Automation of Manual Tasks for Minimally Invasive Surgery

H. Mayer, I. Nagy, D. Burschka and A. Knoll
Robotics and Embedded Systems
Technical University Munich
{mayerh|nagy|burschka|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*— We have developed an experimental system for minimally invasive surgery providing force feedback and automation of recurring task. The system consists of four robotic arms, which can be equipped with either minimally invasive instruments or a stereo camera. The master console provides a stereo view of the field of operation and two input devices can feed back forces to the user. We have utilized this system to assess the possibility of automating difficult handling tasks like surgical knot tying. In order to achieve this, a novel approach for human-machine skill transfer was developed. It constitutes an extension to learning by demonstration, which is a well known paradigm of robotic learning.**

***Index Terms*— learning by demonstration, skill transfer, robotic surgery**

I. Introduction

In recent years, minimally invasive surgery has drawn the attention of both surgeons and researchers on the field of robotics. While the former developed new techniques for surgical interventions in order to improve the treatment of patients, the latter have predominantly worked on the enhancement of the applicability of minimally invasive instruments. A remarkable example of such enhancements is the daVinci^{TM}machine [1], actually providing a complete telemanipulator for minimally invasive interventions. The instruments can be controlled remotely by a surgeon sitting at a master console, which can be placed somewhere in the operation theater. The master console is equipped with sophisticated input devices, which provide an intuitive handling of the instruments (Cartesian control without any chopstick effect). However, these advantages come at the price of reduced immersiveness, since the surgeon cannot feel any forces exerted onto the situs. In addition, working speed is significantly reduced, which renders these types of operations stressful for both patients and surgeons. In order to research on the improvement of these disadvantages, we have developed an experimental system for minimally invasive surgery incorporating force feedback and automation. While the results of the research on force feedback have already been published [2], we focus in this paper on automation of recurring tasks in minimally invasive surgery. We have chosen robotic knot-tying as a reference application since it combines many of the difficulties occurring in automation. One of them is the handling of limp objects like surgical threads, whose positional behavior can hardly be predicted. Another issue is the overall calibration of the kinematic chain, which is disturbed by many influences (like positional inaccuracy, distortion of the instruments and play).

II. Materials and Methods

The findings of this research project have been assessed within a realistic scenario of robotic heart surgery. As mentioned above, minimally invasive knot-tying has been chosen as a benchmark task, because it provides an extent of complexity, which requires new strategies for structuring and transferring information. Some other authors have already addressed the issue of surgical knot-tying [3], [4], [5], but they did not apply a knot-tying task under a combination of the following aggravating circumstances: manipulators with trocar kinematics, real suture material and real tissue. In addition, their work was rather focused on analyzing the knot-tying task instead of its automation.

A. Robotic System

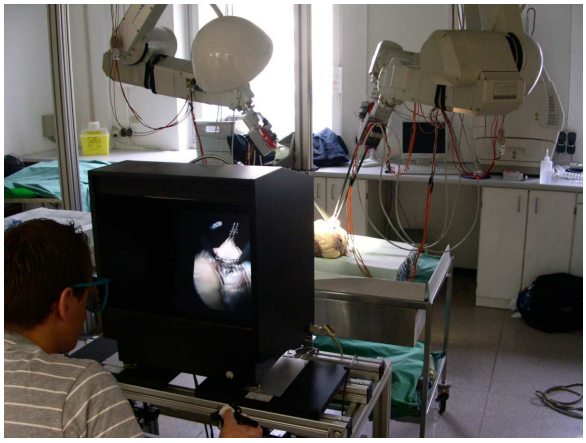

Fig. 1. **Hardware Setup:** Ceiling mounted robots with surgical instruments

We have conducted our experiments with a system for robot-assisted minimally invasive surgery, which was developed by the authors [6]. The slave manipulator of the system consists of four robots (Mitsubishi MELFA 6SL^{TM}, *Mitsubishi Electric Corp.*), which are mounted on a gantry on the ceiling (cf. fig. 1). The robots are equipped with minimally invasive instruments, which are originally deployed with the daVinci^{TM}surgical system (*Intuitive Surgical, Inc.*). The robot is concatenated with the surgical instrument by a magnetic coupling, which prevents the instrument from damages in case of a severe collision. The instruments are powered by small servo motors, which are integrated into the coupling mechanism. Optionally, one of the robots can

0-7695-3093-1/08 $25.00 © 2008 IEEE
DOI 10.1109/ICAS.2008.16

be equipped with a stereo camera instead of an instrument. The master console consists of an aluminum frame, which can be quickly adapted to new geometries. The user's place is located in front of the main in-/output devices, two PHANToM^{TM}haptic displays. Their controller boxes are stored at the base of the frame. The PHANToMs themselves are assembled upside down. This arranges for less constricted flexibility of the stylus pen. The 3D display is placed on top of the frame in a way not reducing the working space of the PHANToMs. As an additional input modality, foot switches are placed at the footwell of the console. One is dedicated as emergency exit while the function of the others can be arbitrarily engaged by software. In addition, forces occurring at the instruments are measured by strain gauge sensors and fed back by means of the haptic devices.

B. Generalization and Instantiation

In order to provide automation of certain recurring tasks, the user has to demonstrate the corresponding trajectories to the system. One prerequisite for the development of our application was the system being able to reproduce the task even after just one demonstration (which significantly distinguishes this approach from the ones of other authors [7]). Therefore, each trajectory is decomposed into smaller parts, socalled motion primitives [8], which constitute meaningful interactions of the end effectors with the environment. This decomposition is achieved by scanning the trajectory for prominent features like inflection points, occurring forces or rapid changes of the speed of the end effector. Since the description of these features is very general, it is not possible to base a sensible decomposition merely on them. Therefore, we use patterns of each task, which consist of a temporal ordered sequence of the features introduced above. These patterns can be easily constructed by the user and they are matched against the detected features in the trajectory. If a match is detected, the trajectory is decomposed as described by the pattern. Otherwise, the trajectory is rejected for not being a valid demonstration of the task.

Once a task has been demonstrated by the user with at least one valid performance of the skill, it is possible to instantiate the task at an arbitrary position in the working space. The only prerequisite is that the user establishes an appropriate initial condition. For our knot-tying example this means to pierce through the tissue and grab the needle with the right hand, and the loose end of the thread with the third instrument. Afterwards, the knot-tying task can be instantiated by pressing a foot switch. At this moment, the manual input of the user will be blocked and a skill is generated, which is suitable for the actual situation. After a preview of the execution of the skill was displayed, the corresponding trajectory can be actually carried out by the robots.

The instantiation of a task at the desired position is accomplished by the following steps: generation of the trajectories of each primitive of the task, connecting these trajectories to form a skill, and displaying a preview of the skill in the simulation environment. During the decomposition of trajectories, four types of primitives are identified: 2D movements, linear movements, force controlled movements and synchronized movements. We have developed different methods of instantiation for each type of primitive:

2D Movement: On the basis of the extracted features, we could instantiate this primitive by means of an adequately transformed 2D spline. Otherwise, the usage of splines has some significant shortcomings: The most important is that any shift of the interpolation points (which might be necessary due to the adaptation to obstacles in the new environment) can lead to unfavorable trajectories [9]. In addition, it is difficult to store temporal features like speed in 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 derive and generalize the corresponding information (a preliminary version of this method has been published in [6]). So far, there has been only little research on dynamical systems for the adaption and generation of motion primitives. Ijspeert et. al [10] have employed dynamical systems as generators for motion patterns in order to mimic locomotion of animals. A related approach has been proposed by Okada et. al. [11]. They have exploited the entrainment phenomenon of dynamical systems in order to stabilize motions of a humanoid robot. The desired trajectory is implemented as an attractor of the recursive definition of the robot's motion. Whenever the starting posture (i.e. the corresponding joint space configuration) lies outside the desired trajectory, the system converges back to it after a while. The system itself is invariant through time (constant vector field) and all parameters are known a-priori (no learning algorithm). They have successfully employed this procedure in combination with the formula of an inverted pendulum in order to stabilize the squat motion of a humanoid robot. Both approaches mentioned above operate on the joint-level of motion generation, whereas the proposed method generates trajectories in Cartesian space.

While we will use an online fluid simulation, some approaches utilize a stationary streamline function to plan trajectories for mobile robots [12], [13]. A 3D version of this so-called panel method [14] has been proposed by Zhang et al. [15] for motion planning of flying robots. Since computing power was quite limited at the time, these approaches has been revitalized lately by other authors [16]. Recently, there has also been work on analyzing trajectories by means of dynamical systems. Dixon et al. [17] 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 projects mentioned above 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 these 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 the 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. [18]):

$$\frac{\partial \overrightarrow{u}}{\partial t} + (\overrightarrow{u} \cdot \nabla)\overrightarrow{u} + \nabla p = \nu \Delta \overrightarrow{u} + \overrightarrow{f} \tag{1}$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0 \tag{2}$$

∇ 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. During feature extraction we have stored the plane parameters of the demonstration in the task-specific knowledge base. Therefore, equations 1 and 2 can be simplified to:

$$\frac{\partial u}{\partial t} + \frac{\partial p}{\partial x} = \nu \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) - \frac{\partial u^2}{\partial x} - \frac{\partial (uv)}{\partial y} + f_x \tag{3}$$

$$\frac{\partial v}{\partial t} + \frac{\partial p}{\partial y} = \nu \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) - \frac{\partial (uv)}{\partial x} - \frac{\partial v^2}{\partial y} + f_y \tag{4}$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{5}$$

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 equal 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. The actual implementation operates on a grid of at least 50×50 cells.

Evaluation of velocities is not centered within a cell, but distributed to a staggered grid in order to assure numeric stability. A detailed description of this methodology can be found in [19]. In addition to discretization, we have to fix the velocities at the boundaries of the simulated area. We will set them to zero, since we want the fluid to adhere to boundaries or objects within the stream. 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. Therefore, 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 still can collide. We will provide a solution to the problem of external collisions below.

With the help of these boundary conditions, we can also define obstacles within the area of simulation in order to adapt the trajectory to new environments. Each 2D movement primitive will be instantiated by a 2D fluid simulation, which is transformed onto the plane stored during feature extraction. Stirring of the fluid is achieved by sampling new points from the spline representation of the corresponding primitive. In this case, equidistant sampling with length d is applied. There is no analytical solution to guarantee an arc length of d for spline $S(x)$, since Δx cannot be determined from $S(x+\Delta x) = S(x)+d$ (note that Δx refers to a distance and has nothing to do with the Laplace operator in equation 1). So far, we have solved this problem by a binary search: Let Δx be an arbitrary initial value and $S(x)$ map to a coordinate within the cell with velocities u_{ij} and v_{ij}. Then, we check if $|(S(x + \Delta x)) - (S(x) + d)|$ falls below a predefined threshold ϵ. If $(S(x + \Delta x)) - (S(x) + d)$ yields a positive number exceeding ϵ, we try $S(x + 0.5\Delta x)$. Otherwise, if $(S(x + \Delta x)) - (S(x) + d)$ yields a negative number less than $-\epsilon$, we try $S(x - 0.5\Delta x)$, and so forth. Normally, this binary search converges in less than five steps, if ϵ has been chosen appropriately. Afterwards, we have to test if $S(x+\Delta x)$ lies within a fluid cell. If the point lies outside the simulation area or within an obstacle cell, we project it to the nearest valid point within the fluid. 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 features of the original trajectory, which has been stored in the task-specific part of the knowledge base. Afterwards, we interpolate the neighboring values of u_{ij}, v_{ij}, u_{ij+1} and v_{i+1j}; i.e. we calculate a preset for these velocities at time step t_n. All other velocities within the grid a derived from fluid simulation. We can now throw a particle into the stream and it will be attracted by the instantiated trajectory of the corresponding primitive. Since we know the position and orientation of the simulation grid, we can generate suitable 3D points in order to instantiate the primitive in the new environment (cf. fig. 2). Since the simulation is only refreshed at discrete points in time ($t_0 < t_n < t_{max}$), we have to apply an interpolation again in order to get positions at arbitrary points in time. Fortunately, this works well even for tiny time steps.

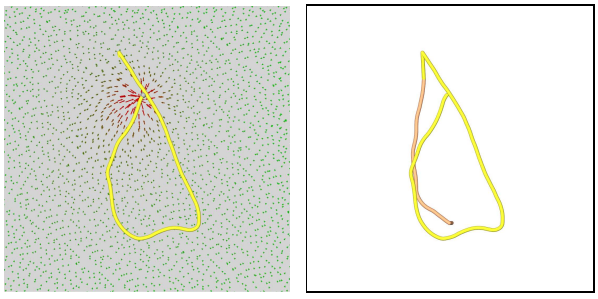

Fig. 2. 2D primitive after fluid simulation (left) and embedded into the skill (right)

Linear Movement: The instantiation of linear movements is comparably easy. All we need is shifting the start and end point to the desired position and connect them with a straight line (cf. purple parts of fig. 3).

Force Controlled Movement: Although this primitive is basically a straight line movement, its instantiation yields some further issues. We have extracted the maximum force and the end points of the corresponding primitives. An intuitive instantiation is moving the gripper along the line between start and end point, until the desired force occurs. In practice, it might happen that the gripper has reached the end point before the maximum force is reached. Therefore, we make allowance for this by elongating the line by 25% of its original length (cf. fig. 3, right end of the green trajectory). When the corresponding skill is performed, the gripper is opened once the recorded force is exceeded. Regardless of the position of opening, the gripper moves to the end of the line. This behavior guarantees a deterministic execution of the skill, which can be reviewed during simulation (i.e. without forces actually applied). Afterwards, the gripper has to move back to the original end of the movement (without the 25% extension) in order to continue.

Fig. 3. Instantiation of a force controlled movement

Synchronized Movement: A synchronized movement can only exist in connection with a 2D movement. A mapping function between the synchronized movement and the corresponding 2D movement is stored during feature extraction. This can be used now to produce the points of the synchronized motion. The mapping function is applied to every point of the 2D primitive we want to synchronize with (cf. fig. 4). Most probably, the starting point of the synchronized primitive will not coincide with the end point of the preceding primitive of the skill. Therefore, both have to be concatenated by a straight line movement.

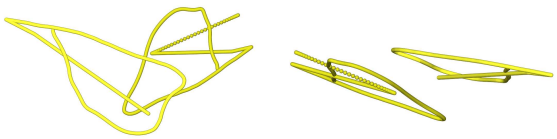

Fig. 4. Synchronized primitive (dark yellow) and reference movement (light yellow)

C. Simulation environment

For an offline evaluation of the newly generated trajectories, a simulation environment of the system has been developed (cf. fig. 5). The GUI comprises an interface to a 3D model of the scene, which can be manipulated in realtime. For each object in the scene, a context menu can be displayed (on the left side) by clicking on the corresponding model. This provides a possibility to adjust the parameters of the underlying object. For example the joint angles of the robots can be altered this way. In addition, the simulation environment also incorporates a key framing module and all

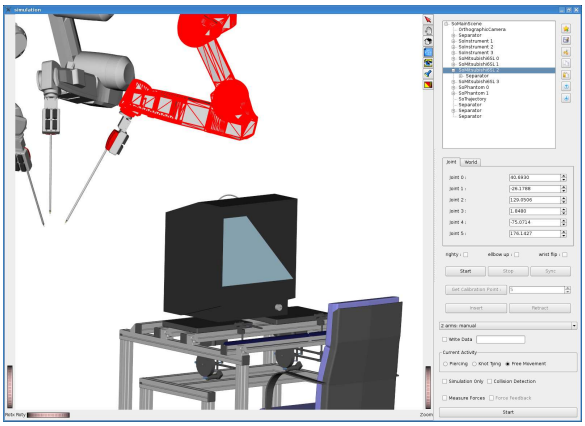

Fig. 5. **simulation environment:** Interventions can be assessed offline

trajectories constructed in the simulation can be checked by a collision detection.

D. Virtual obstacles generated by the GPU pipeline

One major issue, which we have not addressed so far, is the treatment of possible collisions of the robots. While direct collisions of the end effectors and objects in the working space can easily be modeled by obstacle cells, handling external collisions requires much more effort. Movements of the end effector in its working space might induce collisions of parts of the robots carrying the instruments. In our path planning method for 2D movements (the fluid simulation), the working space of the end effector is discretized to a grid. Therefore, we can check each cell of this grid for external collisions, and if so, mark the corresponding cell as so-called **virtual obstacle**, which will be treated like a normal obstacle during simulation. In order to achieve this, each cell is subdivided into 5×5 subcells, allowing for a better resolution. The end effector is set to the 3D position of each subcell (the grid is already transformed to the desired position) and the inverse kinematics is applied to this 3D point. The resulting angles are applied to the robots and the new configuration is checked for collisions by means of an oriented bounding box test. If any collision occur in one of its subcells, the corresponding cell will be treated as virtual obstacle. The complete procedure of deriving virtual obstacles is depicted in fig. 6.

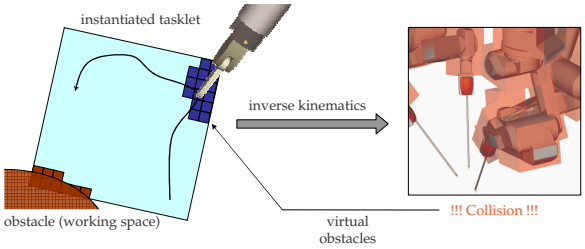

Fig. 6. Generation of virtual obstacles for collision detection

For our application examples we have used a grid of

50×50 cells, each of them featuring 5×5 subcells. In order to generate virtual obstacles for this example, the inverse kinematics and the collision test has to be applied $250 \times 250 = 62500$ times, which renders this method computationally quite expensive. Therefore, we have developed an implementation, which exploits the computing power of the graphics processing unit (GPU) of modern graphics cards. We use certain features of the *OpenGL* pipeline of those cards, which is depicted in fig. 7.

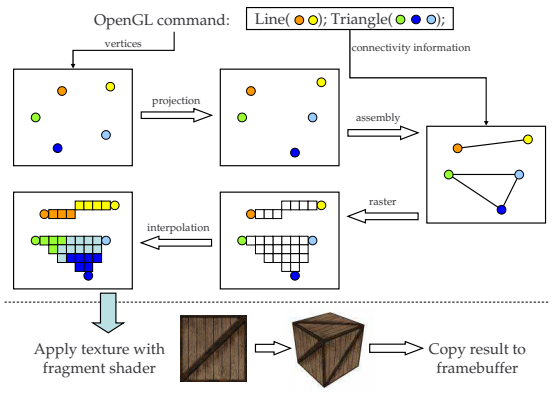

Fig. 7. *OpenGL* pipeline for rendering of 3D objects [*source: Lighthouse3D.com*]

Usually 3D related programs interact with the graphics card via *OpenGL* commands. Basically, these commands provide the card with new geometric information of a scene, which is going to be displayed on the screen. This transformation of 3D information into a 2D viewport image on the frame buffer is called rendering. In short, the rendering process is divided into two major parts: rasterization and texturing. Rasterization is the process of converting the 3D vector-based object of the scene into pixel-based fragments on a 2D plane. Those are forwarded to the texture engine, which optionally projects predefined images onto the fragments (in most cases images to emulate certain surface materials like stone or wood etc.). For most currently distributed cards, both processes rasterization and texturing can be controlled by interchangeable code segments, vertex shaders and fragment shaders, respectively.

We have exploited this principle to implement a fast generation of virtual obstacles (cf. fig. 8). When a 2D primitive is instantiated by means of fluid simulation, the simulation grid is transformed to the right place. Therefore, its corners span a rectangle in 3D space. We will refer to the corners as points $[(x_1, y_1, z_1), ..., (x_4, y_4, z_4)]$. In order to initialize our algorithm, we paint an *OpenGL* rectangle, which is always parallel to the viewport (i.e. it will be projected on a rectangle in the frame buffer). We set the colors of its corners to values $[(x_1, y_1, z_1), ..., (x_4, y_4, z_4)]$, i.e. the coordinates of the corners of the simulation grid are interpreted as colors of the corners of the painted rectangle. In addition, we replace the default vertex shader of the graphics card with an own version. This program simply tells the card to linearly interpolate the colors of the corners when coloring the rasterized fragments. If we access the colors of the fragments[1] in the buffer of the card now, we get the interpolated coordinates on the simulation grid.

We can control the resolution of the grid by adjusting the rectangle painted with the *OpenGL* interface (e.g. if we want to have a resolution of 250×250 subcells, we simply draw a rectangle of 250×250 pixels. Each fragment of the rectangle will be forwarded to the texture engine in order to apply the fragment shader. Theoretically, this process is performed in parallel for all fragments in the buffer, but practically, parallelization is limited by the number of shading units (up to 96 on currently distributed graphics cards). The program of the fragment shader can access all properties of a single fragment in the pipeline. Therefore, we can access the color information of the fragments and use it as input for the shader we have installed on the graphics card. Our version of the fragment shader interprets the color information of the fragments as coordinates on the simulation grid (i.e. positions of the end effector of our system).

Now, we can perform the inverse kinematics of our system in order to derive the angle of the robot bearing the corresponding end effector. Once these angles are available, we can apply an OBB collision detection. We can check for collisions with objects, whose positions are enlisted in the fragment shader. Therefore, collision checks are currently limited to one robot. The results of the collision checks are painted on the frame buffer. Afterwards, we can access the pixels on the frame buffer and generate virtual obstacles from it (cf. fig. 8). During generation, certain preliminaries have to be met, e.g. all obstacle cells need at least two direct neighbors in order to guarantee stability of the simulation algorithm.

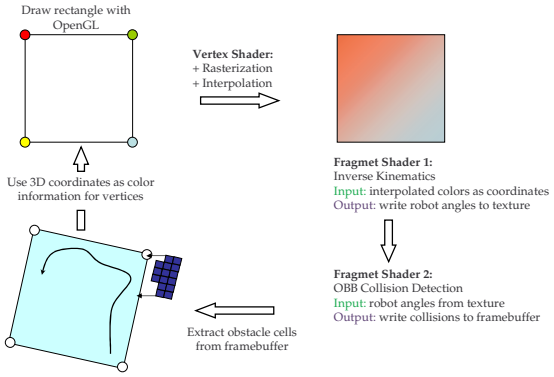

Fig. 8. Generate virtual obstacles with OpenGL pipeline

III. Results

After the collision-free primitives are instantiated in a new environment by the methods introduced above, we have access to separated trajectories for each gripper. The next step is to synchronize the trajectories as it is required

[1]A fragment refers to a pixel in the *OpenGL* pipeline of the graphics card, which is augmented with certain features like color information, depth, surface normal etc.

by the underlying task. For example, before the first 2D movement (winding of the thread) of the automated knot can be performed, we have to wait until the left hand gripper has finished its linear movement towards the winding point. Such waiting times are inserted by simply duplicating the corresponding position in the trajectory of the gripper, which has to wait. This is always possible, since the primitives in a skill are independent from each other, i.e. the gripper stops after the execution of each primitive.

Once the primitives are synchronized, the trajectory of the complete skill can be displayed in the simulation environment (cf. fig. 9). Optionally, it is possible to simulate the movements of the instruments and robots during performance of the skill. This is helpful to detect errors in the knot-tying process or to reveal additional collisions. If all checks have been performed successfully, the trajectory can be carried out in the real-world scenario. After completing the automated knot-tying procedure, the user can continue with manual control. So far we have reached a success rate of approx. 50% for the application of knot-tying within our scenario. Though this rate might be comparably small, it shows that automation is possible in minimally invasive robotic systems.

IV. Conclusion

We have presented an approach of learning by demonstration, which is based on a single demonstration of the task. Our reference application for automation is minimally invasive knot-tying. User demonstrations are decomposed into meaningful primitives by matching user generated patterns against features in the trajectory. Any primitive, which has been generated this way, can be applied to new environments. For 2D movements we have developed an approach based on fluid dynamics. The demonstration is used to stir a fluid. A locally adjusted trajectory of this movement can be generated by throwing a particle into the stream. In addition, this method provides an intrinsic obstacle avoidance, since no streamline can intersect with obstacles. In order to consider external collisions of the robots bearing the end effectors, we have introduced the concept of virtual obstacles, which can be treated analogously with the same algorithm. The presented framework was successfully applied to the real-world application of robotic knot-tying. We are planning to increase the success rate of skill application by a proper revision of the hardware. Therefore, we hope to get rid of mechanical play and increase the quality of calibration.

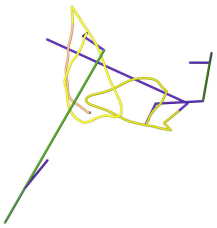

Fig. 9. **Task application in new environment**

References

[1] G. Guthart and J. Salisbury. The Intuitive™ Telesurgery System: Overview and Application. *Proceedings of the IEEE International Conference on Robotics and Automation*; San Francisco, USA; 2000, pp. 618-621.

[2] H. Mayer, I. Nagy, A. Knoll, E.U. Braun, R. Bauernschmitt and R. Lange. Haptic Feedback in a Telepresence System for Endoscopic Heart Surgery. *MIT PRESENCE: Teleoperators and Virtual Environments* ; MIT press, vol. 16, no. 5, pp. 459-470, 2007.

[3] J.Takamatsu, T. Morita, K. Ogawara, H. Kimura and K. Ikeuchi. Representation for Knot-Tying Tasks. *IEEE Transaction on Robotics*; IEEE Robotics and Automation Society, vol. 22, no. 1, pp. 65-78, 2006.

[4] 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.

[5] 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.

[6] H. Mayer, I. Nagy, A. Knoll, E.U. Braun, R. Lange and R. Bauernschmitt. Adaptive Control for Human-Robot Skilltransfer: Trajectory Planning Based on Fluid Dynamics. *Proceedings of the IEEE International Conference on Robotics and Automation*; Rome, Italy; 2007, pp. 1800-1807.

[7] K. Ogawara, J. Takamatsu, H. Kimura and K. Ikeuchi. Estimation of Essential Interaction from Multiple Demonstrations; *In Proceedings of the IEEE International Conference on Robotics and Automation*; Taipei, Taiwan; 2003, pp. 3893-3898.

[8] M. Matarić. Sensory-Motor Primitives as a Basis for Learning by Imitation: Linking Perception to Action and Biology to Robotics. Imitation in Animals and Artifacts, K. Dautenhahn and C. Nehaniv, editors; MIT Press, Cambridge, MA, 2002, pp. 392-422.

[9] 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.

[10] 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.

[11] M. Okada, K. Osato and Y. Nakamura. Motion Emergency of Humanoid Robots by an Attractor Design of a Nonlinear Dynamics. *Proceedings of the IEEE International Conference on Robotics and Automation*; Barcelona, Spain; 2005, pp. 18-23.

[12] J. Decuyper and D. Keymeulen. A Reactive Robot Navigation System Based on a Fluid Dynamics Metaphor. *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature (Lecture Notes in Computer Science)*; Springer-Verlag, vol. 496, pp. 356-362, 1990.

[13] J. Kim and P. Khosla. Real-time Obstacle Avoidance Using Harmonic Potential Functions. *IEEE Transactions on Robotics and Automation*; vol. 8, no. 3, pp. 338-349, 1992.

[14] J. Hess. Review of Integral-Equation Techniques for Solving Potential-Flow Problems with Emphasis on the Surface Method. *Computer Methods in Applied Mechanics and Engineering*; vol. 5, pp. 145-196, 1975.

[15] Y. Zhang and K. Valavanis. A 3-D Potential Panel Method for Robot Motion Planning. *Robotica* ; vol. 15, no. 4, pp. 421-434, 1997.

[16] S. Waydo and R. Murray. Vehicle Motion Planning Using Stream Functions. *Proceedings of the IEEE International Conference on Robotics and Automation*; Taipei, Taiwan, pp. 2484-2491, 2003.

[17] K. Dixon and P. Khosla. Trajectory Representation Using Sequenced Linear Dynamical Systems. *Proceedings of the IEEE International Conference on Robotics and Automation*; Barcelona, Spain; 2005, pp. 3925-3930.

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

[19] T. Griebel, T. Dornseifer and T. Neunhoeffer. Numerical simulation in fluid dynamics. A practical introduction. *SIAM Monographs on Mathematical Modeling and Computation (3)*; Society for Industrial and Applied Mathematics, Philadelphia, USA, 1997.