Generalization normally comprises finding a description of the task detached from any coordinate system. In addition, it is usually desirable to adjust the shape of the trajectories themselves in order to adjust the task to a certain environment. We have addressed the typical challenges arising from this technique with a new implementation of learning by demonstration which employs algorithms from fluid dynamics. The basic idea can be described as follows: imagine a person stirring a fluid, e.g. a cup of tea. Two important observations can be made. The first is, that not only particles lying on the path of stirring are affected, but all particles in the fluid are drawn into a certain direction biased by the stirring. The other observation is, that the fluid, the tea, continues with its motion after stirring has stopped. I.e. while the former phenomenon implements some sort of generalization, the latter realizes a memory. How can these observations be exploited in order to implement the basic principles of learning by demonstration? The answer is obvious: we have to model the environment, where certain manipulation tasks are demonstrated, as a fluid. An important prerequisite is, that all demonstrations of the same task are congruent and can be overlaid (a solution to this problem will be discussed below). Given this assumption, user demonstrations can be seen as the stirring of a fluid. The behavior of the fluid will on the one hand bring about a generalization of the overall movement and on the other hand smooth motions and eliminate outliers. Since fluid dynamics is a field of extensive research with an abundance of different methodologies, we will discuss the selection of an appropriate model in an own section of this paper. For now, it should only be emphasized that a variety of dynamic, non-stationary models is available, i.e. trajectories can cross themselves at different points in time in order to enable demonstrations of complex tasks. A crucial advantage of this approach is, that we will get a working solution (fluid in motion) even after showing only a single demonstration of the task ("one-shot learning"). Nevertheless, it is still advising to have additional demonstrations in order to make the solution more general and to eliminate outliers. If we want to instantiate the learned task in a new environment, we simply transform the previously acquired model to the desired position. The fluid will adapt itself to this new situation: e.g. the trajectory (the flow of the fluid) will bypass obstacles and smooth out discontinuities. Another advantage is, that the starting points within the flow (i.e. the initial positions of the robot's end effectors) are not too important. From each point within a certain neighborhood, particles representing the end effectors will be automatically drawn into the right direction by the simulated current of the fluid.

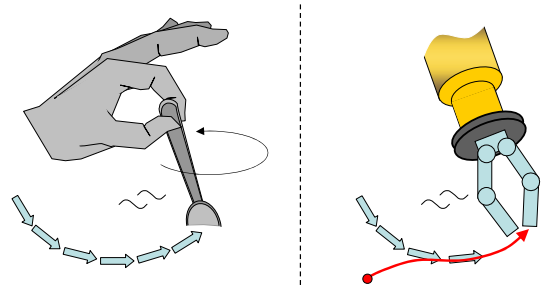

Fig. 1. **Illustration of the principle:** Demonstrations of the task are simulated as stirring a fluid, which consequently smoothes and "memorizes" the movement (note that we employ a non-stationary, dynamic model which allows self-crossing trajectories). Instantiations of the task are simulated as throwing a particle into the flow. This particle, representing the center of the robot's end effector, will be drawn into the right direction by the flow.

II. Materials and Methods

In this section we introduce the hardware which has been used for the experiments and whose operation has raised

the demand for programming by demonstration. Afterwards we will describe the task in detail and propose its decomposition into several subtasks. Finally, these subtasks will be addressed by appropriate algorithms. The crucial part will be the adaption and application of a suitable model for simulation of fluid dynamics.

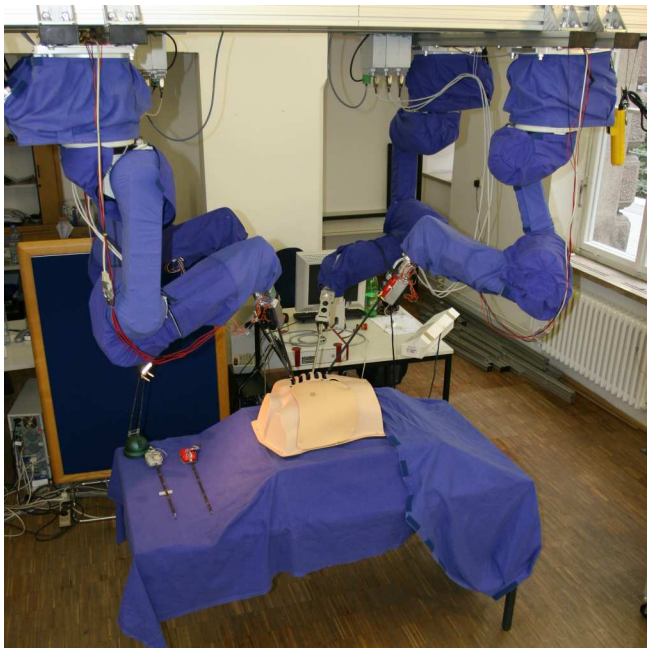

Fig. 2. **Hardware Setup:** Four ceiling mounted robots can be equipped with either a stereoscopic endoscope or different surgical instruments. Instruments are augmented with force sensors. In-output is accomplished at a master console, comprising two 6Dof force feedback devices

A. Hardware Setup

For all of our experiments we have used a robotic system for endoscopic heart surgery. Hardware and software of the system itself have already been introduced to the research community [1], [2]. Therefore, we constrict the following description to an extent necessary for understanding the subsequent sections. As one can see in figure 2, the system comprises four ceiling mounted robots. This setup was chosen after an extensive evaluation with surgeons from the German Heart Center Munich [3]. The ceiling mounted arrangement is advantageous in comparison with other setups, because it does not block access to the operating table, although we even use four robots. The robots are mounted on a gantry assembled of profiled girders which enable a quick reconfiguration of the setup. Several emergency stops guarantee secure handling of the system. Each robot can be equipped with either an endoscopic stereo camera or different surgical instruments. All instruments are augmented with strain gauge sensors in order to measure occurring forces. User interaction is realized with a master console which can be freely placed in the room. It is equipped with two PHANToM^{TM} force feedback devices which on the one hand can be used for 6DoF input and on the other hand provide a 3DoF force feedback derived from the measurements at the instruments. It is possible to switch between control over different robotic arms. Therefore, each of the four robots can be controlled with this bimanual input device (two of them at the same time). The control software of the system implements so-called trocar kinematics, i.e. all instruments will move about a fixed fulcrum after insertion into the body (or a ribcage mockup).

B. Description of the Task

With the system described above we have already performed surgical tasks like cutting and sewing. Due to a more complex handling of minimally invasive systems, an intrinsic issue of these procedures is, that their completion takes significantly more time and suffers from an increased rate of errors. E.g. tying a knot takes up to one minute with a minimally invasive system, while this task is often completed within seconds in open surgery. In addition, one can more often observe breaking of suture material or even ruptures of tissue. Therefore, the intention of this work is to provide a possibility to automate complex tasks like endoscopic knot-tying. The goal is a generic knot that can be applied by the surgeon at a desired position on the tissue. In addition, it should be possible for the surgeon (normally not an expert in programming robots) to augment the system with other tasks (e.g. a different type of surgical knot). The first step towards this goal is to analyze how the knot-tying task is performed by a human. Based on this, surgical knot-tying can be decomposed into the following subtasks (see description of fig. 3).

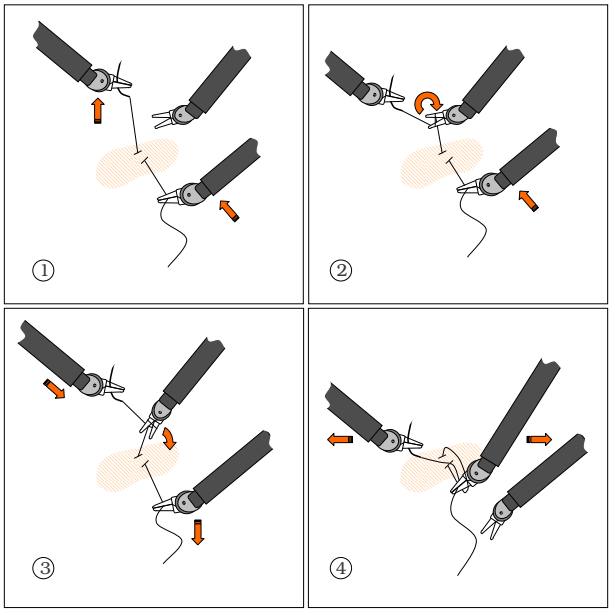

Fig. 3. **Knot-Tying Procedure:** After piercing, the needle with the thread is pulled out of the tissue with the left gripper (1) and winded about the right gripper (2). During this procedure, the loose end of the thread is retained by an assistant. This facilitates grasping of the end by the right gripper (3). Finally, the end of the thread is pulled through the loop around the right gripper in order to finalize the knot (4).

So far, there has been only little research on this particular task of automating a surgical knot. Kang has proposed a

generalized version of a taught knot in his PhD thesis [4]. In his project the task is performed with special knot-tying instruments, while we are using multi-purpose instruments. Hynes et al. [5] and Wakamatsu et al. [6] proposed robotic setups for knot-tying, but did not evaluate the task under realistic circumstances, yet (i.e. small-scale knot performed under the restrictions of trocar kinematics and with original suture material). There exists also some work on analyzing the knot-tying task itself, focusing rather on surgical evaluation instead of automation [7], [8]. We have already implemented knot-tying with our system to alleviate analysis of the task. The system described above is equipped with three grippers and therefore can assume the work of both, master and assistant. It was able to tie a surgical knot in about 30 seconds, which is already an improvement compared to human performance in endoscopic knot-tying. Up to this point no learning was included, i.e. the knot cannot be adapted to unknown environments which is the topic of the following sections.

C. Algorithmic Solution

The procedure depicted in fig. 3 describes a human-centered decomposition of the task. It works fine for two cooperating surgeons (master and assistant), but this human-centered approach is not necessarily the best decomposition of the task in order to transfer it to a technical system. Therefore, we did not start our research with the observation of two humans performing this task, like it is done in manual surgery. Instead, we recorded the movements of a single human working with our experimental system and performing the knot with only two hands (i.e. without an assistant like depicted in fig. 3). We started recording the trajectories after an initial situation as depicted in fig. 3 (1) was reached, i.e. the needle is already pierced through the tissue. As a result of these recordings we have acquired two trajectories (left / right hand) as data source for our research. In the subsequent sections we will use the expressions task, demonstration, skill and primitive in a specific way. A task is a certain, well-defined description how to solve a problem. A demonstration is an instance of this task performed by a human user. A skill is the instantiation of a task on a robotic system. Finally, primitives are meaningful, non-overlapping subsequences of a skill. We now will apply the following steps in order to derive a robotic skill from user demonstrations:

1) **Smoothing of the trajectory** and transformation into a generic, instantiable representation.
2) **Decomposition** of the complete trajectory into meaningful primitives.
3) **Grouping and matching** of primitives from different demonstrations.
4) **Reassembly of skills** from primitives and instantiation as viscous flow

1) Smoothing and Transformation: As one can see in fig. 4, the data acquired from user demonstrations is quite noisy and exhibits some tremor of the subject. Therefore, the first step will be smoothing input data, but this is often associated with loss of information. I.e. before we can smooth the data we have to be sure which parts of it are necessary for further processing and which parts can be neglected without consequences. One idea would have been to use a Gaussian filter in order to remove noise. But this would also remove some important edges, which are essential to perform the task. E.g. the leftmost edge of the trajectory in fig. 4 constitutes a correct pick up of the thread. If this edge is smoothed out even just for millimeters, the thread will be missed. Therefore, we need a kind of intelligent smoothing. We have implemented this feature by means of spline approximation, where important parts of the trajectory (like the pickup edge) are chosen as breakpoints. These points are found by including additional modalities into our considerations. We have exploited temporal dependencies within the trajectory and the state of the gripper. The former is used to identify points where the user moves rather slowly, which indicates increased precision. The latter implies important information where the user was interacting with the environment by grabbing objects (e.g. the thread). At the left side of fig. 5, all locations where the user stayed within a radius of 1.5 mm for more than 0.2 $secs$ are marked with a cone. Therefore, the trajectory is supersampled within a raster of 3 mm and the cone has the same orientation as the gripper which has produced the trajectory. If the gripper is closed, the color of the trajectory changes from green to red. Important points on the trajectory are identified reliably (e.g. the location where the thread is picked up, at the upper left side of the trajectory, is marked by two cones and in addition, the gripper is closed nearby). An important observation is, that there are no cones on the way to the pick up position and back. This indicates that the corresponding part of the trajectory is not too important and therefore a distinct smoothing can be applied. This methodology is based on profound research on the psychology of user interfaces. If

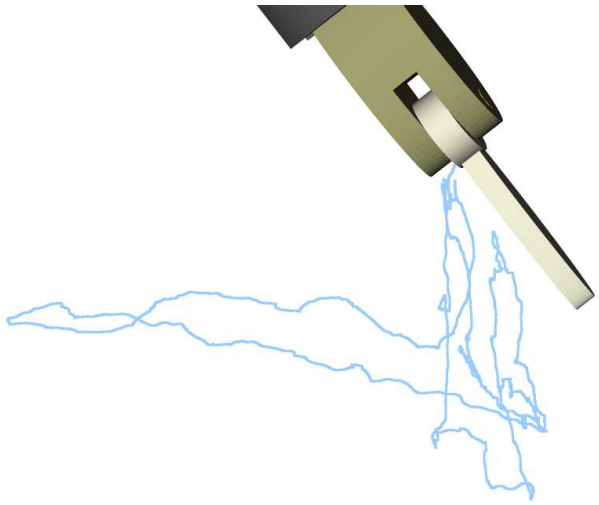

Fig. 4. **Knot-Tying Trajectory:** Visualization of the trajectory of the left gripper during knot-tying. For clarity reasons, the right gripper and its trajectory are hidden. The image was produced with the 3D planning and recording software of our system.

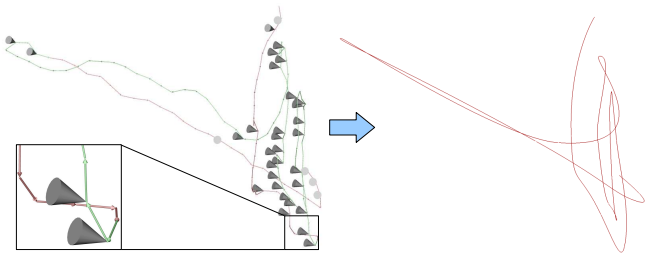

Fig. 5. **Spline approximation:** Significant points are selected from the recorded trajectory (cones on left image). These points serve as breakpoints for a spline approximation of the trajectory (right image). Note that human tremor is annihilated while all significant edges are preserved

a user moves the mouse pointer across the screen very fast, it is not possible to perform precise interactions with the user interface, since the focus window of the eye cannot achieve full focus and concentrate on the task [9]. Furthermore, this observation can be prooven by Accot-Zhai's steering law [10], an enhancement of the well-known Fitts' law [11] for movements on constricted trajectories. In its original form it is expressed as

$$T_C = a + b \int_C \frac{ds}{W(s)} \tag{1}$$

where a and b are constants depending on the nature of the experiment, $W(s)$ is the width of the curved path C, at a certain location s. For clarification, this formula can be differentiated and rearranged to

$$\frac{ds}{dT} = \frac{W(s)}{b} \tag{2}$$

As one can derive from equation 2, the speed of the user moving on the trajectory is proportional to the width of the path. Or in other words, if the user has to stay on a narrowly constricted trajectory (i.e. performing a precise movement) it will take more time than any unconfined movement. This is exactly the phenomenon which was exploited by our algorithm in order to define breakpoints without loss of critical features.

2) Decomposition: After smoothing the trajectory and transforming it into spline representation, the next step is to decompose it into meaningful primitives. The decomposition of manipulation tasks into sensorimotor actions or motion primitives is an approved methodology to reduce complexity [12], [13]. Nevertheless, motivations for this procedure might differ: beside reduction of complexity, they can also be used as a basis for new tasks, by reutilizing different primitives and adjusting their parameters. In addition, a library of primitives can be used to analyze previously unknown user demonstrations. We will concentrate on the latter with this step of the algorithm. In our case the selection of meaningful primitives is complicated, because demonstrations are not normalized. I.e. different demonstrations can be translated or rotated and primitives can appear with different parametrizations. Therefore, we cannot employ procedures proposed for normalized data (e.g. [14], [15]). Another design criterion for our primitive generation is, that we would like to decompose any given trajectory into a sequence of two-dimensional sections which can be embedded into computational fluid dynamics. Using only two dimensions will accelerate this process significantly. It is easy to proof that two-dimensional decomposition is always possible, since every trajectory can be sampled into single lines, which actually have just one dimension. But then, we do not want the primitives to become too simple, i.e. they should at least represent recurring actions with distinct pre and post conditions. In order to reach this, we can look at human manipulation tasks, again. An important observation is, that even the complexest behaviors are actually realized in only two dimensions, regarding the movements of distal extremities (fingers, hands, feet): e.g. playing piano or guitar only takes place on a two-dimensional key- / fingerboard; walking or even stair-climbing can be normalized to two dimensions. These observations are also supported by recent research on learning sensorimotor behaviors [16]. Therefore, we can represent any 3D trajectory A in the following way:

$$A = X_0 \cup T_1 P_1 \cup X_1 \cup T_2 P_2 \cup \ldots \cup T_n P_n \cup X_n = \bigcup_{i=1}^{n} T_i P_i \tag{3}$$

$$P_i \cap P_j = \emptyset \ \forall i, j \in n; i \neq j \tag{4}$$

where P_i is a two-dimensional primitive which is placed at its correct position by a rigid transform T_i. Between the primitives, chunk sections X_j might occur which have no relevance for the task and can be replaced by a direct connection of consecutive primitives. As a starting point for our implementation of task decomposition we can use the features of the previous section: the breakpoints of the spline representation. These points are also important hints for selecting significant primitives. They will be seed points for the actual algorithm, which is based on 3D plane fitting. We have implemented a divide-and-conquer algorithm working on the temporally ordered point set of the trajectory which is going to be decomposed. The central part of this approach is a function which calculates an optimal plane for a given set of 3D points (optimality regarding the sum of perpendicular distances from the plane to all points in the set). This is done by means of PCA, where the first two principal components constitute the desired plane. Thereupon, the algorithm selects the largest subset of points whose planar error is below a certain threshold. Those point sets are collected in a list of primitives which serves as input for the next step. The algorithm is called recursively for the remaining part of the trajectory.

3) Matching and Grouping: The whole learning procedure described in this paper will produce an applicable solution even after one single demonstration ("one-shot learning"). This is a significant improvement over learning based on neural networks. After one demonstration, neural networks can only provide an identical replay of the demonstration, while this approach has an intrinsic adaptive behavior, based on dynamical systems from fluid dynamics (see below). Anyhow, if there are subsequent demonstrations

of the same primitive, we want our system to consider them in order to improve generalization. Therefore, we need a methodology to identify similar primitives and merge their peculiarities. This step is alleviated by the fact, that primitives already have a 2D presentation. Nonetheless, different instantiations of the same primitive may differ in rotation, translation and scaling. Therefore, each newly acquired primitive is matched against a list of already existing primitives, in order to check if it is just another instantiation. This matching is done by singular value decomposition of the corresponding covariance matrix:

$$C = \frac{1}{n}\sum_{j=1}^{n}\left[\ \overrightarrow{P_{ij}} - \overline{P_i}\ \right]\left[\ \overrightarrow{Q_{kj}} - \overline{Q_k}\ \right]^T \xrightarrow{\textit{svd}} C = USV^T \tag{5}$$

$$\textit{where } \overline{P_i} = \frac{1}{n}\sum_{j=1}^{n}\overrightarrow{P_{ij}};\ \overline{Q_k} = \frac{1}{n}\sum_{j=1}^{n}\overrightarrow{Q_{kj}} \tag{6}$$

$\overrightarrow{P_{ij}}$ is the j-th point of the newly found primitive P_i, while Q_k is the k-th primitive in a list of m primitives which have already been identified ($1 \leq k \leq m$). Note that both primitives are stored as splines and therefore both can be resampled with an equal number of n points. The diagonal of S contains the eigenvalues s_1, s_2 of matrix C^TC. U and V contain the eigenvectors of CC^T and C^TC, respectively. By means of these results, we can determine the scaling factor f, the 2D rotation matrix R and the translation vector $\overrightarrow{t}$ which can be used to map P_i onto Q_k:

$$f = \frac{s_1 + s_2}{\sigma_P} \quad \textit{where} \quad \sigma_P = \frac{1}{n}\sum_{j=1}^{n}\left(\overrightarrow{P_{ij}} - \overline{P_i}\right) \tag{7}$$

$$R = f \cdot UV \tag{8}$$

$$\overrightarrow{t} = \overline{Q} - f \cdot R\overline{P} \tag{9}$$

Now, primitive P_i can be mapped onto primitive Q_k by $P^*_{ij} = fR \cdot P_{ij} + t \ \ \forall j \in 1 \ldots n$. This procedure is repeated for all primitives Q_k which are already in the list ($k \in 1 \ldots m$) and for each k, the distance of P^*_i from Q_k is determined (sum of squared errors of the congruent points). If the error exceeds a given threshold, P^*_i is appended to the list as new primitive Q_{m+1}. Otherwise P^*_i is merged with a primitive Q_{opt} which yields the shortest distance. Merging is done by linear interpolation between congruent points of P^*_i and Q_{opt}. Since we need a spline representation of each primitive, it is sufficient to take only breakpoints into account. A breakpoint of P^*_i is only included, if a majority of demonstrations of this primitive shows this breakpoint, too. In order to decide this issue, the original trajectory of each primitive is stored in a list associated with this primitive group. Since these trajectories have been recorded at discrete time steps, we can use them later to determine the speed at certain positions of the primitive.

4) Generalization and Instantiation: So far, we have spline representations of the primitives. This would be sufficient in order to reassemble a certain skill from equation 3, if all T_i are known. Otherwise, an application of splines has some significant shortcomings. The most important is, that any shift of the breakpoints, as it might become necessary due to adaptation to a new environment, can lead to unfavorable trajectories [17]. In addition, it is difficult to store temporal features like speed with splines, since the dependency between interpolation parameter and arc length is nonlinear. Therefore, we propose the usage of dynamical systems known from fluid dynamics in order to instantiate primitives. So far there has been only little research on dynamical systems for storage and generation of motion primitives. Ijspeert et. al [18] have employed dynamical systems as generators for motion patterns in order to mimic locomotion of animals. A related approach was proposed by Okada et. al. [19]. They have used attractors of dynamical systems to generate and stabilize walking movements of a humanoid robot. Both approaches operate on the joint-level of motion generation, whereas the proposed method generates trajectories in Cartesian space. Recently, there has also been work on analyzing trajectories by means of dynamical systems. Dixon et al. [20] have proposed a method for segmenting primitives based on linear dynamical systems. Although this can be used to segment and store motion patterns, the expressiveness of the derived primitives is limited and therefore they cannot be used for generalization (which was not the intention of their work). Each of the three mentioned projects uses time-invariant dynamical systems, which lead to uncomplex and stable solutions for movements in joint space. Contrarily, our goal is to provide trajectory generation in Cartesian space for rather complex motions (e.g. self-crossing trajectories). Therefore, we need a time-dependent system which can reproduce complex trajectories. Such systems are applied in fluid dynamics, where they are used to simulate physical circumstances, e.g. in a wind tunnel. Streaklines occurring in this environments are nothing else than trajectories of particles in a fluid. So far, there has been no attempt to utilize this form of trajectory generation for robotic applications. For our approach we have chosen a dynamical system based on Navier Stokes Equations. These equations describe the behavior of a viscous, incompressible fluid exposed to friction and external forces. The derivation of the equations can be found in various books on fluid dynamics (e.g. [21]). For our experiments we have chosen a simplified form of the equations with constant density, since we restrict our approach to incompressible flows:

$$\frac{\partial \overrightarrow{u}}{\partial t} + (\overrightarrow{u} \cdot \nabla)\overrightarrow{u} + \nabla p = \nu \Delta \overrightarrow{u} + \overrightarrow{f} \tag{10}$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0 \tag{11}$$

∇ is the Nabla operator and Δ is the Laplace operator: $\Delta = \nabla \cdot \nabla = \nabla^2$; $\overrightarrow{u}$ is the velocity of the fluid, ν its viscosity and $\overrightarrow{f}$ are external forces like gravity. As mentioned above, the primitives we want to represent with this dynamical system are already in 2D space. Therefore, equations 10 and 11 can be concretized to (cf. [22]):

$$\frac{\partial u}{\partial t} + \frac{\partial p}{\partial x} = \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) - \frac{\partial u^2}{\partial x} - \frac{\partial (uv)}{\partial y} + f_x \tag{12}$$

$$\frac{\partial v}{\partial t} + \frac{\partial p}{\partial y} = \nu \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) - \frac{\partial (uv)}{\partial x} - \frac{\partial v^2}{\partial y} + f_y \quad (13)$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \quad (14)$$

where u and v are velocities in x and y direction, respectively. We evaluate the equations by means of finite differences within a rectangular area which is subdivided into a grid of equally sized cells. Within these cells the partial derivatives can be replaced by local difference quotients - e.g. $\left[\frac{\partial u}{\partial x}\right]_{ij} \longmapsto \frac{u_{ij} - u_{i-1j}}{d}$, where d is the length of each cell. A detailed description of this methodology can be found in [22]. In addition to discretization, we have to fix the velocities at the boudaries of the simulated area (e.g. u_{00} and v_{00} in fig. 6). We will set them to zero, since we want the fluid to adhere to boundaries or objects within the stream. Therefore, velocities of particles near a boundary will be relatively small, but due to the nature of this equations, no particle will ever reach or even penetrate a boundary. I.e. a kind of collision avoidance is included in our system from scratch (of course, this only affects the position of the end-effector which will be generated from particle simulation - other parts of the robot can collide, yet). By means of

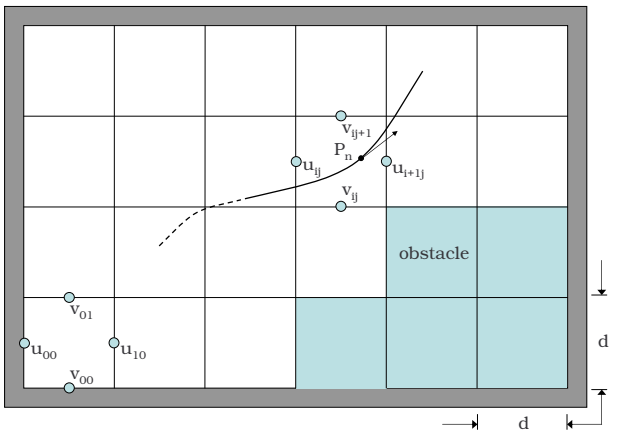

Fig. 6. **Fluid simulation:** The area of interest is discretized into equally sized cells. For clarity reasons this picture shows only a limited number of cells which would be too coarse for practical use - the actual implementation usually works on a grid of at least 50×50 cells. Evaluation of velocities is not centered within a cell, but distributed on a staggered grid in order to assure numeric stability.

these boundary conditions we can define obstacles within the area of simulation in order to adapt the trajectory to new environments. Now we can reproduce a demonstrated skill by reassembling the corresponding primitives with equation 3. Therefore, each primitive will be instantiated by a 2D fluid simulation. This can be achieved by sampling points from the spline representation of the corresponding primitive. In this case, equidistant sampling with length d is applied. There is no direct way to guarantee an arc length of d for spline $S(x)$, since Δx cannot be determined from $S(x+\Delta x) = S(x)+d$. So far, we have solved this problem by a binary search: Let Δx be an arbitrary initial value and $S(x)$ maps to a coordinate within the cell with velocities u_{ij} and v_{ij}. Then we test for $S(x + \Delta x)$ lying within any adjacent cell. If it still lies in cell ij we try $S(x + 2\Delta x)$, if it lies even outside an adjacent cell we try $S(x + 0.5\Delta x)$ and so on (note that Δx refers to a distance and has nothing to do with the Laplace operator in equation 10). Once we have sampled an applicable point S_x from the trajectory, we can determine the speed at this point with the help of the original trajectories which have been stored together with this primitive. Afterwards, we interpolate the neighboring values of u_{ij}, v_{ij}, u_{i+1j} and v_{ij+1} (see fig. 6). I.e. we calculate a preset for these velocities at timestep t_n. All other velocities within the grid are derived from fluid simulation. We now can throw a particle into the stream and it will be attracted by the trajectory of the underlying primitive. Since we know the position and orientation of the simulation grid from equation 3, we can generate suitable 3D points for a skill. Since the simulation is only refreshed at discrete points in time ($t_0 < t_n < t_{max}$), we have to interpolate again to get positions at arbitrary points in time. Fortunately, this works well even for tiny time steps. Therefore, we can sample positions at the frequency of the controller of the robot (approx. 150 Hz). I.e. we can directly control our robots with positions from fluid simulation.

III. Experimental Results

For generating demonstration examples we have used two grippers, but no assistant arm (as opposed to the procedure depicted in fig. 3). This was not necessary, because we have used an artificial heart, where the thread will not be occluded by blood. Therefore, it can be grasped again without the help of an assistant arm that guarantees retainment of the thread's position. In order to get utilizable trajectories, their starting points (positions of the grippers) have to lie within a limited working space ($10 \times 10 \times 10\ cm$). This is not due to a weakness of the methodology described above, but the absolute calibration of the robots is limited. Anyway, the size of this working space is sufficient for endoscopic heart surgery. For our experiments we have recorded several performances of an endoscopic knot (see above). The trajectories have been sampled at the frequency of our robots (approx. 150 Hz). In order to avoid accumulation of points during stops of the user, we only have recorded points which are at least $1mm$ away from their predecessor (which is approximately the accuracy of our system). To get first results and a proof of concept, we have processed the data offline.

First of all, we have applied the spline based smoothing as it was described in section II-C.1. This part of the algorithm has caused no problems and we were able to derive a spline representations of the data (depicted above in fig. 5). The next step was the decomposition of the trajectory into meaningful primitives (see section II-C.2). The trajectory was sampled into smaller parts, but some of them have been too small to be regarded as meaningful primitives. In addition the problem occurred, that some points have been missed, where an abscission of a primitive would have been expedient. All in all, about 30 percent of the primitives have needed manual interference, i.e. we have moved or removed some cutting points within the trajectory. After repeated application of the

above steps to all demonstrations, we have acquired groups of similar primitives derived from different demonstrations. The normalization of the primitives with the SVD-based method described in section II-C.3 has raised no further problems. All primitives of the same group have been merged into one presentation this way. Anyway, it should not be concealed that this positive outcome is partially due to the manual decomposition of primitives which certainly relieves this step.

At this point, we had a single representation for each group of primitives. As mentioned in section II-C.3, this representation comprises the 2D spline and a list of all original trajectories for each primitive pruned from certain demonstrations. We now define a raster (as depicted in fig. 6) sized to fit the 2D spline representation of the primitive group. This raster is divided into 50×50 cells. Afterwards the fluid simulation is started by calculating the starting point of the spline representation. Given this coordinate, we seek the closest points of the trajectories in the list mentioned above. Since we know the sampling rate of the trajectories, we can determine the mean velocity at this points as follows:

$$\overrightarrow{v}_{t,i} = \frac{\overrightarrow{p}_{t+1,i} - \overrightarrow{p}_{t,i}}{dt} \Rightarrow \overrightarrow{v}_t = \frac{1}{n}\sum_{i=1}^{n} \overrightarrow{v}_{t,i} \qquad (15)$$

where i is the number of the demonstration from which the trajectory was taken; $\overrightarrow{p}_{t,i}$ is the corresponding point at time t and dt is the time step for sampling the trajectory. While all other velocities in the raster are set to zero, we initialize the values of u and v (see fig. 6) of the cell closest to $\overrightarrow{p}_t$ with the help of $\overrightarrow{v}_t$:

$$u_{ij} = \left(\frac{\overrightarrow{p}_t e_1^T}{d} - i\right)\overrightarrow{v}_t e_1^T,\ u_{i+1j} = \left(i + 1 - \frac{\overrightarrow{p}_t e_1^T}{d}\right)\overrightarrow{v}_t e_1^T \qquad (16)$$

note that i and j are coordinates of the raster this time. The values for v_{ij} and v_{ij+1} can be acquired accordingly. Given this initial situation, we now can run the fluid simulation until it reaches a stable distribution of u and v, i.e. the change in velocities falls below a certain threshold. We keep this distribution as initialization for the next time step $t + 1$ and preset the next velocity v_{t+1} as described above. Due to the initialization with the results from t, the simulation converges much quicker for all subsequent time steps. Figure 7 shows the simulation at time step 100 where the velocity was set at the position marked with a circle. In order to instantiate the primitive we need a fixture point in the environment where it should be placed. For the primitive depicted in fig. 7 we have taken the point marked with a quad, which is the position of the gripper to be winded about. Once the primitive is fixed and instantiated, it can adapt itself to different initial situations by means of the simulated flow (marked with + in fig. 7). Therefore, the primitive can be reused in different tasks with different geometrical restrictions. Another important feature for learning tasks is the adaption to new environments. In fig. 8 one can see the adaption of the same primitive to an environment with an obstacle. After deforming the trajectory spline, the obstacle is inserted into the simulated environment and the fluid simulation is continued until it reaches a stable distribution again.

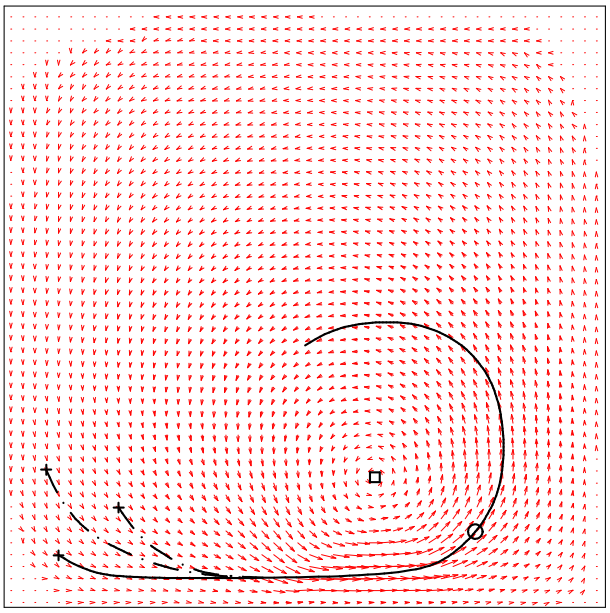

Fig. 7. **Flow visualization:** Fluid simulation of the winding primitive. The vector field is a snapshot at the time step, where the particle (i.e. the gripper) is at the position marked with a circle. Note that the depicted trajectories are overlays just for illustration and the vector field looks different for all other time steps. Therefore, parts of the depicted trajectories might be contradictory to the vector field which is only valid for this particular time step.

IV. Conclusion and Discussion

We have presented a novel approach for learning by demonstration based on fluid simulation. This method can be used to derive a very general form of motion primitives. We have presented techniques for adapting these primitives to different initial situations and to new environments with previously unknown geometric features (e.g. obstacles). The concept has been proven by experiments, although there are still some issues to be resolved in future versions. One particular problem is the segmentation of primitives. Since approx. 30% of the breakpoints are set manually, this step cannot be called autonomous, yet. We are currently working on an improvement by means of inclusion of other modalities like force measurement. Another issue is computation time, in particular the time needed for generalization (e.g. approx. $16.5\ min$ for the winding primitive). This is caused by the fact that we have to run the fluid simulation for each time step (i.e. each sampling point on the trajectory). After inserting an obstacle we have to run the simulation again for each point and it takes significantly more cycles to reach again a stable distribution of velocities. One possible solution to this issue, which we are currently working on, is parallelization and computation on the GPU of a graphics card. Regarding fluid simulation, we are planning to evaluate a 3D version.

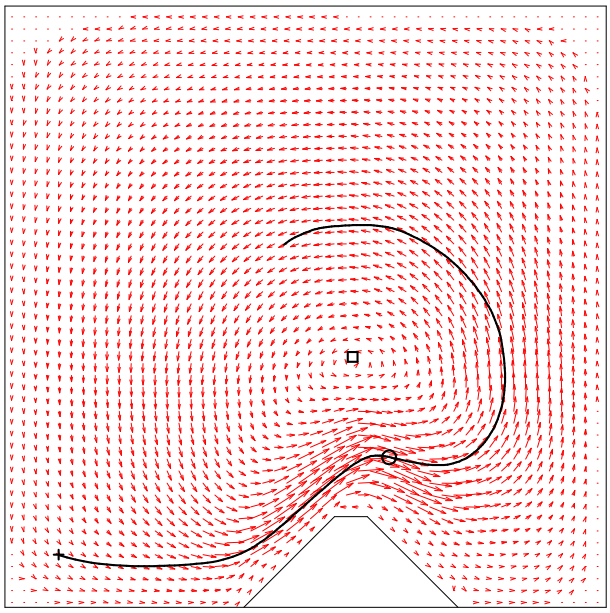

Fig. 8. **Obstacles:** The same primitive as depicted in fig. 7 adapted to an obstacle. At each time step the obstacle is inserted into the raster and simulation is continued under the new circumstances. Additionally, this leads to a shift of the fixture point (marked with a quad) and therefore will also have influence on other primitives (e.g. the movement of the second gripper). Note that the shape of the obstacle is improved for illustration - the real shape used for simulation is discretized with quads at cell level.

V. ACKNOWLEDGMENTS

This work is supported by the German Research Foundation (DFG) within the Collaborative Research Center SFB 453-I4 on "High-Fidelity Telepresence and Teleaction" and by the German Heart Center Munich, Department of Cardiovascular Surgery.

REFERENCES

[1] H. Mayer, I. Nagy, A. Knoll, E.U. Schirmbeck and R. Bauernschmitt. Robotic system to evaluate force feedback in minimally invasive surgery. *In: Proceedings of DETC, ASME Design Engineering Technical Conferences*; Salt Lake City, USA; 2004, nr. 57046 [CD ROM].

[2] I. Nagy, H. Mayer, A. Knoll, E.U. Schirmbeck and R. Bauernschmitt. The Endo[PA]R system for minimally invasive robotic surgery. *In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*; Sendai, Japan; 2004, pp. 3637-3642.

[3] E.U. Braun, C. Haßelbeck, H. Mayer, A. Knoll, S.M. Wildhirt, R. Lange and R. Bauernschmitt. Assessment of surgical experience for telemanipulated heart surgery. *The World Congress on Medical Physics and Biomedical Engineering*; Seoul, Korea; 2006, IFMBE Proceedings, vol. 14 [CD ROM], IOMP Proceedings, vol. 2 [CD ROM].

[4] H. Kang. Robotic assisted suturing in minimally invasive surgery, *Ph.D. thesis*, Rensselaer Polytechnic Institute, Troy, New York, 2002.

[5] P. Hynes, G. Dodds and A. Wilkinson. Uncalibrated visual-servoing of a dual-arm robot for MIS suturing. *In: Proceedings of the IEEE International Conference on Biomedical Robotics and Biomechatronics*; Pisa, Italy; 2006, pp. 204-209.

[6] H. Wakamatsu, A. Tsumaya, E. Arai and S. Hirai. Manipulation planning for knotting/unknotting and tightly tying of deformable linear objects. *In: Proceedings of the IEEE International Conference on Robotics and Automation*; Barcelona, Spain; 2005, pp. 2516-2521.

[7] M. Kitagawa, A. Okamura, B. Bethea, V. Gott and W. Baumgartner. Analysis of suture manipulation forces for teleoperation with force feedback. *In Proceedings of the Fifth International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI), T. Dohi and R. Kikinis (Eds.), Lecture Notes in Computer Science*; Springer-Verlag, vol. 2488, pp. 155-162, 2002.

[8] C. Cao, C. MacKenzie and S. Payandeh. Task and motion analyses in endoscopic surgery, *In: Proceedings ASME Dynamic Systems and Control Division*; Atlanta, USA; 1996, pp. 583-590.

[9] A. Blackwell, A. Jansen and K. Marriott. Restricted focus viewer: a tool for tracking visual attention, *In M. Anderson, P. Cheng and V. Haarslev (Eds.), Theory and Application of Diagrams. Lecture Notes in Artificial Intelligence (LNAI)*; Springer-Verlag, vol. 1889, pp. 162-177, 2000.

[10] J. Accot and S. Zhai. Beyond Fitts' law: models for trajectory-based HCI tasks, *In Proceedings of the ACM CHI Conference on Human Factors in Computing Systems*; Atlanta, Georgia, USA; 1997, pp. 295-302.

[11] P. Fitts. (1954). The information capacity of the human motor system in controlling the amplitude of movement, *Journal of Experimental Psychology*; vol. 47, pp. 381-391, 1954.

[12] M. Kaiser and R. Dillmann. Building elementary skills from human demonstration, *In: Proceedings of the IEEE International Conference on Robotics and Automation*; Minneapolis, Minnesota, USA; 1996, pp. 2700-2705.

[13] S. Schaal, J. Peters, J. Nakanishi and A. Ijspeert. Learning movement primitives, *Springer tracts in Advanced Robotics: International Symposium on Robotics Research (ISRR)*; Siena, Italy; 2004 rec-nr. 1805.

[14] L. Reng, T. Moeslund and E. Granum. Finding motion primitives in human body gestures. *In S. Gibet, N. Courty and J. Kamp (Eds.), 6th International Gesture Workshop. Lecture Notes in Artificial Intelligence (LNAI)*; Springer-Verlag, vol. 3881, pp. 133-144, 2006.

[15] T. Sielhorst, T. Blum and N. Navab. Synchronizing 3D movements for quantitative comparison and simultaneous visualization of actions, *In Proceedings of the Fourth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*; Vienna, Austria; 2005, pp. 38-47.

[16] T. Aflalo and M. Graziano. Possible origins of the complex topographic organization of motor cortex: reduction of a multidimensional space onto a two-dimensional array, *Journal of Neuroscience*; Stanford University's HighWire Press, vol. 26, no. 23, pp. 6288-6297, 2006.

[17] S. Schaal, A. Ijspeert and A. Billard. Computational approaches to motor learning by imitation. *Philosophical transactions of the Royal Society of London, series B* vol. 358, no. 1431, pp. 537-547, 2003.

[18] A. Ijspeert, A. Crespi and J. Cabelguen. Simulation and robotic studies of salamander locomotion. Applying neurobiological principles to the control of locomotion in robots, *Neuroinformatics*; vol.3, no. 3, pp. 171-196, 2005.

[19] M. Okada, K. Osato and Y. Nakamura. Motion emergency of humanoid robots by an attractor design of a nonlinear dynamics. *In: Proceedings of the IEEE International Conference on Robotics and Automation*; Barcelona, Spain; 2005, pp. 18-23.

[20] K. Dixon and P. Khosla. Trajectory representation using sequenced linear dynamical systems. *In: Proceedings of the IEEE International Conference on Robotics and Automation*; Barcelona, Spain; 2005, pp. 3925-3930.

[21] T. Chung. Computational fluid dynamics. Cambridge University Press, New York, USA, 2002

[22] T. Griebel, T. Dornseifer and T. Neunhoeffer. Numerical simulation in fluid dynamics. A practical introduction. *SIAM Monographs on Mathematical Modeling and Computation no. 3*; Society for Industrial and Applied Mathematics, Philadelphia, USA, 1997