Technische Universität München

Forschungs- und Lehreinheit Informatik VI: Robotics and Embedded Systems

Human-Machine Skill Transfer in Robot Assisted, Minimally Invasive Surgery

Hermann Georg Mayer

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:		Univ.-Prof. N. Navab, Ph.D.
Prüfer der Dissertation:	1.	Univ.-Prof. Dr. A. Knoll
	2.	Prof. Dr. Eng. M. Mitsuishi (University of Tokyo, Japan)

Die Dissertation wurde am 17. September 2007 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 14. April 2008 angenommen.

Acknowledgement

Special Thanks to:

Prof. Alois Knoll and Prof. Mamoru Mitsuishi for supervizing my thesis
Prof. Nassir Navab for organizing the examination process
Dr. Gerhard Schrott, Amy Bücherl, Gisela Hibsch and Monika Knürr for supporting me

all my colleagues, especially the crew of the robot lab, Istvan Nagy and Stefan Riesner

all my friends and my family, especially my parents Therese and Hermann Josef Mayer

Abstract

The introduction of minimally invasive surgery into surgical procedures has benefited the well-being of patients, particularly in heart surgery. Before that, due to the heart being occluded by the ribcage, it was often necessary to cut the whole sternum in order to access the region of operation. Now, only small cuts suffice to perform the same operation with endoscopic instruments. Therefore, the convalescence of patients is accelerated, postoperative infections have become rarer, and last but not least the costs for medical aftercare are significantly reduced. However, the introduction of minimally invasive surgery has also posed some new difficulties: The sight on the region of operation is reduced to an area, which can be covered by endoscopes. Together with the “chopstick effect” of endoscopic instruments, this leads to an encumbered hand-eye coordination. In addition, the inevitable tremor of the human hand is amplified by leverage effects. The development of robotic manipulators for minimally invasive surgery has contributed to overcome most of these inconveniences, but, on its part, it has created new challenges. Since the instruments are decoupled from direct control by the human hand, the surgeon perceives no haptic feedback regarding the currently performed actions. Therefore, visual compensation is necessary, which in turn leads to an accelerated fatigue of the operator. In addition, missing force feedback often induces damages of soft tissue and suture material.

Motivated by these challenges, we have developed an experimental system for robot assisted, minimally invasive surgery. It comprises four manipulators (three endoscopic grippers and a stereo camera) and a master console equipped with 3D stereo vision. The main purpose of this project was the evaluation of the impact of force feedback on surgical procedures and the inclusion of automated routines into the control loop of the system. The former was achieved by equipping the endoscopic instruments with force sensors and subsequently feeding back the measurements to haptic devices at the master console. The latter, representing the main focus of this thesis, was realized by providing an automated knot-tying procedure, which can be applied by surgeons for assistance. This kind of automation was implemented by a novel approach to human-machine skill transfer. The methodology itself is independent from this specific set-up (robot assisted surgery), but can be used in any scenario involving manually controllable robots. The user has to provide a high level description of the skill, which is going to be transferred to the machine. This is realized by an abstract pattern comprising significant features of the working environment (e.g. geometric restrictions, maximum forces etc.). Afterwards, the abstract pattern of a skill will be concretized by various user demonstrations. I.e. user demonstrations are matched against that pattern and decomposed into movement primitives, which are charted for relevant information about the task. These primitives will be utilized to perform the demonstrated skill in new, previously unknown environments. The major challenge in this scenario was the performance of a surgical knot at a new location, with unknown features of the suture material, and with instruments exhibiting imprecise calibration and unpredictable play. However, both research goals, incorporation of force feedback and automated knot-tying, have been performed in a surgical scenario involving realistic operations on pig hearts.

Motivation of the thesis

Zusammenfassung

Die Entwicklung minimalinvasiver Operationstechniken hat sich zunächst äußerst positiv auf die Patienten ausgewirkt. Dies gilt insbesondere für die Herzchirurgie, da hier der Zugang zum Operationsgebiet durch den Brustkorb eingeschränkt wird. Daher waren Eingriffe am Herzen in der Vergangenheit oft mit einer Durchtrennung des Brustbeins oder der Rippen verbunden, was für den Patienten eine langwierige Heilungsphase nach sich zog. Durch den Einsatz endoskopischer Instrumente kann das Gewebstrauma auf ein Minimum (kleine Schnitte zum Einführen der Instrumente) beschränkt bleiben. Der Heilungsprozess wird zudem dadurch begünstigt, dass es bei dieser Art von Eingriffen seltener zu Infektionen oder ähnlichen Komplikationen kommt. Der Nachteil dieser Technik besteht jedoch darin, dass der Einsatz der Instrumente eine ungewohnte Hand-Auge Koordination erfordert und das Zittern der Hand durch den Hebeleffekt verstärkt wird. Zudem ist die Sicht auf das Operationsfeld stark eingeschränkt. Diese Schwierigkeiten wurden durch den Einsatz von Telemanipulatoren weitgehend überwunden, allerdings zum Preis neuer Nachteile. So werden die Instrumente nun nicht mehr direkt durch den Chirurgen geführt, sondern Roboter übertragen die Bewegungen der Eingabegeräte. Daher kann der Chirurg keine Kräfte oder Oberflächenbeschaffenheiten mehr fühlen. Die fehlende Sensorik kann zwar zum Teil visuell kompensiert werden, allerdings beschleunigt sich dadurch die Ermüdung des Chirurgen.

Angesichts dieser Herausforderungen haben wir beschlossen, ein eigenes robotergestütztes Chirurgiesystem zu entwickeln. Dieses besteht aus vier Manipulatorarmen, die jeweils mit einem minimalinvasiven Instrument oder einem Stereoendoskop ausgestattet werden können. Zudem haben wir eine Masterkonsole entwickelt, mit der die Instrumente gesteuert werden und an der, die an den Instrumenten auftretenden Kräfte zurückgekoppelt werden können. Zusätzlich werden die Bilder der Stereokamera mit einem 3D-Bildschirm angezeigt. Die beiden Hauptaufgaben, die das System erfüllen sollte, waren die Evaluation der Kraftrückkopplung und die Einbindung teilautonomer Sequenzen in chirurgische Arbeitsabläufe. Letzteres konnte durch eine neuartige Technik für den Mensch-Maschine Skilltransfer ($=$ Übertragung von Fertigkeiten) realisiert werden. Die Methodik ist unabhängig vom vorliegenden Szenario und kann in allen Umgebungen eingesetzt werden, in denen eine manuelle Steuerung von Robotern möglich ist. Der Benutzer legt zunächst ein abstraktes Muster der zu übertragenden Fertigkeit an. Dieses ist noch nicht genau genug um die Tätigkeit tatsächlich ausführen zu können, sondern enthält allgemeine Angaben wie Maximalkräfte oder Einschränkungen des Arbeitsraums. Danach werden die Fertigkeiten durch Vorführungen des Benutzers konkretisiert. Diese Vorführungen werden anhand des abstrakten Musters validiert und in Bewegungsprimitiva zerlegt. Die Primitiva können wiederum dazu benutzt werden, um die Fertigkeit in einer neuen, vorher unbekannten Umgebung auszuführen. Für die Umsetzung der Primitiva wurde eine neue Technik entwickelt, die auf der Simulation von Fluiddynamik basiert. Die genannten Techniken und auch die oben angesprochene Kraftrückkopplung wurden in einem realistischen Chirurgieszenario an Schweineherzen erfolgreich angewandt.

Contents

Chapter 1

Introduction

In this thesis we present a novel framework for human-machine skill transfer. This approach is based on the well-known learning by demonstration paradigm. Demonstrations are decomposed into primitives by means of abstract motion patterns. The application of skills is guided by a knowledge base, comprising self-contained, spatio-temporal schemes for the execution of movement primitives. Complex primitives are applied by means of dynamic vector fields (similar to those applied to fluid dynamics problems). Due to this knowledge-driven instantiation, only one demonstration by the user is necessary for successful application of the skill.

We have chosen surgical knot-tying as an application example for our framework, providing all criteria of a challenging learning task. Therefore, we have implemented a robotic system for minimally invasive surgery. Apart from automated knot-tying, we have employed this system to perform experimental surgical tasks, like the closure of an atrial septum defect. In addition, we provide an intuitive user interface for planning and performing operations.

1.1 Machine Learning

Ever since the development of complex tools and machines, their inventors have always contrived possibilities to make them easily operated by a broad variety of users. In order to achieve this goal, it is necessary to hide as much complexity as possible behind comfortable and standardized user interfaces. This proceeding has promoted the acceptance of new technologies and it has helped to make such products a commercial success, which today is also an important criterion. The probably most famous example following this strategy is the car. Today, cars come in quite different varieties and layouts, but they all share the same concepts of user interaction: a steering wheel for direction control, pedals for acceleration and braking, and an interface to adapt the gear ratio (either manual or automatic). This standardization rendered millions of people all over the world possible to learn to drive cars just after a short training and it turned the car into a promoter of a whole sector of industry. Other famous examples can be easily identified, like household appliances or electrical equipment.

While all devices mentioned above serve one unique purpose (like driving, dish washing or playing CDs), it is much more difficult to design intuitive user interfaces for multi-purpose devices like computers, machines for computer aided manufacturing (CAM) or robots. Today, the functionality of most

computers is abstracted by operating systems, which are still quite complex and require extensive training before they can be mastered by end users. The same applies to complex CAM-machines, which play an important role in modern manufacturing processes. Therefore, users of such machines have to become experts in handling the operation of the machine itself, before it can be employed in the value-added process of the company. If the operation procedure of the machine could be simplified and standardized, the employment of machines would be more effectively focused on production and it is easier to keep pace with emerging technologies. This conclusion particularly applies to robots, because they are most versatile CAM-machines controlled by complex computer systems.

A first step towards intuitive human-robot interaction was the concept of “Teaching” (also “Teach-In”). In its basic form, “Teaching” means to instruct a certain trajectory by moving the robot with a control stick. A human operator transfers a skill to perform a certain task onto a robot by means of demonstrating the underlying trajectory. The trajectory is saved and can be repeated in place. Being the most obvious advantage of this technique, the user does not have to be an expert in programming robots. In addition, re-programming robots this way is very fast and straightforward, as long as the application environment does not change (e.g. the target position of a pick-and-place action is fixed etc.). This requirement constitutes a narrow limit for the adoption of “Teaching” in previously unknown environments. In most of the cases it is not possible to simply shift the trajectory to another position, since the absolute accuracy of robots is limited. Therefore, researchers in the field of robotics have tried to formulate more abstract concepts for skill transfer, detached from the demonstration of a concrete trajectory. An obvious step was to provide the user with a toolkit of pre-formulated trajectories (so-called primitives), which can be combined to a skill in order to perform a certain task. A similar phenomenon can be observed in nature: if special cells in the spinal cord of a frog are excited, the locomotor system reacts with a linear movement of the leg [186, 95] (cf. fig. 1.1). Obviously, the inverse kinematics enabling such a behavior is part of an intrinsic system, which can be accessed by higher level functions. Transferred to robotics, this paradigm means to assemble task-specific skills from a set of sensori-motor primitives. The embodiment of certain primitives can be quite complex, if information from different sensors has to be included: for instance in order to provide primitives for controlling a multifingered hand [124] or for the force-controlled insertion of a peg into a hole [117]. Regarding these examples, the primitives had to be pre-programmed by an experienced user. This restricts the composition of new skills to some combinatorial variations of given primitives [107]. Therefore, later approaches tried to derive primitives from user demonstrations in order to provide more flexibility [161]. In its original form, this paradigm implies that primitives have to be detected in user demonstrations without any contextual information, only based on cues extracted from unsupervised data mining like cluster analysis (e.g. k-means algorithm), inflexion points of the trajectory or occurring contacts (if those are detected by force sensors). This method turned out to be too unspecific in order to derive useful primitives [172]. At most, very simple primitives, like getting into contact with a surface or linear movements, have been successfully derived from user demonstrations this way. In contrast, if too much spatio-temporal information is included into the detection algorithm, the approach will be restricted to a specific task (like learning the movements to balance an inverted pendulum [58]). This discussion seems to end up in a vicious circle: if less information is included

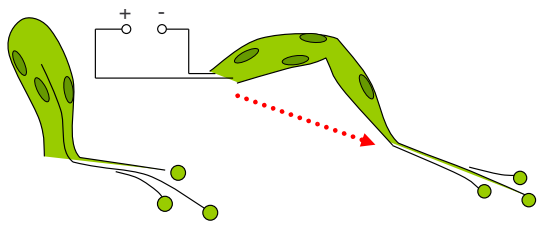


Figure 1.1: Bio-inspired primitives

bio-inspired movement primitives

into the system, potentially detected primitives may be too simple; if more information is included, the whole procedure cannot be adapted to previously unknown tasks any more. An interesting hint how to overcome these hitches has been given by the research group of Atkeson [61]. Instead of using primitives to assemble new skills, they have employed them to accelerate learning tasks. They augmented their learning system with pre-programmed primitives. Later, all user demonstrations are checked for occurrences of these primitives, which will in turn serve as inputs for their learning algorithm. A generalized version of this approach is the application of Hidden Markov Models (HMMs) in order to recognize primitives in user demonstrations [104, 169]. The states of the HMM are described by certain events in the application domain (like contacts with surfaces, pressing a switch etc.). By means of this discrete state machine, user demonstrations can be decomposed into transitions between those states. While HMMs provide reasonable results in the classification of human handwritings (as well as speech recognition), their usability in primitive segmentation is limited. This is mostly due to them being focused on the description of states, but a detailed description of transitions can hardly be included into this model. And after all, motion primitives basically are transitions. Therefore, we propose an approach, which is on the one hand based on state machines (like HMMs), providing stable pre- and post-conditions, but on the other hand is more flexible by replacing unstructured transitions with abstract patterns. These abstract patterns initially contain no spatio-temporal information and can be used to compose new (abstract) skills. Afterwards, user demonstrations are matched against these abstract skills. If an occurrence is detected, the spatio-temporal information of the demonstration can be used to turn abstract patterns into primitives and abstract skills into applicable skills, respectively.

Once the abstract skills are augmented with specific information from the demonstrations, and thus ready for application, the question is, how the primitives can be executed and best be adapted to new environments? One solution would be to simply omit this issue by replaying a trajectory extracted from the demonstrations at the desired position. Apart from the decomposition of the task into primitives and possibly some smoothing, this will lead to results similar to “Teaching”. Another possibility is the adoption of neural networks in order to improve generalization of primitives in new environments. Unfortunately, neural networks need lots of demonstrations before they can produce well generalized results [14].

Since new environments may contain obstacles, potential fields can be employed to drive proper adjustment of motion primitives [119]. While potential fields provide a straightforward approach for obstacle avoidance, their application is limited by the following disadvantages: movements between several obstacles in the working space may lead to an oscillatory behavior, particularly if there are narrow alleyways between different obstacles. Movements along the borders of an obstacle are not possible, since the potential field will cause an orthogonal deflection. Some more limitations can be found below in chapter 5. In order to overcome these limitations, we propose the application of dynamic vector fields in order to apply primitives to new environments. Furthermore, this approach allows the inclusion of spatio-temporal information derived from multiple demonstrations, it provides generalization as known from neural networks and it can emulate attractive potential fields as well. The major disadvantage of the approach is its complexity, regarding implementation as well as computation time. Fortunately, dynamic vector fields are subject of intensive research (especially in the field of fluid dynamics) and therefore, lots of algorithms and implementations already exist and can easily be transferred to other application domains. In addition, most algorithms can be parallelized (at least the one we have implemented), which makes this a forward-looking approach with regard to the

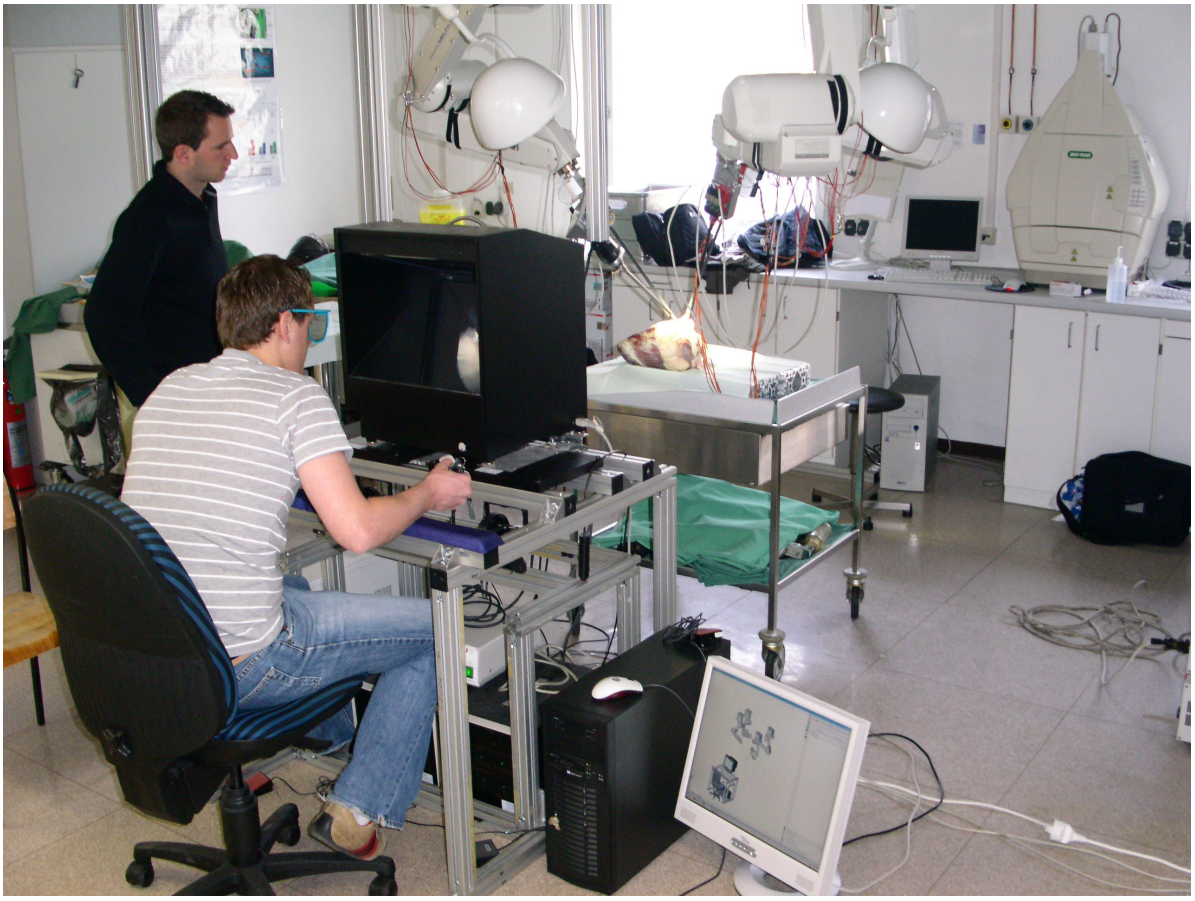


Figure 1.2: System set-up at the German Heart Center Munich

fast proliferation of multi-processor and multi-core hardware.

1.2 Application Scenario: Robotic Surgery

Our application scenario is a robotic system for endoscopic surgery, which constitutes a benchmark platform for skill transfer techniques due to several challenges:

1. Multi-actuator, multi-sensor scenarios like this pose strict requirements on synchronization and sensor integration.
2. Handling of limp and unstructured objects like sewing material and soft tissue whose properties can be hardly predicted.
3. Calibration is difficult due to the high number of unknown disturbance variables, and therefore, fault-tolerant methods are required.
4. Strict timing targets on different levels of the system architecture (e.g. real-time force feedback and synchronized movements of the end effectors in order to perform surgical tasks).

Like comparable tele-manipulators for robotic surgery [97, 129] our system consists of a master console and slave actuators (cf. chapter 2). The master console offers 3D-vision, 6DoF input, and force feedback (cf. fig. 1.2). The actuators are composed of a 6DoF robot and an endoscopic instrument (or camera), constituting an 8DoF kinematic chain. Therefore, operations can be carried out via trocar kinematics (cf. chapter 3.3). All endoscopic instruments are equipped with strain gauge sensors. Sensor readings are amplified and translated to the corresponding coordinate frame of the force feedback devices. The importance of force feedback in endoscopic surgery has been proven by an extensive evaluation of the system (cf. chapter 4). The software of the system (cf. chapter 3) is structured into different modules and arranges for all kinds of interactions with the system. We included a key-framing module, which can be used for intuitive programming of trajectories: significant positions of the end effector can be stored as key-frames and an interpolation engine provides path planning in between. Via context sensitive menus, all robots can be controlled remotely. In real-time mode, the actuators can be controlled by the haptic input devices, thus providing an interface for bi-manual operation. Trajectories can be stored, replayed or analyzed for further processing. All software components follow a strict model-view-controller architecture and, if applicable, are represented by a 3D model in an interactive scene. In addition, the visualization module comprises basic CAD functions, allowing for the adaption of the displayed scene graph. The control architecture of our software is based on abstract classes. Therefore, integration of new hardware (particularly robots and instruments) can be handled quite easily. Anyway, the main focus of this work will be on human-robot skill transfer in this scenario of robotic surgery. Practicability of the techniques developed in chapter 5 will be evaluated with the example of surgical knot-tying.

The procedure of **surgical knot-tying** is depicted in fig. 1.3: first, with the dominant gripper, the thread

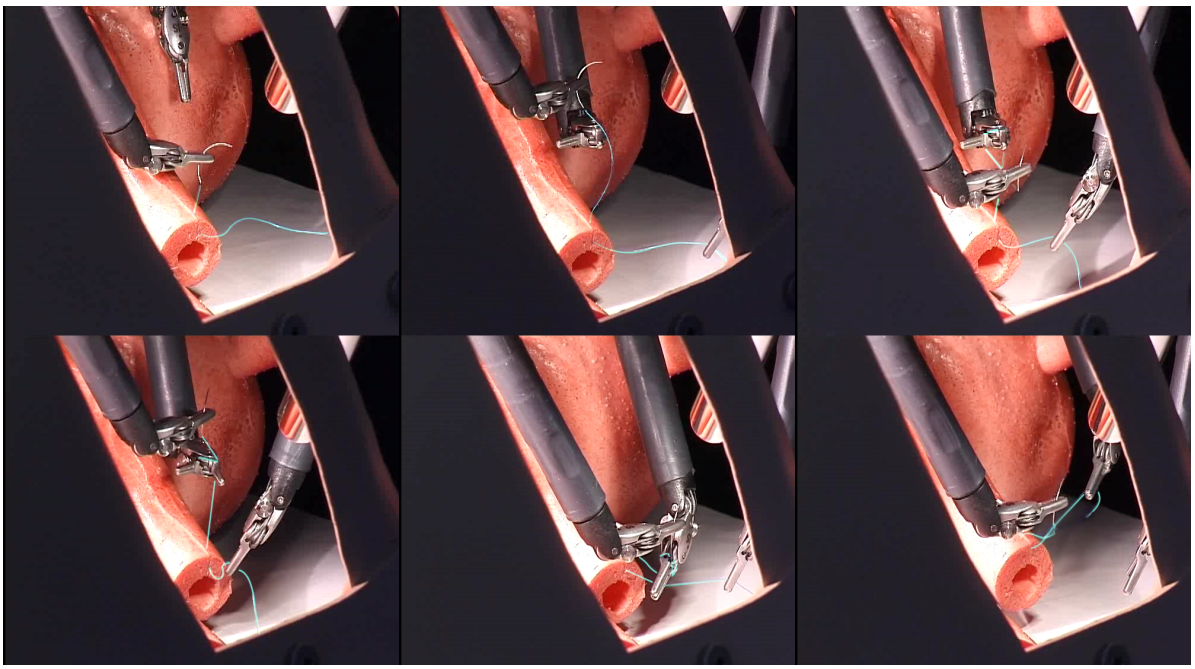


Figure 1.3: Subsequent steps of surgical knot-tying with endoscopic instruments

is pulled to get enough material for winding. Then, the thread is winded about the subordinate gripper. Once one or two loops are formed this way, the subordinate gripper grasps the end of the thread, which is hold by an assistant gripper in this example. Although full automation of this task would be

surgical knot-tying

technically feasible, we restricted our approach to semi-automated knot-tying for safety reasons: the surgeon has to find a convenient location to place the knot, pierce the tissue and decide whether or not the knot can be tied with the specific parameters. A preview module incorporated in the simulation environment serves as basis for decision making. I.e. after preparing the trajectory for knot-tying, the whole process is pre-displayed in the virtual environment, where possibly harmful interactions with the tissue or collisions may be detected before the trajectory is actually performed. The trajectory itself is an instance of a skill adapted to a specific working environment.

1.3 Skill Transfer: Situated Dialog and Scaffolding

As mentioned above, skills are composed of abstract primitives, which are concretized by user demonstrations; i.e the user shows the system how to perform a certain primitive. In order to alleviate communication, human-computer interaction is realized as a situated dialog. **Situated learning** has emerged in psycho-linguistics and was adopted for human-computer interaction in the late eighties [201]. Its basic proposition is that effective learning of applied tasks can only proceed, if teacher and trainee share relevant information about the learning environment. Transferred to human-computer interaction this implies that descriptions of tasks are based on common sensori-motor abilities. For example, a human cannot demonstrate a skill requiring optical cues (like the orientation of an object) to a system without visual perception. Otherwise, the user cannot be expected to demonstrate tasks exceeding the limits of human motor functions (at most, bi-manual, Cartesian movements). Or in other words, skills have to be demonstrated within the working domain and without requiring extrinsic knowledge. Systems based on situated dialogs between humans and machines have already proven effective in learning tasks [226, 130]. In our case, human-machine interaction divides into two consecutive components: an abstract description of the task and its augmentation with specific demonstrations. This strategy can be understood as a dialog depicted in fig. 1.4 with the example of surgical knot-tying. The left part of fig. 1.4 shows a situated dialog between a teacher and a trainee. The first step is giving an **abstract description** of the task, which is going to be taught. This step is not necessarily situated, since it is independent from the actual working environment. It can better be seen as a conceptual explanation, e.g. like one that is given via telephone. Consequently, the trainee will not be able to perform a skill exclusively based on this description. The description rather serves as a pattern to draw attention to certain features and to distinguish, which parts of the succeeding demonstrations will be important and which can be neglected. This is an important step towards learning based on a small number (best: one!) of user demonstrations. After giving this abstract description, **the teacher concretizes the skill, by presenting instances** within the application environment. Those will be used by the trainee to build up an execution plan from the abstract description. This methodology is a specialty of situated learning and is called **scaffolding**, which traces back to the work of L. Vygotsky in the 1930s [211]. The expression scaffolding itself has first been coined by Wood et al. [221] and refers to the teacher giving assistance on certain aspects of a task, which are beyond the capability of the trainee. In our architecture scaffolding is realized by the structuring of complex tasks by means of abstract patterns. This information cannot be derived by the system exclusively based on user demonstrations, but has to be communicated before learning the skill can succeed.

scaffolding

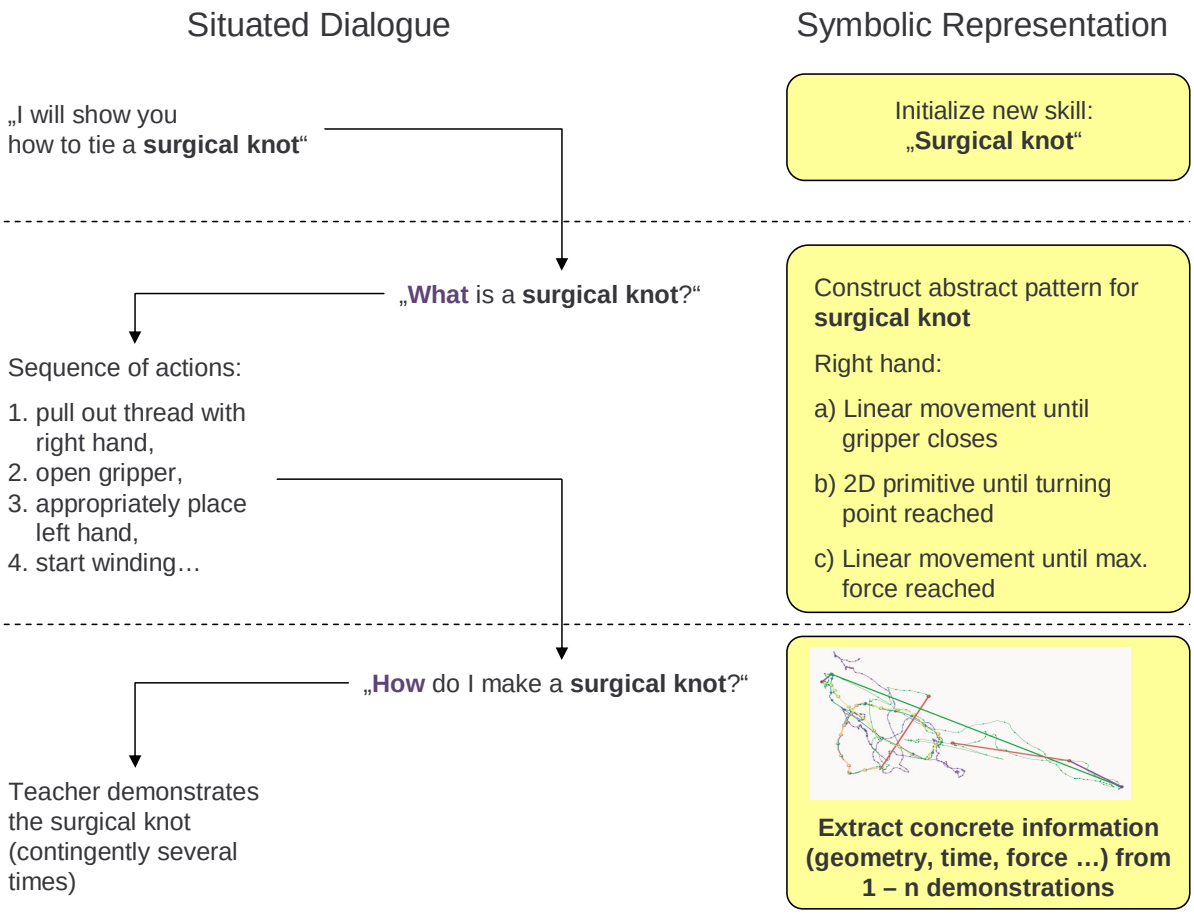


Figure 1.4: Human-machine interface for skill transfer

Since speech recognition and semantics is not of our concerns, we designed a user interface based on a symbolic form of the dialog just mentioned (right part of fig. 1.4). The **symbolic representation** of a skill is defined by means of certain events and primitives. I.e. events are certain changes in conditions of the environment, like closing of a gripper, inflection points of a trajectory, or the emergence of forces. Abstract primitives can be seen as transitions between these events. They are realized for instance as linear movements, 2D primitives, or force controlled transactions. In order to alleviate the **assembly of abstract skills**, we provide a drag and drop user interface as depicted in fig. 1.5. It enables an easy arrangement of the linear succession of primitives and events. Each track of the arrangement corresponds to one manipulator (instrument) of the system. For example, the

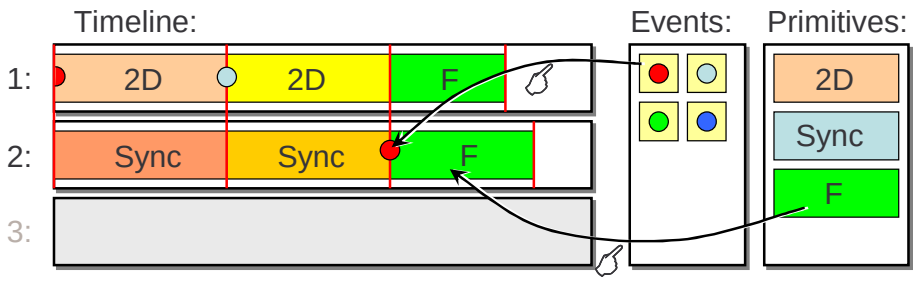


Figure 1.5: User interface for abstract skill assembly

arrangement for instrument no. 1 in fig. 1.5 reads as follows: starting with a “gripper state changed” event (red circle) we expect a 2D primitive until an inflection point of the trajectory occurs (gray event). After that, regarding all demonstrations of knot-tying tasks, another 2D primitive will follow until a synced event occurs. Synced events correspond to actual events in other tracks (e.g. in the current example: in track no. 1 the mentioned synced event corresponds to the “gripper state changed” event in track no. 2). Synchronization not only applies to events, but is also an important method regarding whole primitives. As depicted in fig. 1.5, the second track starts with primitives, which are synchronized with the 2D movements of track no. 1. This is an abstract description of the winding part of knot-tying: while the dominant hand (track no. 1) is winding the thread about the subordinate hand, the latter has to perform synchronized movements in order to coil up the emerging loop.

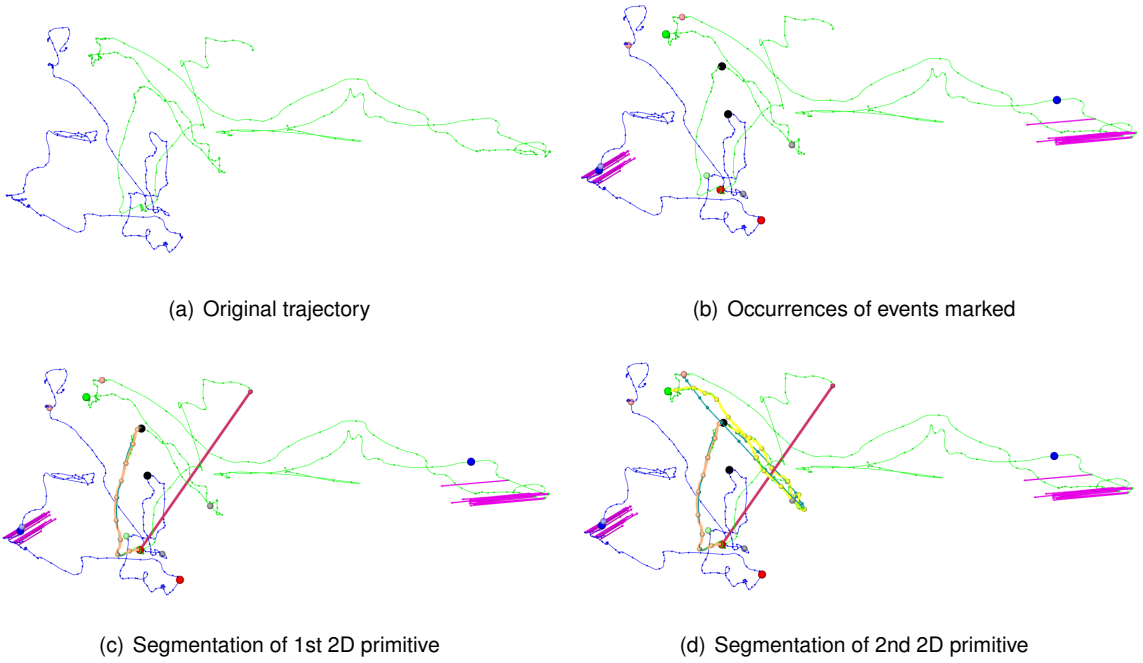
An abstract pattern (like the one described above) is applied to a specific demonstration by performing the following steps:

1. Search the trajectories for occurrences of all possible events. This step is illustrated in fig. 5.20(b), where all events are marked by spheres, colored depending on to the type of event.
2. Copy all events of a certain trajectory to all other trajectories constituting synced events. For example in fig. 5.20(b), “gripper state changed” events are depicted in dark red, while synced versions of this event type are depicted in light red.
3. After marking all events, the trajectories can be searched from the beginning for occurrences of abstract primitive patterns. Fig. 5.22 shows a snapshot after the second 2D primitive of track no. 1 is segmented.
4. Finally, all detected primitives are stored in a database for later application. For efficiency reasons, most of the parameters are stored by pointers to the original trajectory.

Apart from segmentation, this algorithm also performs a plausibility check on different levels, starting at event detection: if certain events, occurring within an abstract description, cannot be found within the specific trajectory, the latter is rejected for not being a valid demonstration of the abstract skill. The same applies to primitive segmentation (step 3): if the concrete instance of a primitive does not correspond to its abstract plan (e.g. data within a 2D primitive does not fit well to a plane), the whole demonstration will be rejected, too. Once a demonstration is accepted and segmented into primitives, the skill can be performed at any place in the environment. For this part of the algorithm, it is very important to work reliably after just a single demonstration. Otherwise, the usability of the system and its acceptance by end users will be diminished. Once the user demonstration is accepted, the usability of the system is still limited to an identical replay of the latter. In order to enable the adaption of a skill to unknown environments, we have to provide a knowledge base, which serves as a basis for skill application. Regarding our system, the knowledge base provides answers to the following questions:

- Where is the end of the thread located in the environment?
- How should the subordinate gripper be moved in order to form a loop?
- What are the essential parts of a 2D primitive and how can it be adapted to avoid obstacles?

The first question is answered by a module, which is concerned with thread recognition and scene reconstruction [26]. A detailed explanation of the underlying algorithms can be found in [163]. The



(a) Original trajectory

(b) Occurrences of events marked

(c) Segmentation of 1st 2D primitive

(d) Segmentation of 2nd 2D primitive

Figure 1.6: Illustration of subsequent steps of the segmentation algorithm

coordinates of the thread's end are used as an endpoint of the abstract primitive "linear movement towards visual cue". Every time this primitive is instantiated, the corresponding endpoint has to be calculated by the module by means of the images from the endoscopic camera.

The second question, regarding the synchronization of subordinate and dominant gripper, is answered by a description of their functional dependency. The easiest way to achieve this would be the determination of a simple translation between both grippers. Again, this assumption would result in a replay of the acquired trajectory and would work only if both grippers were moved homogeneously. In practice, trajectories exhibit much more complex dependencies between both manipulators: for example in the loop making part of surgical knot-tying, the movements of the subordinate gripper are scaled and flipped copies of the trajectory of the dominant gripper. Therefore, we assume at least an affine transformation as a model for this dependency, allowing for arbitrary rotations and translations, as well as shearing and scaling. Besides synchronization, affine transformations also overcome errors due to calibration. All calibration errors of the manipulators can be expressed as affine transformations, as long as the angular stepping of the motors is correct and their bearings are not afflicted by some kind of unbalance. This approach is a trade-off between simplicity (i.e. allowing application after just one demonstration) and the necessary flexibility of the dependency function. Another approach we evaluated for this problem, was the application of recurrent neural networks. Although this method demands much more demonstration examples to produce reasonable results, it usually provides better approximations of nonlinear dependency functions. Therefore, we provide this method as an alternative in our knowledge base. Affine transformations are only used as dependency functions as long as there are not enough demonstrations. If the error produced by affine transformations exceeds the error of neural networks, the latter are put into charge to handle synchronization.

1.4 Complex Primitives: Vector Fields

The last question mentioned above deals with 2D movements, marking the most complex primitives in our approach, i.e. all 3D trajectories are decomposed into arrangements of transformed 2D primitives. The feasibility of this method can be easily accepted: every trajectory can be discretized even by 1D lines. On the other hand, primitives should remain as complex as possible in order to preserve task-dependent structures. It can be observed that important sections of human precision movements can be reduced to 2D primitives, e.g. making surgical cuts, playing guitar or even walking stairs. This observation is supported by resent research on neural activity [52]. Certain cells in the human cortex take care of a proper transformation of 2D movement patterns (primitives) in order to apply them to tasks in 3D space [53]. Regarding our approach, the local transformation of a primitive is extracted from the demonstrations, but we still have to provide a method to generate an adequate trajectory for its application in new environments. In order to solve this issue for 2D movements, we employ the following analogy: Imagine a person stirring a fluid, e.g. a cup of tea. Two important observations can be made. The first is, 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 its motion after stirring has stopped. While the former phenomenon implements some sort of generalization, the latter realizes a memory. How can these observations be exploited to implement the basic principles of learning by demonstration? The answer is obvious: we model the environment, where certain 2D primitives are executed, as a fluid [20].

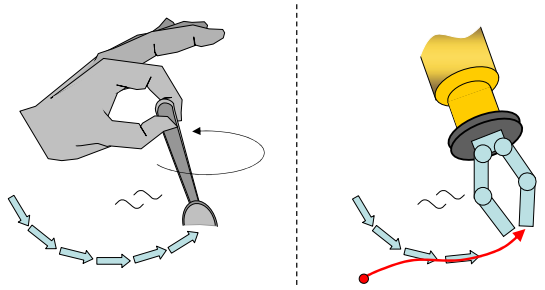


Figure 1.7: Illustration of the basic principle

generalization by fluid dynamics

An important prerequisite is that all demonstrations of the same task are congruent and can be overlain (corresponding solutions will be discussed below). Given this assumption, the 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. Needless to mention that the initial example of stirring a cup of tea is too simple to cover all these requirements. While this example, demonstrating how to make a loop with a spoon in a cup of tea, will end up in a stationary, vanishing motion of the tea, we will need a more complex model to cover more general demonstration challenges. Since fluid dynamics is a field of extensive research with an abundance of different methods, we will discuss the selection of an appropriate model in an own section of this work. For now, we only want to mention that a variety of dynamic, non-stationary models is available, where 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 get a working solution (fluid in motion) even after showing only one single demonstration of the task (one-shot learning). Nevertheless, it is still advising to have additional examples in order to make the solution more general and to eliminate outliers. After showing the demonstration, we can save the decisive parameters of the model in a very compact form. If we want to instantiate the learned task in a new environment, we simply transform the previously acquired model to the desired position and the fluid will adapt itself to this new situation. The intended starting point within the flow (i.e.

the initial position of the robot's end effector) is not that important: From each point within a certain neighborhood it will be automatically drawn into the right direction by the simulated current of the fluid. In addition, the trajectory (the flow of the fluid) will bypass obstacles and smooth out discontinuities. It is also possible to include "inter-robot" collision detection into this model. This is particularly important in robot assisted surgery, since the manipulators are mounted at close quarters in order to enable bi-manual and assisted operations. Collisions between different robots (outside the actual working space) can be considered by inserting virtual obstacle cells into the fluid. In order to determine virtual cells, collision detection has to be performed for every position inside the field of application (cf. fig. 1.8, right). This field corresponds to a small area in the working space of the minimally invasive instrument (e.g in fig. 1.8 the right picture is a magnification of a 2x2 cm area near the tip of the manipulator in the front). The left hand side of fig. 1.8 shows a collision of the robots, detected by an oriented-bounding-box (OBB) check. Since the area mentioned above has to be sampled in an adequate resolution, this OBB check has to be performed for at least 250 x 250 positions, which renders this process computationally expensive. Therefore, we have implemented this algorithm to be performed on the graphics hardware (GPU) of our control system.

Putting it all together allows us to perform user-demonstrated skills in previously unknown environments after just one demonstration. The proposed framework was successfully applied to a realistic scenario within a safety critical environment: automatic knot-tying in a system for minimally invasive surgery.

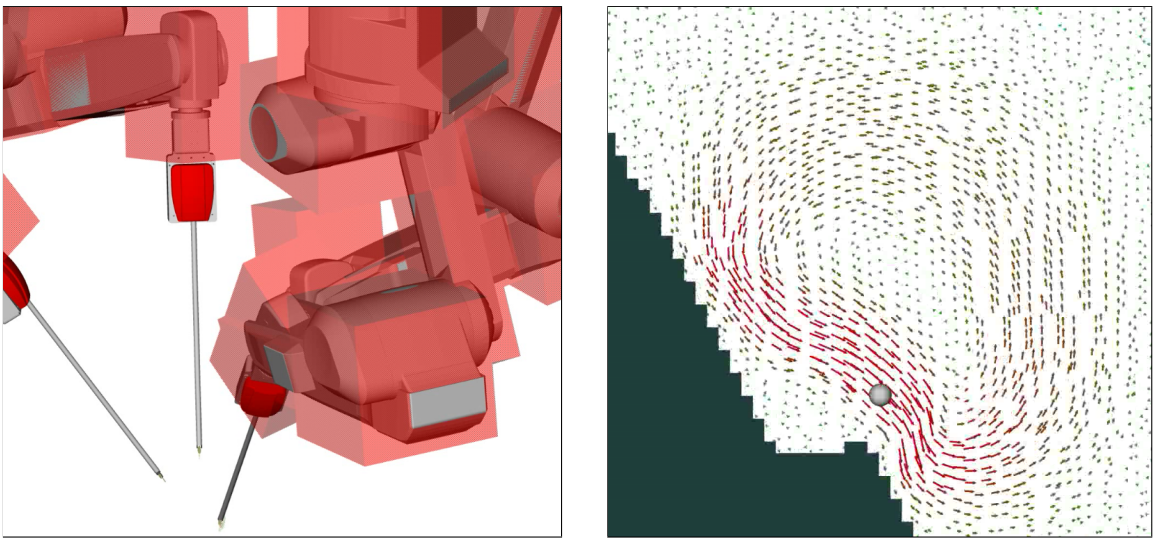


Figure 1.8: GPU-accelerated collision detection

Chapter 2

Hardware

In the last decades of the past century, minimally invasive surgery was introduced by surgeons of different disciplines. Particularly heart surgery profits from this technique, since access to the operating field is usually constricted by the ribcage. The system presented below is not only capable of minimally invasive interventions, but also a specialty of it, endoscopic surgery, can be performed. The difference is in the size of the access ports. While minimally invasive cuts usually do not dissect bones, they are still large enough to supply the field of operation with working materials, or even to observe the action point. In contrast, endoscopic cuts are limited to a necessary size for inserting an instrument. Therefore, operations of this kind are much harder to perform, but offer great advantages over minimally invasive or conventional interventions. Minimally invasive and endoscopic cardiac surgery not only minimizes the collateral surgical trauma, but it also results in quicker recovery of patients. The length of the hospital stay and the infection rate can be reduced. Therefore, patients massively profit from this treatment option. On the other hand, surgeons have to cope with increasingly complex working conditions. Since MIS is performed through a small port or "key-hole" in the patient's chest (cf. chapter 3), surgeons must learn to operate with unfamiliar and often awkward surgical instruments [62]. Hence, the techniques of minimally invasive surgery have been applied infrequently, particularly in the field of heart surgery. An important step to push this technology was the introduction of telemanipulators, which have been especially designed to overcome the fulcrum effect of minimally invasive instruments. The surgeon no longer operates an instrument itself, but it is driven by a special device with a Cartesian user interface, which surgeons can handle as usual, i.e. like instruments for open surgery. Commercial examples of such devices are the *daVinci*- [97] and *ZEUS*-system (the latter has been discontinued). They offer as much freedom of movement as the hand of the surgeon in conventional open surgery, thus, providing six degrees of freedom instead of four like conventional endoscopic instruments. In addition, they assist the surgeon with motion scaling and provide a stereo vision interface at the input console. Another important feature of teleoperated surgery is the avoidance of physiological tremor, which is inherent to every movement of the human hand [55, 106]. Surgeons can now operate with such surgical mechatronic assistants in a comfortable, dexterous and intuitive manner [82, 83, 84]. Despite the obvious potential advantages of robot assisted MIS, most researchers and surgeons in this area agree that the lack of haptic feedback is the most important drawback of currently available systems [153]. The inability of the operator to sense the applied forces causes increased tissue trauma and frequent suture material damage. The systems are telemanipulators with no direct position control (the control loop is implicitly closed by "visual servoing" of the surgeon). Both features are important in order to move the surgeon up in the control hierarchy, i.e. to implement "shared control" or "partial

autonomy". In order to overcome these deficiencies, two crucial issues have to be solved:

1. Inclusion of force measurement at the end effectors and corresponding force feedback for the surgeon.
2. Implementation of full Cartesian control over the end effectors by means of an intuitive input console.

The latter is indispensable for calculating exact directions of forces in a known coordinate frame. Therefore, regarding the hardware of the system, one of our main research interests was the construction and evaluation of force measurement and feedback in realistic scenarios of robotic surgery.

2.1 Previous Work

commercial systems

Historically, there have been two main competitors on the market for minimally invasive surgical robots, *Intuitive Surgical Inc.* [111] and *Computer Motion Inc*. While the former is still successfully marketing the *daVinci* telemanipulator mentioned above, the latter was merged into *Intuitive* in 2003 [147]. Therefore, research on the corresponding products of *Computer Motion*, the *ZEUS* surgical system [180] and the *AESOP* endoscopic arm [193], has been discontinued. During this time the "Robdoc Crisis" occurred, which was a major drawback for robotic surgery. *Robodoc* [98] is a robotic assistant to drill holes for hip replacements. While it has not got an approval of the US Food and Drug Administration (FDA) yet, thousands of operations have been carried out in Europe and the Far East. Follow-up studies have shown that this robotic procedure occasionally harmed some nerves near the hip bone, which caused painful side effects. Therefore, robotic surgery as a whole has aroused suspicion, especially by patients. Meanwhile, the errors which led to these side effects have been identified and eliminated, and there are plans for new systems featuring improved navigation [41]. Therefore, robotic surgery was rehabilitated and new demand for robotic procedures has raised again. Currently, two new commercial products appeared on the market: The first was the the *Laprotek system* [88], which provides slave manipulators similar to *ZEUS*. Its master console provides comfortable control of the instruments and a video display of the operating field. According to the vendor, it is also capable of force feedback, which is based on measured motor torques, and therefore, probably not very accurate. Another currently developed system is based on lightweight robotic arms of the German Aerospace Center (Deutsches Zentrum für Luft- und Raumfahrt, DLR). While their robotic technology is currently evaluated in conventional medical applications [175, 44], they are planning to provide a commercial robotic system for minimally invasive surgery named *KineMedic* [129]. Apart from these general purpose MIS robots, there are dedicated commercial systems available, like the *Pathfinder* robot from *Prosurgics Inc.* [154] for neurosurgery, or the *CASPAR*-system [178] for orthopedic surgery. The latter has been a German competitor of the *RoboDoc* system, but suffered from the same deficiencies.

Robotic surgery has also drawn the attention of many researchers. At the University of California, Berkeley, a robotic system was developed, which has already been used to perform certain surgical tasks like suturing and knot-tying [70]. The Korean Advanced Institute of Science and Technology has developed a micro-telerobot system that provides force feedback and is used for surgical training [134]. In Germany, at the University of Karlsruhe, a prototype system for robotic surgery was assembled and evaluated (*ARTEMIS*) [209]. They have developed a master station and slave robots and carried out animal experiments, but meanwhile the project has been discontinued. Nakamura et al.

[164] have developed a 4DoF slave robot to evaluate heartbeat synchronization. As input master they have used a standard PHANToM device, but they did not exploit its force feedback capabilities. Muñoz et al. [159] have proposed a robotic system for transurethral prostate resection. Either an endoscope or a minimally invasive instrument can be flanged on a *Stäubli* industrial robot. Mitsuishi et al. [152] have developed a surgical system for far distance teleoperation. It consists of a custom-made master console and a slave manipulator, which provides 7DoF input for each hand (including closing of the gripper) and mechanically enforced trocar kinematics. They employed their system to suture a blood vessel, while master and slave were located 700 km away from each other. Meanwhile, they have developed a new version of the system comprising force feedback, and used it for surgical teleoperation (cholecystectomy of a pig) [151]. In a recent experiment they have increased the distance between master and slave to over 3750km (Fukuoka–Bangkok) [56]. While similar projects rely on dedicated fiber optic lines [143], they have based their approach exclusively on conventional network infrastructure by means of providing a sophisticated software for data encoding. There are also some unusual solutions, like the hyper finger of Ikuta et al. [109], a snake-like hyper-redundant robot which is capable of operations inside body cavities, like the frontal sinus or the interperitoneum. Besides the development of new prototypes, some researchers have also been focused on the enhancement of existing systems. The employment of force feedback devices in connection with teleoperated robots was proposed after preliminary studies at the Stanford University [207]. Adhami et al. have augmented the *daVinci*^{TM} system with an IR marker based positioning system, which assists the OR team in proper placement of the arms [51]. Another example is the enhancement of the *ZEUS*^{TM} system with a master console that consists of two PHANToM haptic devices [135], but they did not use the PHANToMs to feed back instrument forces. Anyway, force feedback in minimally invasive surgery is the topic of many publications in the field of haptics research. While some projects are restricted to the application of force feedback in virtual environments for education and simulation (e.g. Cotin et al. [74], [131] or Baumann et al. [59]), other researchers deal with the incorporation of force measurement in robotic end effectors [48], which is more relevant for our work. Though haptics is only a subsidiary human sense (according to Mair et al. [140] it contributes only 4% to the overall impression of telepresence), it has been proven by our own research that inclusion of haptics can significantly improve telepresent work [8]. In recent years, a variety of systems have been developed which incorporate haptic feedback into surgical systems. Like us, Kennedy et al. [120] have used PHANToM devices for haptic feedback in surgical procedures, but they did not measure contact forces at the instruments. Instead, they tried to calculate forces from tissue deformation which was captured with a stereo camera system. So far, they conducted only mockup experiments, but have not tested their algorithms in realistic environments like minimally invasive heart surgery. In an experiment of blunt dissection, Wagner et al. [213] have shown that inclusion of force feedback significantly reduces forces exerted on tissue. Tavakoli et al. [203] have developed own master and slave devices for force feedback and measurement, respectively. They found the profit from force measurement can be increased, if, in addition to haptic feedback, data is presented visually. In their opinion, this is due to the fact that the sensitivity of the human hand is quite limited (to a resolution of 0.5 Newton according to [194]). Another haptic device for telesurgery was proposed by Gosselin et al. [86]. It can be used for 6DoF input (plus closing of the gripper) and 6DoF force feedback in telesurgery. If feedback of the closing force of the gripper is required, a device similar to that proposed by Okamura et al. [174] could be employed. We have implemented force measurement by augmenting the instruments with strain gauge sensors. Recently, this has also been done by Pitakwatchara et al. [179]. They have performed characteristic motions of minimally invasive surgery (like grasping and pulling) and evaluated the forces. Houston et al. [38] have

research systems

augmented microgrippers with strain gauge sensors in order to measure grip forces. Force feedback has also been included into ultrasound guided surgical procedures by Wagner et al. [212] in order to provide an additional modality for improved navigation.

2.2 EndoPAR: *Kuka*-based System

As mentioned above, our main goals are the evaluation of force feedback in minimally invasive surgery and the implementation of Cartesian control in order to enable automation. Therefore, we decided to construct our own research platform, which meets all these requirements. After some pilot surveys on the *daVinci* system at the German Heart Center Munich, we started working on our own system in 2003. It was based on two *Kuka KR6/2* robots which are mounted on the footing of our lab. Since these robots are normally employed for industrial welding tasks with strict speed guidelines, their dynamic abilities (jerk-free acceleration and maximum speed) are sufficient to perform surgical tasks. In minimally invasive surgery, the so-called port or trocar kinematics (see below) will cause high accelerations due to leverage effects (e.g. in a typical setting with MIS instruments inserted 10 cm into the patient, each movement of the instrument's tip by 1 cm will cause a movement of the robot's flange of about 5 cm). We have equipped each robot with an original MIS instrument as it is deployed with the *daVinci* system. In this first version of our system, instruments have been flanged to the robot's end effector by rigid adapters. These adapters have been equipped with small servo motors to drive the grippers of the MIS instruments. All instruments are augmented with strain gauge sensors for force measurement. The forces are transformed to an appropriate coordinate system and fed back to the user via two *PHANToM* haptic devices [144]. These devices are also used for 6DoF user input. Another indispensable feature of such a system is 3D vision. Otherwise, it is impossible to perform delicate manipulations like knot-tying. Therefore, we have equipped a third robot (*Stäubli RX 90*) with a stereoscopic MIS camera. The acquired visual information was presented to the user by a head mounted display, which was fixedly attached to the user's console. We have named this first version of the system EndoPAR for "Endoscopic Partial-Autonomous Robot". Although the first results acquired with this system have been promising, the experiments have also yielded some new issues (for evaluation of the system and derivation of the results confer to chapter 4). One major drawback was the employed *Kuka* robots being quite bulky, and therefore, blocking access to the operating table. In addition, the low-level interface of the robots has often caused problems, even to the point of hardware damages (destruction of MIS instruments). Another issue was the 3D vision system with its head mounted display: The quality of the employed HMD was very poor, and therefore, has complicated navigation in 3D space.

2.3 ARAMIS: *Mitsubishi*-based System

Given the hitches of the EndoPAR system, we have decided to develop an improved version of the system at the beginning of 2005. This time we took smaller robots (*Mitsubishi MELFA 6SL*). In order to perform more complex operations, which require an assistant arm, we have integrated four robots of the same type. They can either be equipped with different MIS instruments (e.g. gripper, knife, scissors), or with an endoscopic stereo camera. Based on the experience with the EndoPAR system, the robots are mounted on the ceiling this time, or strictly speaking, on a gantry built of aluminum girders (cf. fig. 2.2). In addition, all cables, which are needed to operate the system, are passed along the gir-

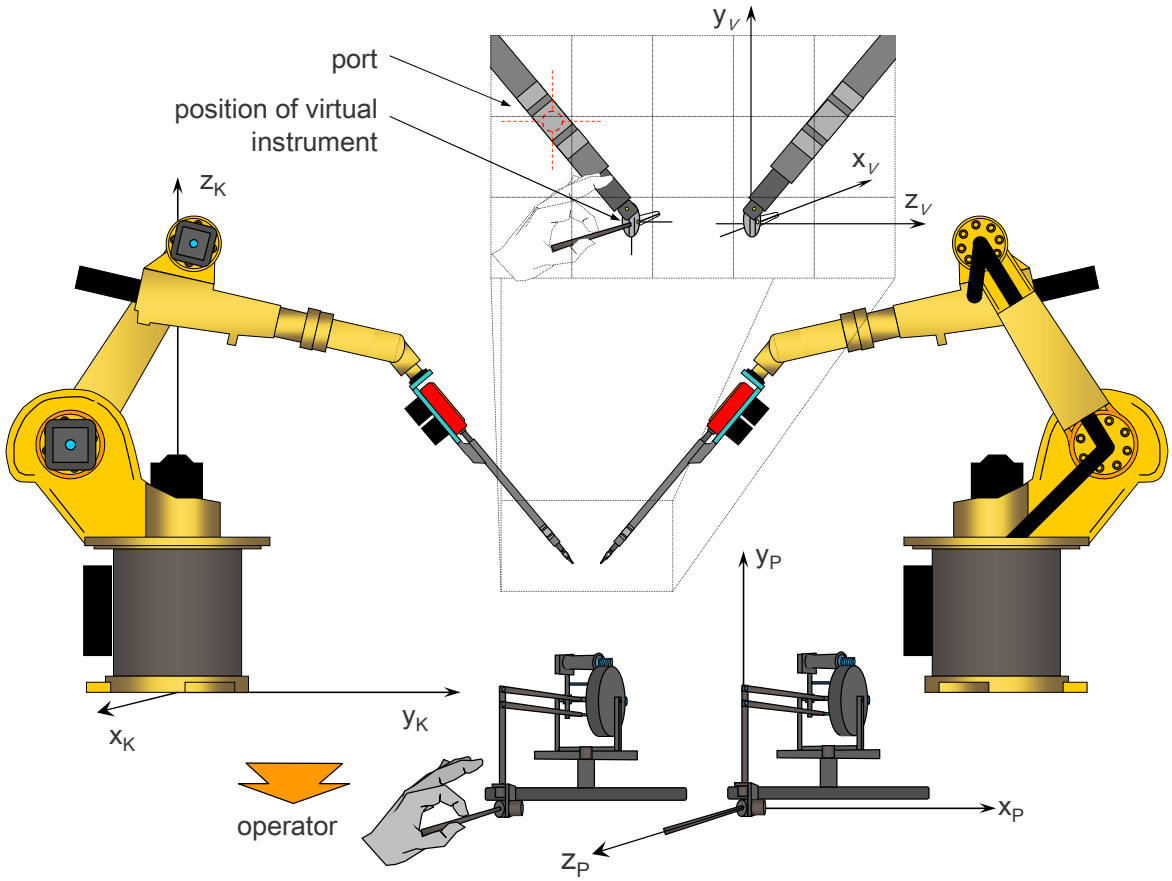


Figure 2.1: The first version of the research platform: EndoPAR

ders, i.e. no objects obstruct the access to the operating table. The controller boxes of the *Mitsubishi* robots are much smaller and more silent than the ones of the *Kuka* robots. The low-level interface connecting to the robot's controller is realized by means of standard network protocols (TCP, UDP) and works very reliable. This technique also enables a new level of telepresence. While the old system interface was based on serial connections to the robots and instruments, the new one is based on ethernet and CAN bus. Recommended cable length for a 19200 baud serial line is 15m, i.e. the master console has to be in the same room as the robotic slave system. The cable length of the new system is limited by the CAN-bus, specifying a maximum cable length of 50m. Therefore, it is possbible to site the master console outside the operating theater in order to open up possibilities for new applications (e.g. parts of an operation can be performed from an unsterile room for educational purposes or similar). In order to be prepared for new requirements, we also moved the system's software to a new quad-CPU machine. We have already made extensive use of its computing power for the numerical fluid simulation (see below). We have also assembled a new version of the master console, equipped with a better alternative of 3D vision. After several tests, we decided to use a 3D display based on orthogonally polarized stereo images. This gives a more perspicuous impression of the 3D environment. The description of the software, the algorithms and all experimental data, except the evaluation of force-feedback, refer now to this new system. If some parts of the soft- / hardware still refer to the old EndoPAR system, it will be explicitly stated.

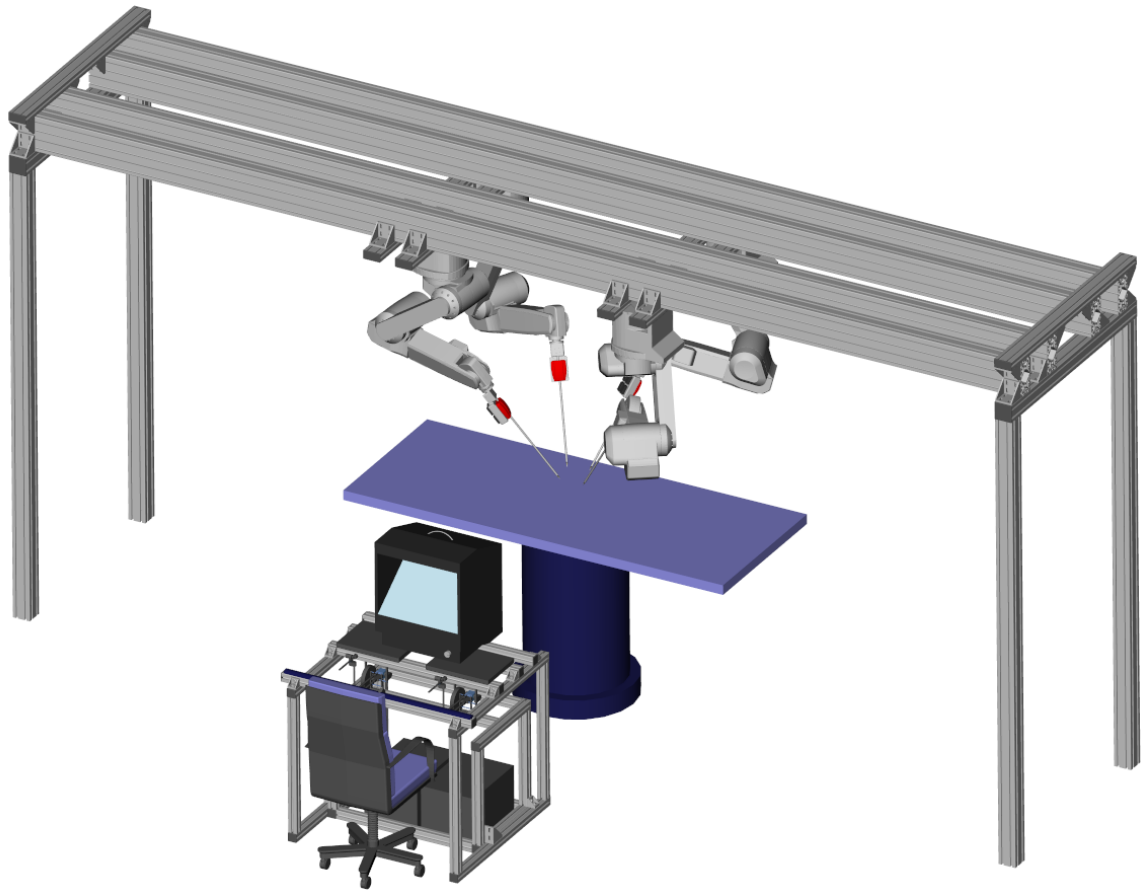


Figure 2.2: The improved version of the research platform: ARAMIS

2.4 Master Console

Like many other systems for telepresent and minimally invasive surgery, the structure of our system can be mainly divided in two major parts: a robotic slave system for direct manipulation on the subject and a master console for user interactions. After several tests of the ergonomics, we have chosen a setup depicted in fig. 2.3 as user interface. It 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* haptic displays. Their controller boxes are stored at the base of the frame. The *PHANToM*s themselves are mounted 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 *PHANToM*s. 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.

haptic feedback

2.5 *PHANToM* Force Feedback Devices

As mentioned above, one of our main research goals was the evaluation of the impact of haptic feedback on minimally invasive surgery. If we speak of haptic feedback in connection with this research project, we actually restrict ourselves to force feedback. Therefore, the terms haptic feedback and

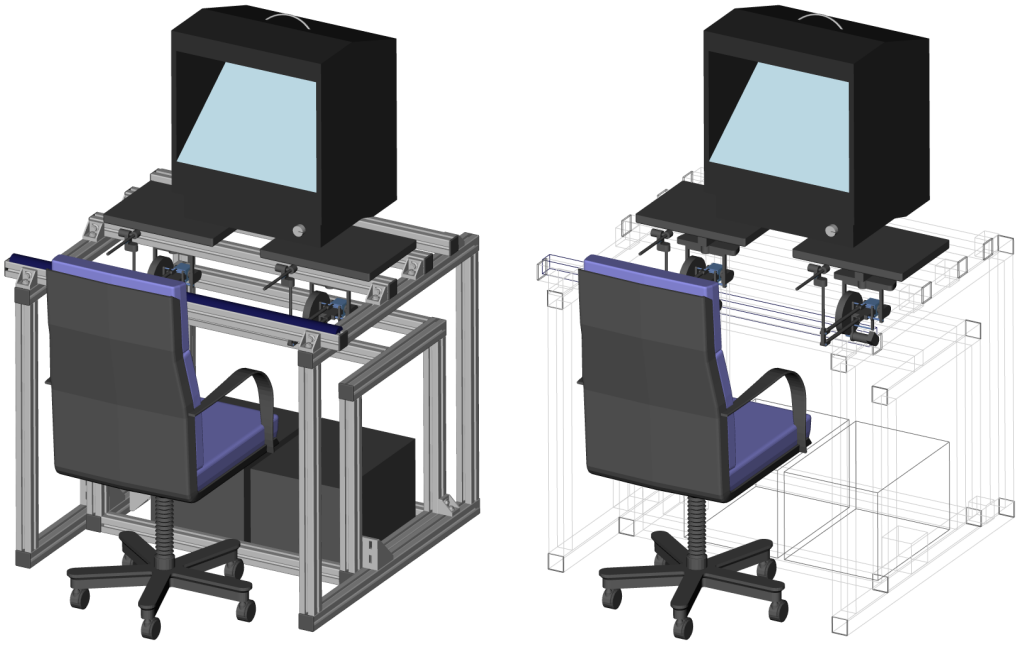


Figure 2.3: Master console with PHANToM^{TM}devices and 3D screen

force feedback are used interchangeable within this text. In general, haptic feedback also comprises tactile feedback, i.e. perception of certain properties of a surface (particularly its smoothness). This kind of haptic feedback will not be subject of our research. We do not want to measure and feedback the properties of surfaces, but the tensile and shearing stress of objects. Today, there are not many devices available providing realistic force feedback. Many of them are based on a hexapod layout constituting a parallel mechanism. Examples of this type of construction are the *Falcon* from *Novint Technologies Inc.* and the *Omega.X* from *Force Dimension Inc*. The advantage of the employed parallel mechanism is the generation of comparably high forces to be displayed at high fidelity. On the other hand, these devices suffer from a restricted working space regarding translational as well as rotational degrees of freedom. Another approach for 3DoF force feedback devices is based on serial mechanisms as they can be found in robots. Examples of the serial type are the *PHANToM* device from *Sensable Technologies Inc.* (which actually was one of the first devices on the market) and the *HapticMASTER* from *Moog FCS B.V*. These devices provide larger working spaces (in relation to the overall size of the construction), while the amplitude of force display is limited. Since we do not want to reflect high forces, but prefer an ample working space instead, we decided to acquire two *PHANToM* devices. The version we employ, provides a working space of approx. $40 \times 25 \times 20\ cm$ and maximum forces of 8.5 Newton. Forces are generated by three small servo motors assembled at the first three joints of the device. The remaining three joints constitute a swivel-bend-twist wrist, which provides an overall capability of 6DoF input. We have evaluated different configurations of the device (cf. fig. 2.4). The original one, depicted in the middle, exhibits some disadvantages. The superstructural parts complicate the assembly of the console (e.g. it is not possible to place a monitor on top of the device). In addition, the placement of the forearm restricts the kinematic abilities of the wrist of the device. In order to overcome the latter disadvantage, we have altered the position of the forearm as depicted in fig. 2.4 on the right hand side. Needless to mention that this did not resolve the first issue. Finally, we decided to mount the device upside-down. This leaves space on top of the device while the forearm is

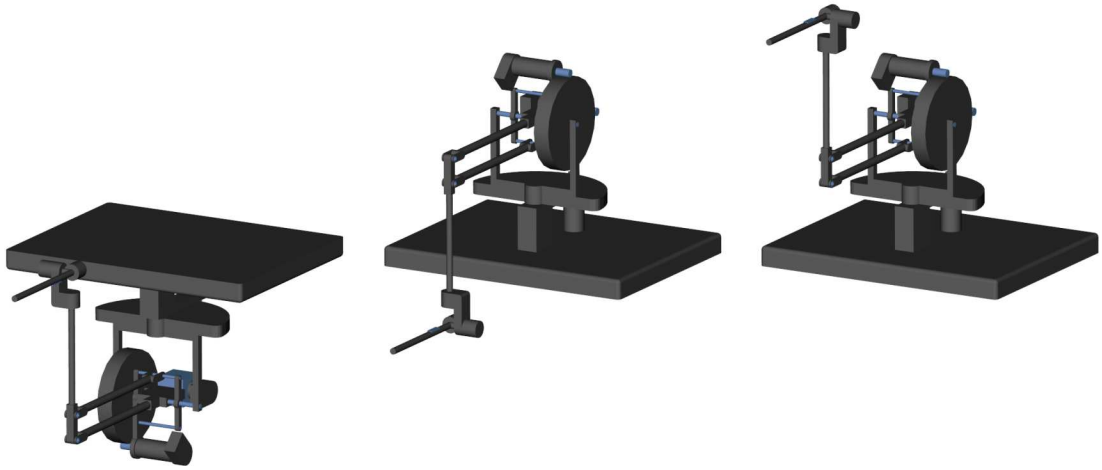


Figure 2.4: Different configurations of the PHANToM^{TM}device

in a convenient position. In addition, the danger of a gimbal lock of the wrist is still low. The only thing left is to flip the geometry of the inverse kinematics and the force generation (see below).

2.6 Instruments and Force Sensors

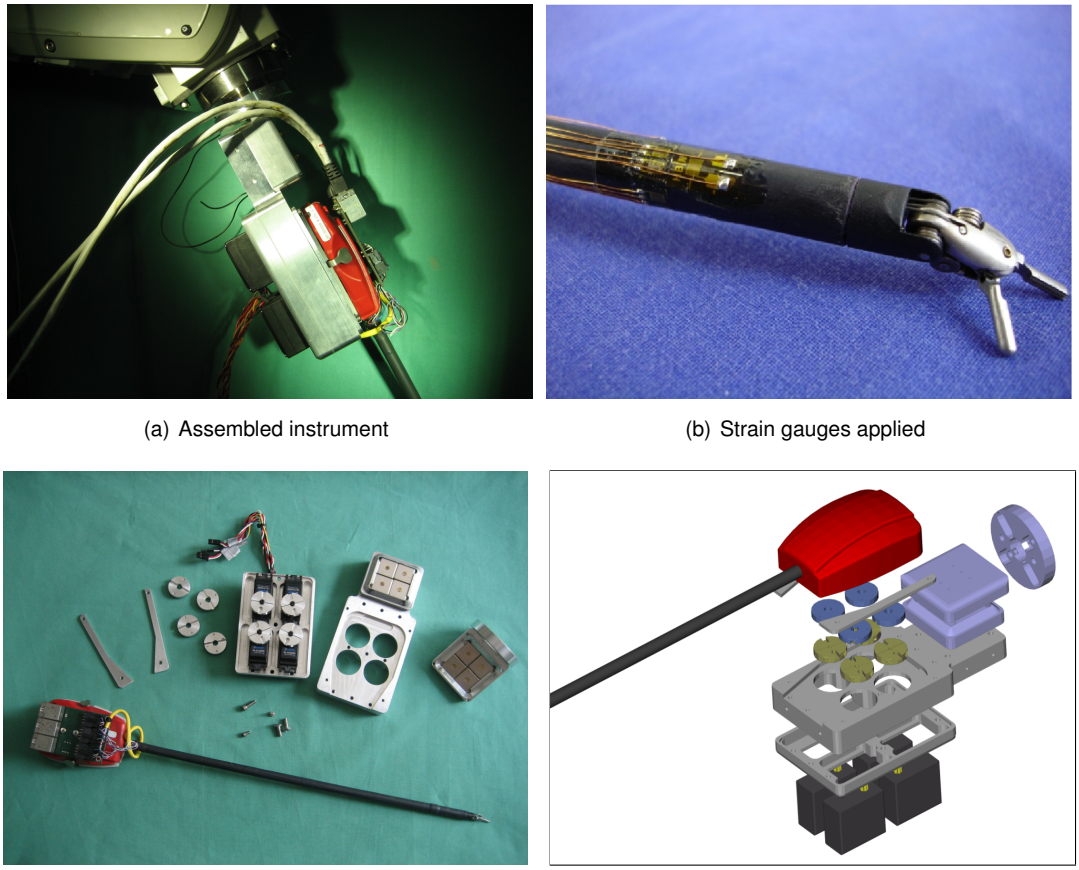
The most interesting feature of the employed *PHANToM* devices is their capability of displaying forces to the user. Forces are fed back by small servo motors incorporated in the device. They are used to steer the stylus pen toward a certain direction. This creates the sensation of occurring forces while the user is holding the pen at a certain posture. Our version of the *PHANToM* device can display forces in all translational directions, while no torque is fed back. In order to be able to display realistic forces during operations, we have equipped the instruments with force sensors.

Like other research groups [37, 45], we have used the *EndoWrist* instruments of the *daVinci* system. Since the shafts of these surgical instruments are made of carbon fiber, force sensors have to be very sensitive and reliable. Therefore, we decided to apply strain gauge sensors, which are employed in industrial force registration 2.5(b). Strain gauge sensors are usually made of silicon, which provides by far the best sensitivity toward elongations. Its piezoelectric resistance increases under dilation and decreases under compression, respectively. The resulting resistance R is linearly dependent on the axial stress ($\sigma = F/A$), the relative elongation ($\varepsilon = \Delta l/l$) of the resistor and its transversal contraction ($\Delta r = -\mu r\varepsilon$), where μ is the Poisson ratio, a material-dependent constant. For small, reversible elongations ($\varepsilon < 10^{-3}$) the following equation holds:

force measurement

$$\frac{\Delta R}{R} = \frac{\Delta l}{l} - \frac{2\Delta r}{r} + \frac{\Delta \sigma}{\sigma} = \left[1 + 2\mu + \frac{\Delta\sigma}{\sigma\varepsilon}\right]\varepsilon = K \cdot \varepsilon \tag{2.1}$$

Therefore, the resistance depends only on the elongation ε and a material-dependent constant K, which denominates the sensitivity of the material and accounts for a dimension of 100 for silicon. These strain gauges are very compact, cheap and with fast reactions. We applied the sensor gauges at the distal end of the instrument's shaft, i.e. near the gripper. This yields the inevitable disadvantage of sensor readings being flawed with tensions of the driving wires. This is particularly an issue of



(a) Assembled instrument

(b) Strain gauges applied

(c) Adapter parts with Oldham coupling

(d) Exploded view

Figure 2.5: Sensorized instrument, magnetic adapter and servos

measurements in axial direction when the gripper is opened and closed. Therefore, we have disabled this axis for our experiments and restricted the measurements to lateral distortions. Due to their layout as full bridges (see below), they are not disturbed by the closing state of the gripper. Anyway, since disturbing forces can still exceed an acceptable limit during extensive movements (e.g. knot-tying), we provided a switch to annihilate force levels in situations when advanced sensitivity is demanded. As mentioned above, the sensors have been directly attached to the instruments by a heat-resistant two-component adhesive and have been coated by shrinking tubes. Therefore, this technology is both waterproof and heat-resistant in order to survive medical autoclaving. Since we did not know the exact material parameters of the instruments, we calibrated the sensor system with standard weights. In order to make sensor readings more stable, we employ four sensor gauges interconnected as

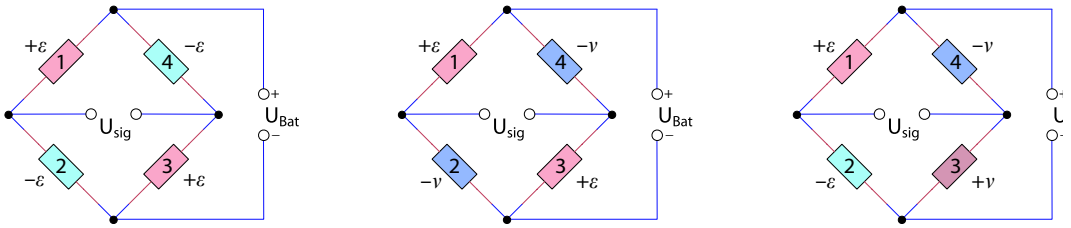


Figure 2.6: Possible layout of strain gauge sensors

Wheatstone full bridge for each translational direction. Usually, there are three possibilities to arrange sensor gauges as a full bridge. We have chosen the layout depicted in fig. 2.6 (left side) with all four gauges measuring elongations along the main axis. Sensor readings are independent of thermal fluctuations, since resulting changes of single sensors eliminate each other. In addition, the chosen layout offers the advantage of the resulting resistance still being linearly correlated to elongations. Other possibilities presume that two of the sensors are perpendicularly arranged to the axial direction in order to measure lateral strains. These solutions are either non-linear or not resistant to changes in temperature. For measurement of x- and y-forces, the bridge parts for dilation (1 and 3 in fig. 2.6) and compression (2 and 4 in fig. 2.6) are attached on opposite areas of the shaft (cf. fig. 2.7 left). Therefore, the forces measured in these directions are independent from axial elongations as they may occur during opening and closing the gripper (induced by the steel wires inside the shaft).

For digitizing the sensor readings we have used *GSV3CAN* measurement amplifiers from *ME-Messsysteme GmbH*. The blank with the main components is very compact (cf. fig. 2.7: blank size 30 x 55 mm) and can be attached directly onto the instruments. It also provides a CAN-bus interface capable of transmission of 1220 16bit values per second. This is necessary, since haptic interaction with stiff contacts requires a sampling rate of at least 1000 Hz. For internal processing, the amplifier provides a sampling rate of 10 kHz. Therefore, it calculates the medium of eight sensor readings prior to transmission. Since we know the position and orientation of the grippers, we can transform the

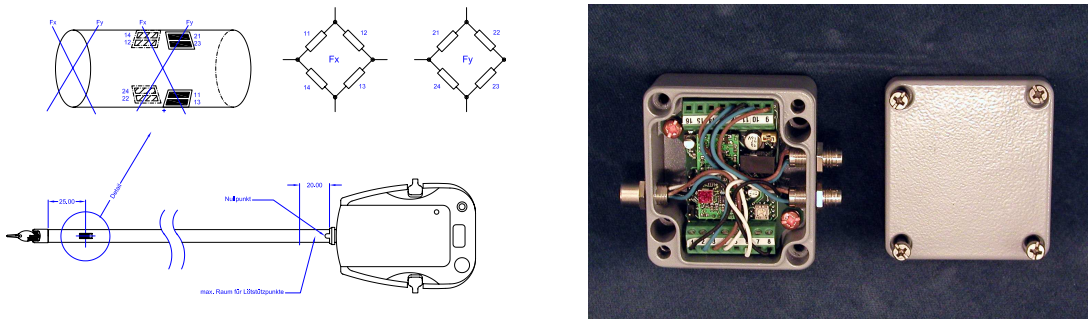


Figure 2.7: Arrangement of the sensor gauges (left) Measurement amplifier (right)

forces to the coordinate frame of the *PHANToM* devices. In addition, forces can be arbitrarily amplified by our control software and recorded to a data base for subsequent evaluation. Figs. 2.8 show some experiments we accomplished before we started the actual evaluation. The left figure shows the force progression during breaking of suture material, and in the right one a winding task is depicted. This shows that strain-stress forces induced by the wires are minimal during thread breakage, since the orientation of the instrument is not changed. They are also acceptable in procedures that actually include minor changes in the gripper positions, like winding. The corresponding force plot shows that the characteristic force progression during winding (a sinusoidal function) dominates disturbing effects.

2.7 Vision Subsystem

Due to extremely high-precision requirements in modern minimally invasive surgery (arteries in heart surgery have a diameter between 1 and 2 mm), it is imperative to provide an accurate 3D view. Most of the other systems mentioned above come with magnifying three-dimensional endoscopes and ap-

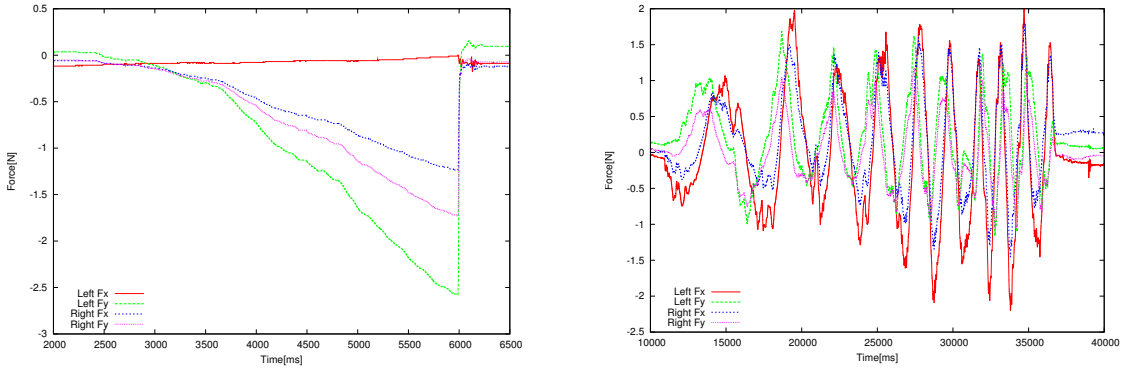


Figure 2.8: Force progression during thread breaking and winding [*plots published in [21], data recorded by I. Nagy*]

propriate display devices. This is also an essential prerequisite for high accuracy machine vision. The **ARAMIS** system is equipped with a 3D endoscope providing two separate optical (fiberglass) channels and two synchronized CCD cameras. A magnetic coupling mechanism (fig. 2.9) allows quick mounting and dismounting without losing the calibration against the rest of the system. We have precisely determined the intrinsic and extrinsic parameters of the camera system in order to achieve the necessary calibration for scene reconstruction [163]. Like the instruments, the endoscopic camera can be moved by means of trocar kinematics and can either be actively controlled by the operator or automatically be guided by the system (following the region of interest, i.e. where the instruments are placed).

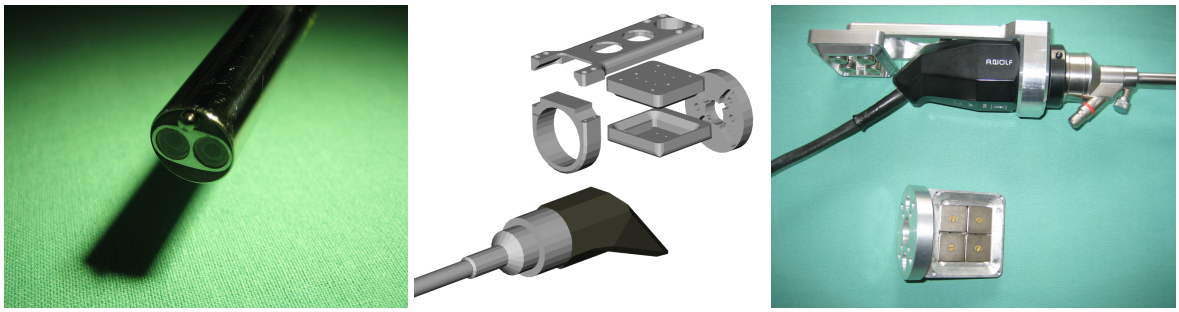


Figure 2.9: Endoscopic stereo camera with customized magnetic adapter

Images taken by the stereo camera system can be displayed via different options: The first is a head mounted display (HMD), which was part of the first version of our input console. Another possibility is to alternately display left and right images on a CRT-screen. In this case, the operator has to wear shutter glasses, which are triggered by the output on the screen. A third option is the projection of the acquired pictures on a silver screen with two video projectors. The projectors have to be equipped with polarizing filters, which are orthogonally arranged. Observers have to wear glasses with an appropriate polarization. Unfortunately, picture quality is the weak point of all these options. Therefore, we have developed a new prototype based on optical systems, originally designed for three dimensional interpretation of air photographs. A pair of images is displayed on a high-resolution CRT-screen. Each image is redirected to the corresponding eye of the beholder by an eyepiece containing superfinished mirrors. This system provided by far the best picture quality, but tests have shown that it is more convenient to work with non-ocular systems, because the operator can observe the hands,

and therefore, gets a better hand-eye coordination. The currently employed system is a compromise between picture quality and intuitive handling. It is based on two orthogonally polarized flat screens (cf. fig. 2.3). If observed with correspondingly polarized glasses, a 3D image emerges, which provides good stereopsis.

2.8 Controller

The controller of our system is a *dual-Opteron* workstation equipped with two *dual-core Opteron* CPUs. It runs on 64bit Linux 2.6. The board is populated with 8 GB of main memory and PCI as well as PCI express slots. The legacy PCI slots are necessary because of the PHANToM^{TM} devices, which interface via standard PCI cards. In addition, there has to be a clearance of at least one slot between the two cards, due to the geometry of the connectors. The system is equipped with a PCI express graphics card and hot plug SATA hard disks. Due to performance reasons, the measurement of forces and the controller of the instruments' servos is sourced out on a second computer. The latter is equipped with a PCI card for real-time CAN-Bus connections.

Chapter 3

Software

Control software of our robotic system is organized in four top-down layers: user interface, inverse kinematics, collision detection and HAL (hardware abstraction layer). On top of the hierarchy, the user decides how to interact with the system in order to perform a certain task. Interactions can be carried out either online or offline. The **online interface** provides a possibility to control the robots in real-time with the PHANToM devices. This comprises filtering human tremor, scaling of movements and velocity control. In addition, the user can switch between different manipulators. All data generated with the online interface can be forwarded to the offline interface for visualization, recording and archiving. In addition, trajectories can be recorded and forwarded to the **machine-learning module** of our software.

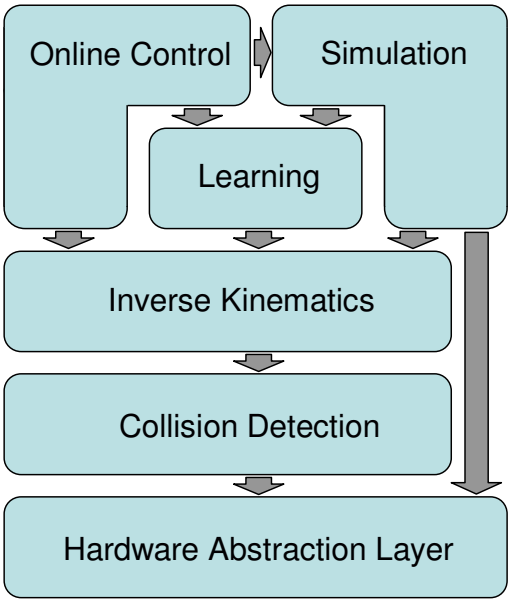


Figure 3.1: Control scheme

The main part of the offline interface is a **simulation environment**, which provides a virtual reality model of the real system. Using this interface, complex scenarios can be assembled and tested: i.e. trajectories can be previewed within the simulation environment before they are actually executed. This function is supported by a novel planning tool based on keyframing (as it is implemented in animation software). Both online and offline interfaces can optionally incorporate functionality of the learning module, which implements human-machine skill transfer. This opens up the possibility for “learning by demonstration”, which is documented in chapter 5. Afterwards, all data intended for transmission to the controller of the robot is transformed into joint space by the **inverse kinematics** module. Besides this basic functionality, this module also provides so-called trocar kinematics, which is required for minimally invasive procedures. All movements generated by the kinematics module can optionally be checked by an underlying **collision detection**. An alternative implementation of the corresponding algorithms is ported to the GPU of the graphics card. This option is employed to accelerate the application of primitives in the skill transfer module (cf. chapter 5).

Having passed collision detection, the transformed data can now be sent to the **hardware abstraction layer**. The HAL module provides a generic interface for controlling different robots in joint space and implements hardware specific details. Via the simulation environment, the user also can directly interact with the HAL, e.g. by adjusting the joints of the robots or limiting their working space. In the following sections we will give an extensive bottom-up description of the control hierarchy mentioned above.

3.1 Hardware Abstraction Layer

Since we do not want to restrict our software to the interaction with a specific type of robot, we have implemented a control layer, which abstracts from hardware specific details. The input to this layer is joint angles, which are going to be retraced by the robots. Currently, our implementation can interact with two different types of robots: *Kuka KR6/2* [132] and *Mitsubishi MELFA 6SL* [150]. Additional types of robots can be added quite easily (cf. section 3.5). Joint angles must be provided in radians, implemented as 64bit `double` variables. A specific implementation must at least provide an interface for setting the robot to real-time mode and access functions for reading / writing the angles in real-time. Optionally, additional functionality, like setting the working space, adjusting joint limits etc. can be provided and offered to the user by superordinate modules.

3.1.1 Kuka Interface

As mentioned above, we have employed two Kuka KR6/2 robots for the first version of the system (EndoPAR). Unfortunately, the controller of the robots provides no straightforward low level real-time interface, which is required to implement trocar kinematics (see below). Therefore, we have based our control hierarchy on a self-made extension of the Kuka controller [183]. The controller interconnects to the robots via serial line. After switching the robots to real-time mode, a server, running on the Kuka controller, polls for new positions every 20ms. If proper timing is missed, i.e. if not at least one new position is provided during this interval, the server stops and sets back the robot to non real-time interaction.

3.1.2 Mitsubishi Interface

The Mitsubishi controller can be extended with an easy-to-use low-level interface. Its functionality is provided after installing an interface card and updating the controller's firmware. Afterwards, all commands can be issued via Ethernet. The interface provides a data link function and real-time control. The data link function can be used for offline alteration of certain parameters of the controller (e.g. restrictions of the working space or joint limits). In addition, it can be used to load control programs, start and stop servos, and subsequently, enable the real-time mode. For security reasons, all data link commands are transmitted via TCP packets (connection oriented). In contrast, the real-time commands are transmitted via the connectionless UDP protocol in order to increase transfer rates. Real-time commands are accepted every $6.8ms$ by the controller and can be issued to adjust joint angles or move in Cartesian space. Our implementation only manipulates joint angles, since we need a special type of inverse kinematics (see below) and all configurations should be checked for collisions before we send them to the controller. Using TCP and UDP for communication offers the advantage of implementing remote parts of the interface on a user-defined platform with an arbitrary programming

language. We have chosen a standard PC platform and have implemented our control software with C++. A typical example for issuing a real-time command in C++ is the following code extract:

```
sprintf(buff, "1;1;SRVON");
send(sockfdTCP, buff, strlen(buff), 0);
numrec = recv(sockfdTCP, buff, 512, 0);
```

The actual language commands to the controller (e.g. the command `1;1;SRVON`) are encapsulated inside a TCP packet, which can be generated and transmitted by an arbitrary programming language (here C++). Using these packet switched interfaces requires every controller to have a unique IP-address. In addition, a port must be set in order to establish a connection via network sockets.
To increase flexibility, every robot is controlled by a dedicated thread of our software. These threads are set to real-time priority in order to override all subordinate functions, like the graphical user interface. Although scheduling of threads cannot be guaranteed, and the underlying Ethernet infrastructure is susceptible to packet loss and unpredictable delay, the provided solution is, by far, fast and reliable enough to satisfy the real-time demands of the controllers of the robots.

3.1.3 *PHANToM* Interface

In order to feed back forces to the user, we have to transform the amplified measurements of the strain gauges to the corresponding coordinate frame of the *PHANToM*s. Since we have modified the layout of the device (cf. 2.4) and its driver (cf. A.6), we have to provide dedicated methods in order to feed back forces to the user. Force feedback depends on the position of the wrist relative to the base of the device (i.e. leverage effects have to be regarded).

force feedback
low-level interface

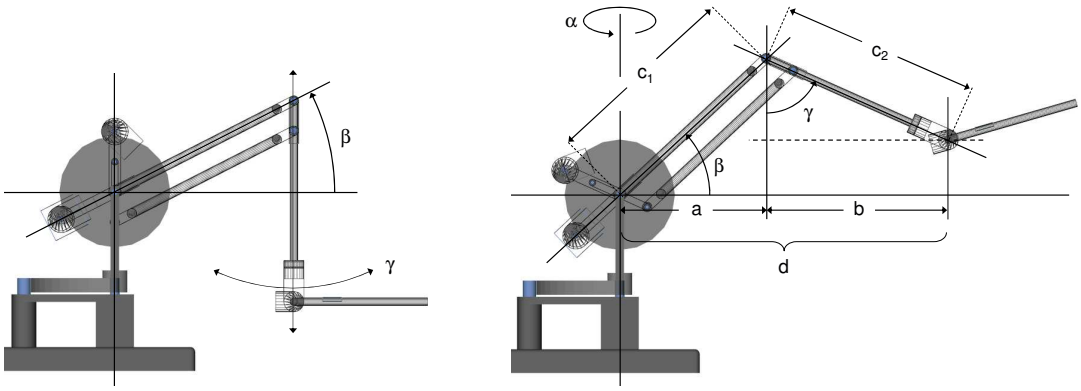


Figure 3.2: Independence of α and β (left) Calculation of d (right)

As one can derive from the left hand side of fig. 3.2, the upper arm of the PHANToM constitutes a parallelogram. This is very helpful regarding the determination of the position of the end effector and the calculation of forces, respectively. If the angle of the forearm servo (no. 3) is fixed at γ, changes in β will result in a parallel translation of the coordinate frame of the forearm, but no rotation is caused. Unlike conventional robot geometries (cf. fig. A.1), rotations of the coordinate frame of the forearm servo are independent from the position of the upper arm. The perpendicular distance d of the wrist from the base (cf. right side of fig. 3.2) amounts to:

$$d = a + b \quad \textit{where} \quad a = cos(\beta) \cdot c_1 \quad \textit{and} \quad b = sin(\gamma) \cdot c_2 \tag{3.1}$$

In order to alleviate arbitrary scaling, it is convenient to transform servo torques to normalized forces of $1N$ at distance c_1. The dependency between the integer values sent to the driver and the corresponding torque of the servo is not documented. Therefore, we have calibrated the system with a spring scale attached to the elbow of the device (i.e. at distance c_1). The shoulder servo (no. 1) contributes $\overrightarrow{F_1}$ to the overall force at the end effector. Given the geometry of the device (cf. left side of fig. 3.3), the components of vector $\overrightarrow{F_1}$ can be determined as follows:

$$\overrightarrow{F_1} = \begin{pmatrix} cos(\alpha) \cdot (d/c_1) \\ 0 \\ -sin(\alpha) \cdot (d/c_1) \end{pmatrix} \cdot 1N \tag{3.2}$$

The torque exerted by the upper arm servo (no. 2) will also be transformed into a normalized force of $1N$ at the elbow (at distance c_1). The force at the end effector will be the same, since the forearm always points towards the direction of the force vector (due to the constructional parallelogram of the upper arm).

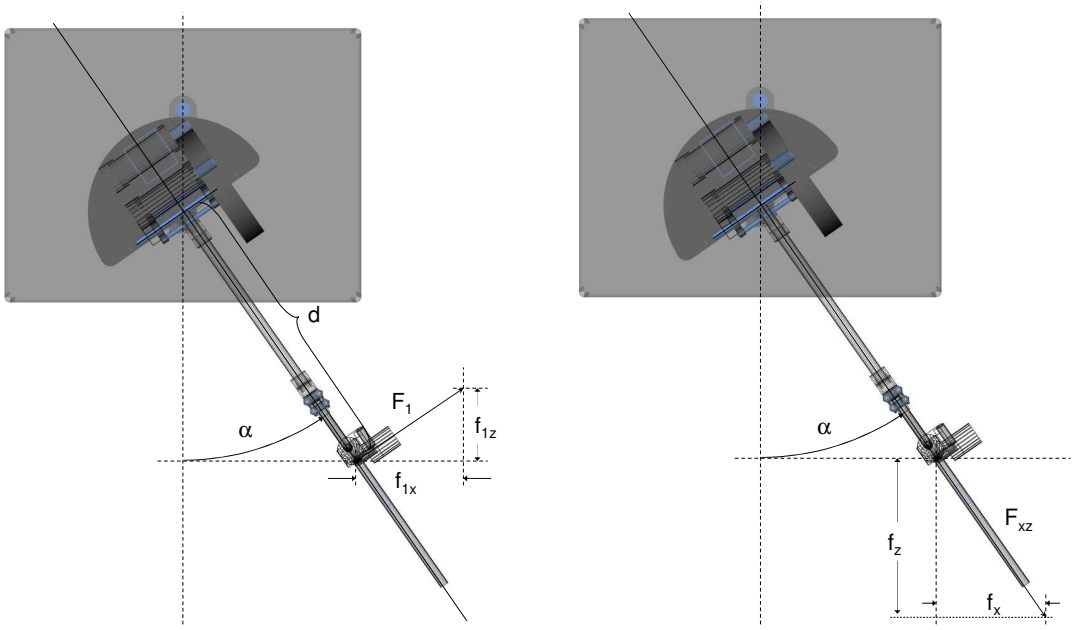


Figure 3.3: Force of servo 1 (left) and Subdividing force F_{xz} (right)

The components of the relevant force vector $\overrightarrow{F_2}$ can be derived from the left blueprint in fig. 3.4:

$$\overrightarrow{F_2} = \begin{pmatrix} \langle \overrightarrow{F_{2xz}}, \overrightarrow{e_1} \rangle \\ sin(\beta) \\ \langle \overrightarrow{F_{2xz}}, \overrightarrow{e_3} \rangle \end{pmatrix} \cdot 1N = \begin{pmatrix} sin(\alpha) \cdot cos(\beta) \\ sin(\beta) \\ cos(\alpha) \cdot cos(\beta) \end{pmatrix} \cdot 1N \tag{3.3}$$

where $\langle \overrightarrow{F_{2xz}}, \overrightarrow{e_i} \rangle$ is the scalar product of the force parallel to the xz-plane and the corresponding unit vector used for projection. These projections can be derived from the right side of fig. 3.4.
In a similar way we can determine the force exerted by servo 3. Again, we translate the torque of the servo into a unit force applied to the end of a lever of length c_1. Note that the distance between servo 3 and the wrist is c_2 (cf. fig. 3.2), but for the original and the toppled configuration of the PHANToM (cf. left and middle depiction in fig. 2.4) it holds: $c_1 = c_2$. Therefore, regarding force $\overrightarrow{F_3}$ (cf. fig. 3.4, right

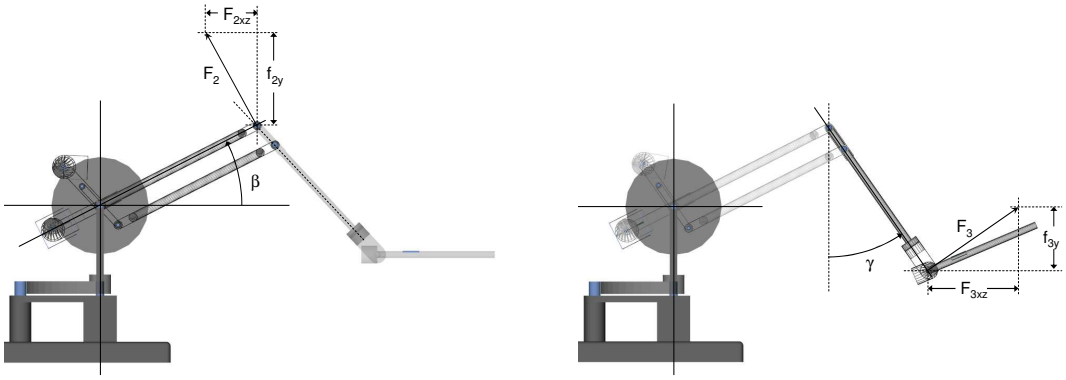


Figure 3.4: Force of servo 2 (left) Force of servo 3 (right)

side), we get:

$$\overrightarrow{F_3} = \begin{pmatrix} \langle \overrightarrow{F_{3xz}}, \overrightarrow{e_1} \rangle \\ sin(\gamma) \\ \langle \overrightarrow{F_{3xz}}, \overrightarrow{e_3} \rangle \end{pmatrix} \cdot 1N = \begin{pmatrix} -sin(\alpha) \cdot sin(\gamma) \\ sin(\beta) \\ -cos(\alpha) \cdot sin(\gamma) \end{pmatrix} \cdot 1N \tag{3.4}$$

Given these considerations, the overall force vector at the end effector (i.e. at the wrist of the device) amounts to: $\overrightarrow{F} = \sigma_1 \cdot \overrightarrow{F_1} + \sigma_2 \cdot \overrightarrow{F_2} + \sigma_3 \cdot \overrightarrow{F_3}$ where σ_i is the scaling factor of the corresponding servo torque. This expression constitutes the following linear system of equations:

$$\overrightarrow{F} = \begin{pmatrix} f_x \\ f_y \\ f_z \end{pmatrix} = \begin{pmatrix} f_{1x} & f_{2x} & f_{3x} \\ f_{1y} & f_{2y} & f_{3y} \\ f_{1z} & f_{2z} & f_{3z} \end{pmatrix} \cdot \begin{pmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \end{pmatrix} = F \cdot \overrightarrow{\sigma} \tag{3.5}$$

We can solve this system by inverting F. Before we can do this, we have to ensure that F is invertible, i.e. vectors F_1, F_2 and F_3 are linearly independent. Apart from this consideration, it is also important to assure invertibility of the corresponding Jacobian matrix, i.e. the matrix containing all partial derivations of the speed vector of the end effector. In case of singularity of matrix F the assignment of torques to the corresponding servos will not be unique. This will bring about problems if we move away from this position, but want to maintain a constant force at the end effector. Instead, an abrupt change of the torque's amplitude of a certain servo may result. In contrast, singularities of the Jacobian matrix indicate degenerated positions in the working space where it is not possible to exert forces in all directions. For example, if the position of the wrist intersects with the axis of servo 1, no forces can be applied in direction of the axes of servo 2 and 3, unless the arm is rotated at infinite speed. Since the mathematical examination of singularities is quite intricate and not necessary in our case, we restrict ourselves to geometrical considerations. As one can see from the blueprints of the device, there are only two positions of the device, which will cause problems: The first unfavorable position is reached if γ amounts to $180°$, i.e. the arm is completely stretched out. Anyway, this position can never be reached mechanically. The other case occurs, if the position of the wrist intersects with the axis of the first servo. Because corresponding positions can be reached, we switch off the device if the distance of the wrist from this axis falls below a threshold of $5\ cm$, which should never happen during normal operation. For all other positions we can determine the scale factors for the motor torques by solving equation 3.5, i.e. we are now able to apply arbitrary forces (within the absolute limits of the device) to the end effector, since we have solved all geometrical dependencies. Unfortunately, there are temporal dependencies too. If no forces are applied at a certain point in time and we suddenly

apply a specific force in the next time step, this will lead to undesirable jerks and vibrations of the end effector. In addition, the user will be surprised and may react in awkward ways. Therefore, we fade-

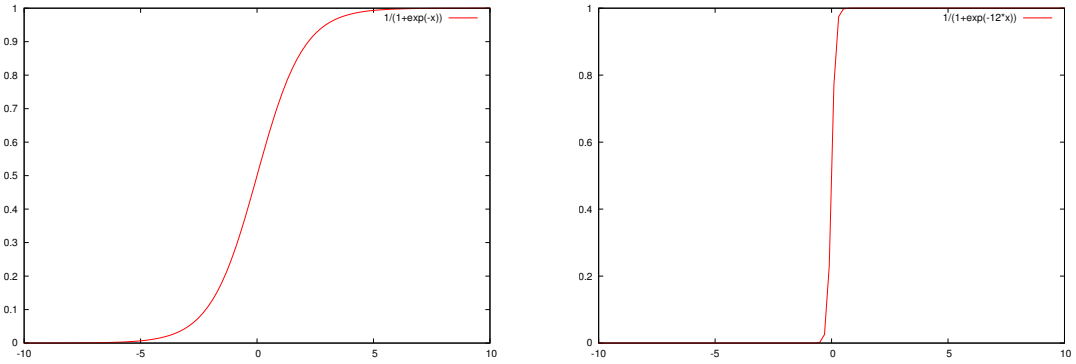


Figure 3.5: Different scale factors for the simulation of objects

in forces by increasing them smoothly by linear interpolation. This works fine, if we want to display measured forces, which are independent from the position of the end effector. In contrast, this method will not work, if we want to simulate rigid contacts with objects in the working space. Linear fade-in of forces will lead to an unrealistic impression of softness. On the other hand, sudden changes will lead to vibrations again. A reasonable compromise is to increase the force with an exponential function. We have chosen the function $\frac{1}{1+e^{-ax}}$, where x is the distance from the simulated surface and a will be interpreted as the softness of the object (cf. fig. 3.5). While the force progression, produced by the scale factor displayed at the left side, will emulate a rather soft object, a steeper function (right hand side) will give the impression of a hard object with little resilience. Note that the hardness of simulated objects is limited by the sampling rate of the device. A device with high sampling rate is able to display a steep force progression like depicted at the right side of fig. 3.5 without jerk. Our device has a sampling frequency of $1000\ Hz$, which is regarded to be the lower limit for jerk-free simulation of rigid contacts.

3.2 Collision Detection

Before joint angles are sent to the HAL module, postures of the manipulators can be optionally checked for collisions with various objects in the working space. This is especially useful, if semi-automated trajectories are going to be executed. Therefore, we have implemented a collision detection based on oriented bounding boxes (OBBs). Those are the smallest possible boxes enclosing a given object. Although objects usually are better approximated by convex hulls than boxes, the latter are favorable because of their outstanding computing performance. In addition, for most parts of our robots, the difference between the convex hull and the corresponding OBB is negligible. In comparison to axis-aligned bounding boxes (AABBs; cf. box A in fig. 3.6 (b)), the construction of OBBs cannot be solved analytically, and statistical methods like PCA will not yield an optimal solution. Fortunately, the CAD drawings of the robot's parts are already aligned to the coordinate axes, and therefore, the AABBs of the untransformed components of the robots are a pretty good approximation of the OBBs. All that is left is transforming them exactly like the corresponding components. As depicted in fig. 3.7, the Mitsubishi robots consist of seven independently moved parts. The seventh OBB around the flange is omitted, since it is covered by the OBB around the wrist, which is extended to create a safety margin.

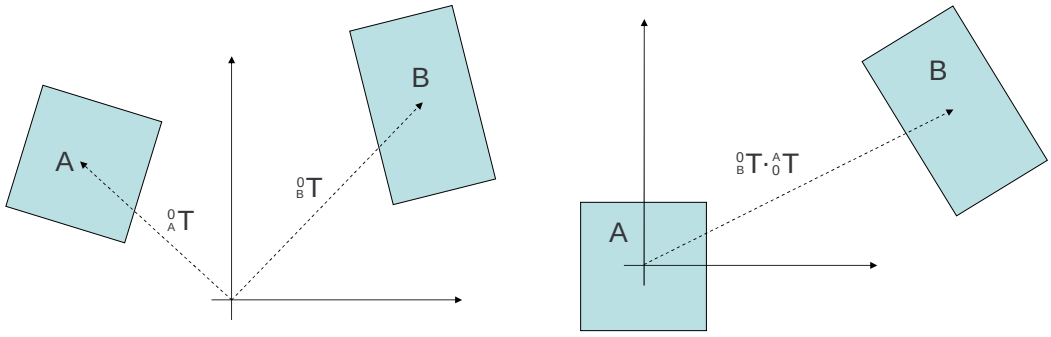


Figure 3.6: (a) OBBs before (b) and after transformation

We can easily use the OBBs described above to construct an AABB around the whole robot in every time step. This is achieved by picking the overall minimum and maximum value of all corners of the corresponding OBBs for each coordinate axis. Further calculations of robot-to-robot collisions are carried out, only if the engulfing AABBs of the corresponding robots intersect, thus, speeding up computation. If collisions cannot be excluded due to this test, we proceed with pairwise collision tests of the individual parts of the robots. Regarding OBB-OBB intersections, we have employed a similar algorithm as it is implemented in the Coin API [202]. Note that this algorithm is not optimized for minimization of overall operations, like the one presented in [80], but for minimization of matrix-vector operations. This feature will be exploited when porting the implementation to the graphics card.

collision detection by oriented bounding boxes

In order to detect a collision of box A with box B (cf. fig. 3.6), both are transformed by ${}^{A}_{0}T$, the inverse of the transform describing the tool frame of A. Therefore, coordinate frame A is now coincident with the origin, turning box A into an AABB. The definition of box A can be reduced to giving the minimum and maximum 3D-point of the AABB, $\overrightarrow{min}_A$ and $\overrightarrow{max}_A$. Consequently, the determination if corners B_i (of box B) are located within box A is reduced to the following interval checks:

$$c = \bigvee_{i=0}^{7} \left(B_i \geq \overrightarrow{min}_A \wedge B_i \leq \overrightarrow{max}_A \right)$$

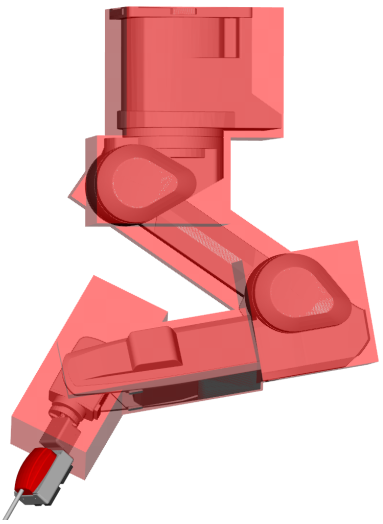


Figure 3.7: OBBs

If c is true, at least one corner of box B is situated within box A, and therefore, both boxes intersect and further checks can be omitted for this pair. Unfortunately, condition 3.2 is only sufficient, but not necessary. In some cases, all corners of box B are located outside box A, but they are nonetheless intersecting. This is due to the edges of B intersecting with planes of A, or vice versa. An example of this case is depicted in fig. 3.8 (a), where edge B_{56} intersects with the frontal plane of box A. Therefore, we have to apply an additional edge-box test, where all edges of box B are checked for intersections with faces of box A. In order to achieve this, we first normalize the edge vector (below the normalized edge vector will be referred to as $\vec{e}$). Now, we can check if this edge intersects with one of the faces constituting box B. If we want to apply this to the depicted example (intersect edge B_{56} with the minimum x-axis face of the AABB of box A), we first have to ensure that $min(B_5^x, B_6^x) < \ min_A^x < max(B_5^x, B_6^x)$. If this holds, we determine whether the intersection point $\vec{P}$ is

located on the corresponding face or not:

$$\vec{P} = \vec{S} + (min_A^x - min(B_5^x, B_6^x)) \cdot \vec{e}$$

where $\vec{S}$ is the corner of edge B_{56} with the lowest value for x (in this case $\vec{B_5}$, since $min(B_5^x, B_6^x) = B_5^x$). This test has to be repeated for each combination of the twelve edges of box B and the six faces of box A. Afterwards, it is still possible for boxes to intersect without meeting these prerequisites. An example is depicted in fig. 3.8 (b) where neither corners of box B are located within box A nor edges of box B intersect with box A. These cases are covered by interchanging the roles of A and B, i.e. the procedure is repeated for box B being transformed to the origin in order to constitute an AABB. This time it is sufficient to restrict the detection test to enclosed corner vertices.

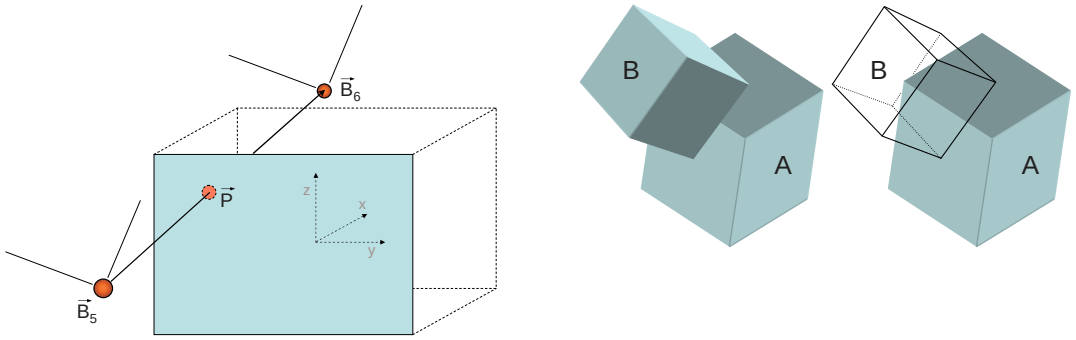


Figure 3.8: (a) Edge-box intersection (b) Special case of intersection

3.3 Inverse Kinematics

The inverse kinematics of individual types of robots can be found in the appendix (A.3). In this section we describe the trocar kinematics of our robotic system for minimally invasive surgery. Trocar kinematics means, the end effector (in our case an instrument for minimally invasive surgery) has to pass a fixed opening in the surface of the patient's body. This is the principle idea of minimally invasive surgery: to perform all surgical tasks through small keyhole-openings (so-called ports) in order to avoid traumatic cuts of tissue and bones. Evidently, this procedure restricts the degrees of freedom of the instrument. Feed and rotation axes always have to intersect with this fixed port. With respect to kinematics, this point is often called trocar point. Given the position and rotation of the end effector, we will get the position and rotation of the robot's flange as the result of the instrument's trocar kinematics. Therefore, we will gain full Cartesian control over the instruments. As a final result we will get three joint angles for the minimally invasive instrument, plus six angles for directly controlling the robot's joints (after applying the individual inverse kinematics of the robot, which bears the instrument). Recently, a comparable solution to this inverse kinematics challenge has been proposed by Locke et al. [42].

3.3.1 Mechanical Setup

As mentioned above in chapter 2, the motor part of our system consists chiefly of two instruments for minimally invasive surgery, which are mounted on industrial robots. Both instruments employed in our system are distributed by *Intuitive Surgical*. Each of them consists of several mechanical parts.

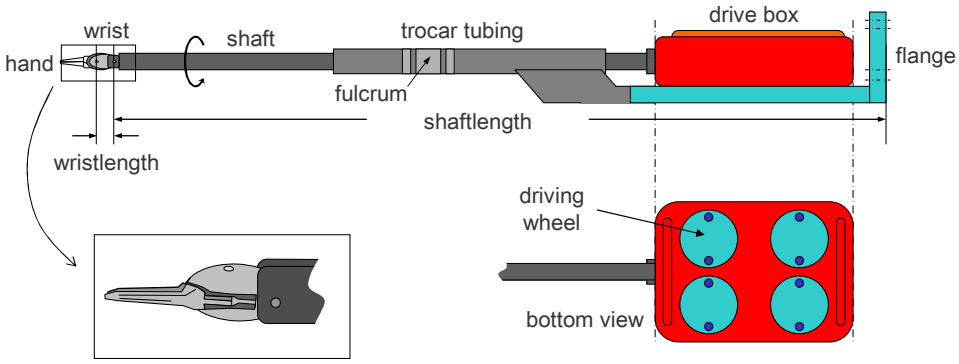


Figure 3.9: Instrument for Minimally Invasive Surgery

The part mounted on the driving device is called drive box. It contains four driving wheels, one for each degree of freedom of the instrument. Note that both fingers of the end effector can be moved independently. The driving wheels are connected to spindles, which control the moving parts of the instrument via steel wire linkage. The long cylindrical part, which catenates the drive box with the end effector, is called shaft. It is constructed as a hollow tube made of carbon fiber and contains the steel wires for end effector control. The shaft can be rotated about its longitudinal axis. The end effector consists of an iron wrist and hand. The hand itself has two fingers, which share one pivot axle. The axis of rotation of the wrist is arranged perpendicularly to the finger axis. For convenience we have chosen the following convention of axis assignments: The longitudinal axis of the shaft is conceived as x_M-axis. The rotational axis of the wrist constitutes the z_M-axis, while hand rotations take place about the y_M-axis. M indicates these axes being parts of the mechanical system (cf. fig. 3.10). Later, we will introduce different coordinate systems of the instruments in order to alleviate mathematical considerations.
The instruments are mounted on an aluminum adaptation, which can be flanged to the robots. Optionally, a second tube enclosing the shaft can be attached to this adaptation. It provides stability and prevents the patient's body from contacts with the rotating shaft. The last rotational axis of the robot and the rotational axis of the instrument are identical. Thus, rotations about the longitudinal axis of the shaft can either be executed by the servos, driving the rotational axis of the instrument, or directly by the robot.

3.3.2 Notation

In this section we often deal with different coordinate systems and conversions between them. Therefore, it is sensible to introduce some conventions (similar to those used in [75]). Within this section we will refer to coordinate systems with different capital letters. For example K denotes the coordinate frame of the robot (since the first type of robots we employed has been a Kuka KR6/2). For transformations between different system we use a capital T with the original system as superscript and the target system as subscript (both on the left side of the T). For example, ${}^{K}_{P}T$ denotes the transformation from the robot system into the Phantom system (i.e. frame P denoted relative to frame K). The rotational part of a transformation T_r is denoted as a 3×3-matrix, while translations T_t are indicated by subscripts regarding the corresponding axis. Therefore, a homogeneous transformation matrix will

look as follows:

$$ {}^{K}_{I}T = \begin{pmatrix} {}^{K}_{I}T_{00} & {}^{K}_{I}T_{01} & {}^{K}_{I}T_{02} & {}^{K}_{I}T_{x} \\ {}^{K}_{I}T_{10} & {}^{K}_{I}T_{11} & {}^{K}_{I}T_{12} & {}^{K}_{I}T_{y} \\ {}^{K}_{I}T_{20} & {}^{K}_{I}T_{21} & {}^{K}_{I}T_{22} & {}^{K}_{I}T_{z} \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

Vectors are represented by small letters, while points in 3D space are written down as capital letters. The corresponding coordinate system is indicated as a subscript. For example x_P means the x-axis of the PHANToM system and A_K is a point relative to the robot system. If the referred coordinate system is clear from the context, subscripts are omitted.

3.3.3 Coordinate Systems

We want to introduce some special coordinate systems in detail because we will use them for further explanations. The most important issue with this respect is the correlation between different parts of the systems. In order to facilitate the integration of trocar kinematics into the simulation environment (see below), we have determined a basis frame using the orientation of the *Coin* coordinate system [202]. Postures relative to the basis system are denoted as ${}^{0}_{X}T$, where X is the corresponding tool frame. The origin of our basis frame is placed at the bottom of the operating table (cf. fig. 2.2). Therefore, coordinates of positions of an instrument usually exceeds $1000mm$ in y-direction. In our setup the user takes place in front of the robots. Therefore, in the EndoPAR setup displayed in fig. 2.1, the x_K-axis points toward the direction of the user, while the z_K-axis points up to the ceiling. Since all systems have a dextral orientation, the y_K-axis points to the right hand side.
In the EndoPAR version of our system, the PHANToMs are placed in front of the user, and therefore, their z_P-axes are collinear with the x_K-axis of the robot system. The y_P-axis points up to the ceiling, while the x_P-axis points to the right hand side (cf. fig. 2.1). Note that this convention still holds for the new version of the system (ARAMIS) where the PHANToMs are turned upside down.

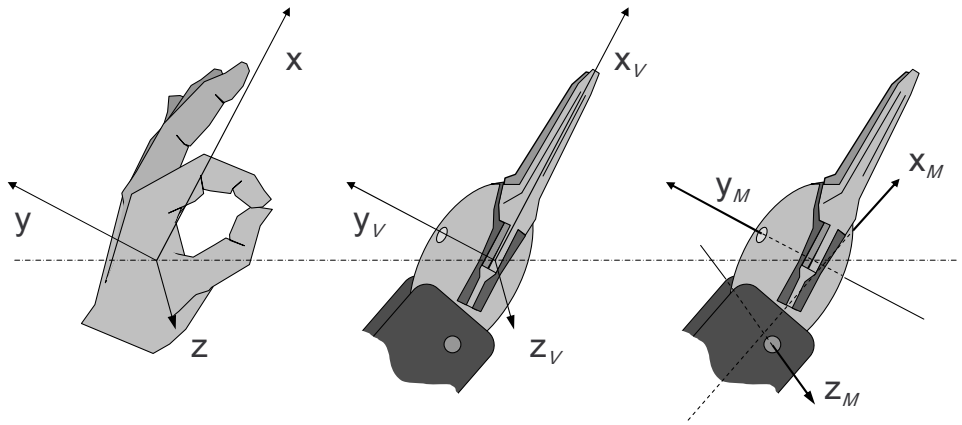


Figure 3.10: Tool frame of virtual instrument

A crucial issue is the determination of the coordinate system of the minimally invasive instrument. We want the control of the instruments to be as intuitive as possible. If we take the human hand as an archetype, we can make important observations: If we perform very precise manual tasks (e.g. a surgeon making a cut), we turn the hand about a rotation center lying near the first link of the fingers (cf. fig. 3.10 left side). Therefore, we introduce the tool frame of a virtual instrument as shown in the middle

of fig. 3.10. Like the human hand, rotations of the wrist turn about the z-axis and z_V-axis, respectively. Any rotation about these axes is called pitch. The x_V-axis (roll axis) points always toward the direction of the closed end effector (the grippers, scissors, a scalpel etc.). Consequently, the y_V-axis (yaw axis) is aligned as shown in fig. 3.10 in order to form a dextral system. Mechanical rotation axes of the real instrument are named x_M, y_M and z_M (cf. fig. 3.10 right side). Unfortunately, the mechanical axes do not coincide with the axes of the virtual tool frame. Therefore, an important issue is the mapping of movements of the virtual instrument on joints of the real world axes. We will refer to this later in detail.

3.3.4 Port Restrictions

Minimally invasive instruments are controlled by the operator via two PHANToM devices (6DoF position control with 3DoF force feedback). We want to generate the impression of the user controlling each instrument's motion as a direct mapping of the finger position (measured with the Phantom stylus: cf. fig. 2.1). This behavior has already been implemented with commercially available systems like *Intuitive's daVinci* and provides an overwhelming hand-eye coordination. This issue gets even more complicated as fixed fulcrum points (ports) for the instruments are required. As mentioned above, these ports are small incisions in the patient's body surface, which render the surgeon possible to reach positions inside the body without making an invasive cut. In most of the cases, three ports are required (cf. fig. 3.11). Two for the instruments and one for the endoscopic camera. Therefore, possible movements of the instrument's shaft are restricted to insertion, retraction and rotation exclusively about the corresponding port. All other motions (e.g. tangential translations along the body surface) would force the port to displace and harm the patient. Therefore, the axis of the shaft of each instrument must always be aligned with the corresponding port.

trocar kinematics

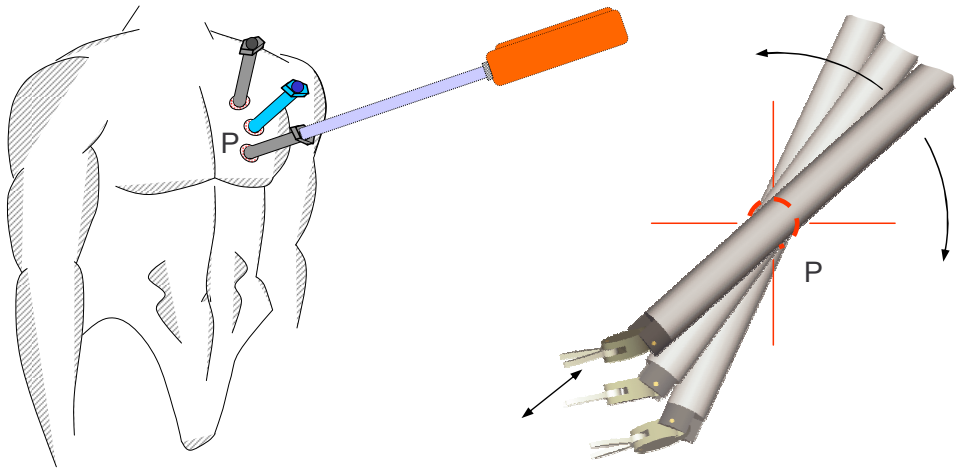


Figure 3.11: Location of the instrument and camera port

The most comfortable way of moving a surgical instrument inside the body is Cartesian control. Therefore, we assume the position of the instrument's end effector (e.g. gripper, scissors etc.) is given by a homogeneous transformation matrix derived from the input device. Positions and rotations of the instruments, as well as the position of the port, are denoted relative to the basis frame 0; i.e. postures of the PHANToM device have to be appropriately transformed. As we mentioned above, the x_V-axis of the instrument is always aligned with the end effector. The stylus of the PHANToM initially points

toward the direction of the z_P-axis (cf. fig. 2.1). Therefore, if we intend to employ the stylus as a surrogate for the surgical instrument, we have to rotate the stylus matrix by -90 degrees. This induces the impression of manually controlling the instrument's end effector (as depicted in figure 2.1), and thus, increases immersion. As a result, we get the transformation matrix ${}^0_V T$, for converting stylus movements (i.e. user inputs for the posture of the virtual instrument system) to instrument postures in the basis system.

$$
{}^0_V T = {}^0_P T \cdot {}^P_Y T \cdot {}^Y_V T =
$$

$$
= \begin{pmatrix} 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} {}^P_Y T_{00} & {}^P_Y T_{01} & {}^P_Y T_{02} & {}^P_Y T_x \\ {}^P_Y T_{10} & {}^P_Y T_{11} & {}^P_Y T_{12} & {}^P_Y T_y \\ {}^P_Y T_{20} & {}^P_Y T_{21} & {}^P_Y T_{22} & {}^P_Y T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & {}^Y_V T_x \\ 0 & 1 & 0 & {}^Y_V T_y \\ 0 & 0 & 1 & {}^Y_V T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}
$$

Initially, we convert postures determined in the PHANToM frame into postures described in the basis frame by applying ${}^0_P T$. ${}^P_Y T$ denotes the posture of the tool frame of the stylus relative to the PHANToM system, i.e. this transformation actually determines the user's motions at the input devices. Finally, a translation ${}^Y_V T$ is applied in order to set the starting points (offsets) of the individual instruments. The resulting matrix ${}^0_V T$ describes the tool frame of the virtual instrument relative to the basis frame.

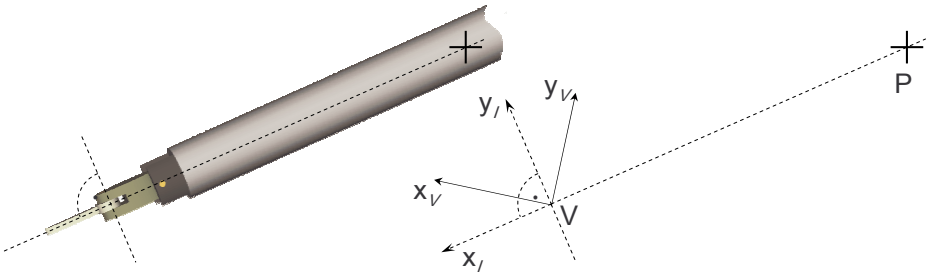


Figure 3.12: Initial position of the virtual instrument

As a next step, we determine the tool frame of the instrument's shaft I, when all joints are set to zero. In this initial state, the alignment of the shaft is determined by the positions of the port P and the virtual instrument V (cf. fig. 3.12). Point V marks the origin of the virtual instrument system, i.e. $V = {}^0_V T$. Therefore, the x_I-axis of the shaft is identical with the vector $\overrightarrow{PV}$. For convenience, we preset the z_I-axis of this system to be parallel to the xz_0-plane of the basis system.

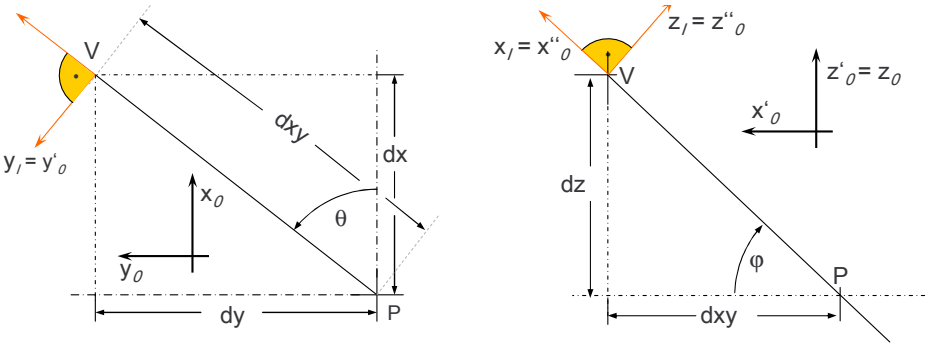


Figure 3.13: Z-Y-(X)-Euler angles of the virtual instrument system

We determine the rotation of tool frame I by means of two consecutive Euler-rotations, which transform the basis system to I. The angle of the first rotation about the z_0-axis is denoted as θ. It can be

determined by means of the $atan2$-function, which serves in robotics as a substitute for the well-known tan^{-1}-function. We measure the distance between the port and the instrument's position by the length of the difference vector $\vec{P} - \vec{V}$. The left side of figure 3.13 shows the first quadrant of the xy-plane of the basis system. By definition $atan2(dy, dx)$ will always calculate the angle θ, which rotates the original system about the z_0-axis until the new vector y_0' it is collinear with the y_I-axis of the shaft in its initial position. As result we get a rotated intermediate system with axes x_0', y_0' and $z_0' = z_0$. In order to get to the desired initial position of the shaft, we subsequently rotate the intermediate system about the y_0'-axis until x_0'' is collinear to $\vec{PV}$. In order to achieve this, we calculate the length of dxy by Pythagoras' law: $dxy = \sqrt{dx^2 + dy^2}$. Therefore, dxy will always be positive. The distance is determined by the z-component of the difference vector $\vec{P} - \vec{V}$. By applying the $atan2$-function, we get the angle φ. Remember, since dxy is always positive, $atan2(dz, dxy)$ always returns the acute angle between $\overrightarrow{PV}$ and the $x'y_0'$-plane. Since the y_0'-axis points out of the drawing plane of figure 3.13 (right side), we have to apply $-\varphi$ for a dextral rotation. In order to ensure the y_0''-axis being parallel to the xy_0-plane, we set the rotation about the x_0''-axis to zero. Now, we have reached the initial instrument position I: i.e. $x_0'' = x_I$; $y_0'' = y_I$; $z_0'' = z_I$. Given θ and $-\varphi$, we can construct the corresponding homogenous rotation matrix ${}^0_I T$, which transforms the basis system to the initial position of the instrument. A detailed derivation of this formula can be found in various books about robotics (e.g. [75] or [223]). Therefore, we abstain from further details here.

$$ {}^V_I T = \begin{pmatrix} \cos(\theta) * \cos(\varphi) & -\sin(\theta) & -\cos(\theta) * \sin(\varphi) & {}^0_V T_x \\ \sin(\theta) * \cos(\varphi) & \cos(\theta) & -\sin(\theta) * \sin(\varphi) & {}^0_V T_y \\ \sin(\varphi) & 0 & \cos(\varphi) & {}^0_V T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

Note that we have already applied the following equivalences: $cos(-\varphi) = cos(\varphi)$ and $sin(-\varphi) = -sin(\varphi)$. Translations in matrix ${}^V_I T$ are the same as in ${}^0_V T$ (cf. fig. 3.12), but rotations are different. While the rotational part of ${}^0_I T$ gives the position of the instrument with no rotations applied to the joint angles, ${}^0_V T$ determines the state after the intended rotation of the instrument's axes. In order to apply this rotation mechanically, we need to calculate the transformation from ${}^0_I T$ (initial position) to ${}^0_V T$ (desired position). Or in other words, we have to determine ${}^I_V T$. This is best be done by multiplying the inverse matrix of the description of the initial position onto the right side of ${}^0_V T$: ${}^I_V T = {}^0_V T \cdot {}^0_I T^{-1}$. By means of this result, we can determine the Y-Z-X fixed rotation angles for the instrument. In contrast to Euler rotations, fixed (yaw-pitch-roll) rotations always refer to the initial system. According to [75] the Y-Z-X fixed rotation can be expressed by the following rotation matrix:

$$ {}^I_V T_{(YZX)} = \begin{pmatrix} c(\beta)c(\gamma) & -s(\beta) & c(\beta)s(\gamma) \\ c(\alpha)s(\beta)c(\gamma) + s(\alpha)s(\gamma) & c(\alpha)c(\beta) & c(\alpha)s(\beta)s(\gamma) - s(\alpha)c(\gamma) \\ s(\alpha)s(\beta)c(\gamma) - c(\alpha)s(\gamma) & s(\alpha)c(\beta) & s(\alpha)s(\beta)s(\gamma) + c(\alpha)c(\gamma) \end{pmatrix} $$

where $c(x)$ and $s(x)$ are used as abbreviations for $\cos(x)$ and $\sin(x)$, respectively. Therefore, the individual rotation angles α_V, β_V and γ_V can be extracted as follows:

$$ \begin{aligned} \alpha_V &= atan2({}^I_V T_{21}, {}^I_V T_{11}) \\ \beta_V &= atan2(-{}^I_V T_{01}, \sqrt{{}^I_V T_{11}^2 + {}^I_V T_{21}^2}) \\ \gamma_V &= atan2({}^I_V T_{02}, {}^I_V T_{00}) \end{aligned} $$

We want to give a brief explanation, why we have chosen Y-Z-X fixed rotations for extracting the angles. This order of angles can be consecutively applied to the joints of the instruments. The first

rotation about the y_I-axis arranges for movements of the grippers of the instrument. It can be directly applied to the mechanical y_M axis. Note that this rotation of the instrument leaves the positions of all mechanical axes unchanged. This is quite useful, since Yaw-Pitch-Roll angles refer to a fixed system. As a next step we apply the rotation about the z_I-axis, by bending the wrist of the instrument. Since z_I and z_M do not coincide, the corresponding joint angle to be applied to the z_M-axis needs further calculations (see below). By moving the wrist, the mechanical rotation axis of the grippers will also change, but this does not matter since we have already applied the corresponding rotation of the grippers. The remaining rotation about the initial x_I-axis is performed by turning the shaft as explained below.

Every other order of angles will either have no mechanical correspondence (e.g. Z-Y-Z) or will lead to errors due to the fact that already executed rotations are changed afterwards due to mechanical dependencies. For example, assume we had extracted Y-Z-X-Euler angles from the matrix. If we now perform the rotation about the y_I-axis, the x_I-axis changes its location to $x'_I \neq x_I$ and the z_I-axis to $z'_I \neq z_I$, respectively. In contrast, mechanical application of the corresponding joint angle to the grippers will not change the position of y_M, which will thus not be collinear to y_I anymore. Therefore, subsequent rotations about the y'_I-axis cannot be mapped on the mechanical y'_M-axis anymore.

In 3.6 we have determined the angles for moving the virtual instrument from its initial position to the desired posture. As mentioned above, we have assumed that all rotational axes of the virtual instrument intersect in one point. We now have to consider how to simulate this behavior with the real instrument and its mechanical axes. The mechanical y_M-axis and the virtual y_V-axis are identical as we have mentioned above. Therefore, we can directly apply α, the rotation about the y_V-axis, to its mechanical counterpart, the yaw of the fingers ($\alpha_M = \alpha_V$).

Because the y_M- and the z_M-axis of the instrument do not intersect, we cannot apply the z_V-rotation without modifications. We have to calculate the corresponding angle from known geometric features. We already know the rotation angle about the z_V-axis of the virtual instrument (β_V: cf. fig. 3.14). In addition, the shaft of the instrument always has to pass the port P. Due to the Y-Z-X fixed angle convention, the virtual z-axis and its mechanical counterpart are always parallel (but not necessarily identical). Therefore, we can calculate the rotation about the mechanical y_M-axis (β_M). Note that the position of ${}^0_V T_t$ remains unchanged, but the shaft of the instrument is tilted about the port.

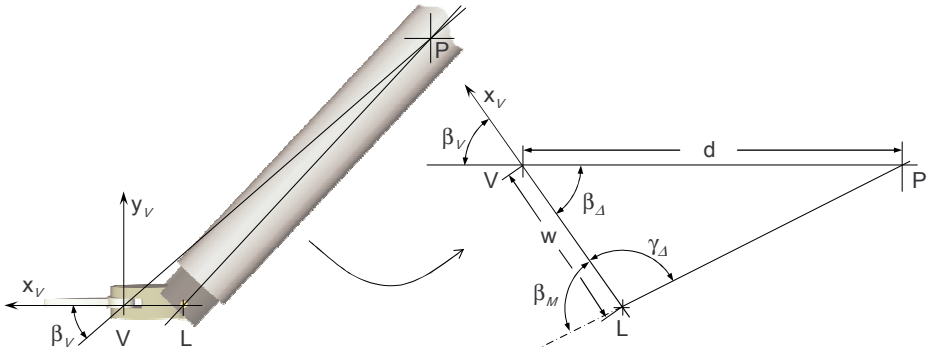


Figure 3.14: Calculation of β_M

As one can derive from figure 3.14, we already know two sides of the triangle, w and d, where w is the length of the instrument's wrist and d is the distance between the port and the position of the virtual instrument system $V = {}^0_V T_t$. Additionally, we know the rotation angle $\beta_\triangle = \beta_V$, which is constituted by w and d. Therefore, the following equations hold:

$$\tan\left(\frac{\gamma_\triangle - \alpha_\triangle}{2}\right) = \frac{d-w}{d+w}\cot\left(\frac{\beta_\triangle}{2}\right), \quad \frac{\gamma_\triangle + \alpha_\triangle}{2} = 90° - \frac{1}{2}\beta_\triangle;$$

By applying $atan$ to the first equation and adding the second one, we get:

$$\gamma_\triangle = \tan^{-1}\left(\frac{d-w}{d+w}\cot\left(\frac{\beta_\triangle}{2}\right)\right) + 90° - \frac{1}{2}\beta_\triangle;$$

Accordingly, we derive $\beta_M = 180° - \gamma_\triangle$. Note that we will use this procedure regardless of the sign of β_V. We always use the absolute value of β_V within the formula explained above, but we remember the sign for later adaption of the resulting angle β_M.

The last angle remaining is the mechanical rotation of the instrument's shaft. In order to reach the desired posture of the virtual instrument, we perform a rotation of γ about the initial x_I-axis (see derivation of transform ${}^I_V T$) ending up with system V. Therefore it holds: $x_V = x''_I$, $y_V = y''_I$ and $y_V = y''_I$. Unfortunately, we can only apply rotations to the real x_M-axis, which is identical with the z_F-axis of the robot's flange (cf. fig. 3.15). It should be clear from picture 3.14 that the longitudinal axis of the shaft x_M ($\overrightarrow{PL}$) is not collinear with the x''_I-axis ($\overrightarrow{PV}$) as long as β_M is different from zero. Therefore, we have to determine γ_M by an additional calculation. The chosen convention is that the y_F-axis is initially parallel to the xy_0-plane. Note that every other convention would naturally change the outcome of γ_M, but will not influence the final position of the instrument. In figure 3.15, the point V^+ is the position of the virtual instrument if γ_M were zero. Therefore, we have to rotate the shaft about x_M until V^+ coincides with the actual position of the virtual instrument $V = {}^0_V T_t$. In other words, we have to find a rotation that transforms x_F, which is parallel to a plane spanned by y_0 and $\overrightarrow{PL}$, to y''_I, which is parallel to a plane spanned by $\overrightarrow{PL}$ and $\overrightarrow{PV}$ (cf. fig. 3.15). In order to determine this planes, we first have to find the position of the inflexion point L (cf. fig. 3.14). We know the posture of the virtual instrument ${}^0_V T$. Since axes y_V- and y_M are identical, we can get the orientation of the wrist ${}^0_W T_r$, by back-rotating ${}^0_V T$ by $-\alpha_M$. The x-axis of the resulting system points against the direction of the wrist, and therefore, we can step back on this axis for the known length of the wrist w. Thus, altogether we get:

$$\overrightarrow{L} = {}^0_V T \cdot \begin{pmatrix} \cos(-\alpha) & 0 & \sin(-\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(-\alpha) & 0 & \cos(-\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -w \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

Given all points, we now can determine γ_M. The best way to do this, is calculating the intersection angle of the plane spanned by V^+, L and P, and the plane spanned by V, L and P. We get this angle by intersecting the normals of these planes. The plane spanned by V^+, L and P can be constituted by the cross product of vector $\overrightarrow{PL}$, which is identical with the rotation axis of the instrument's shaft x_M, and the y_0-axis. The normal of the other plane, spanned by the target points of the instrument (V, L and P), is determined by the cross product of vectors $\overrightarrow{PV}$ and $\overrightarrow{PL}$. Note that the direction and order of vectors for the cross product is crucial, in order to get the right direction of the normals. The angle between the normals can be calculated with the formula for the scalar product. Since the result will always lie between $0°$ and $180°$, we also have to find the right quadrant, which can be derived from γ_V. Finally, we have all angles to control the instrument.

The next step is to determine the position and rotation of the flange of the robot (${}^0_F T$), where the

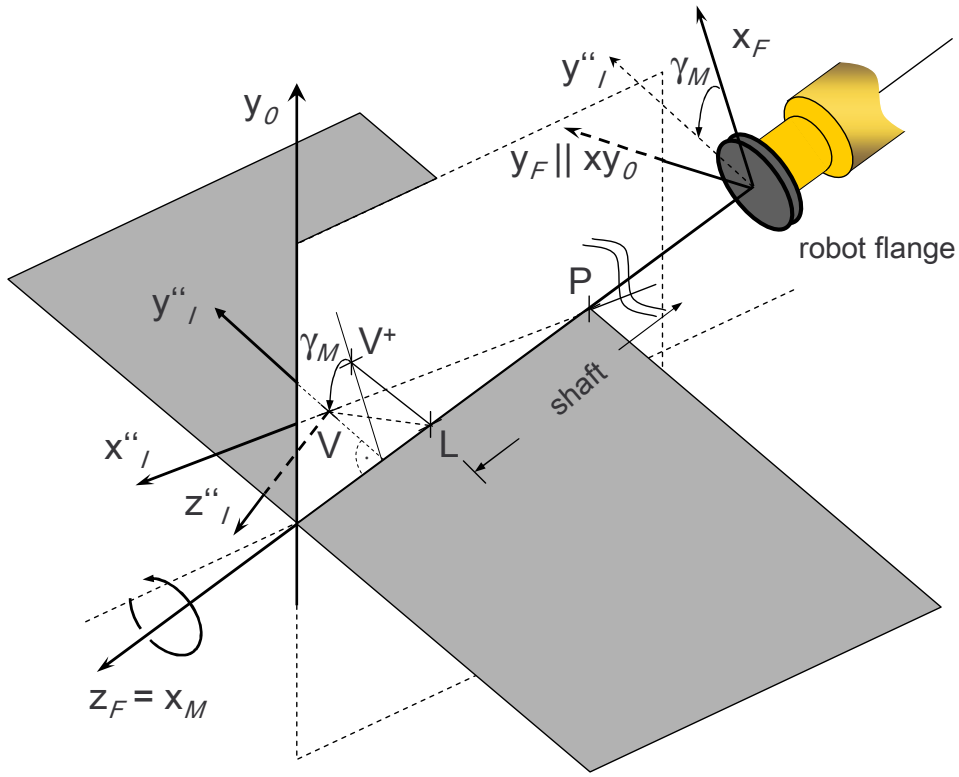


Figure 3.15: Calculation of γ_M

instrument is fixed to. The x_M-axis of the instrument is collinear with vector $\overrightarrow{PL}$, while the y_M-axis is parallel to the xy_F-plane. Since the points P and L are given, we can directly derive the necessary rotation angles for the transformation ${}^0_FT$, which describes the position of the robot's flange in basis coordinates. This is done in similar fashion as it was for the initial position of the virtual instrument (cf. fig. 3.12). In contrast to the arrangement of ${}^0_IT$, we do not use the formula for Z-Y-X-Euler angles here, but Z-Y-Z-Euler angles. This has the advantage, that we can optionally perform a part (or even the complete) instrument rotation by means of the last axis of the robot. In other words: $\gamma_M = \gamma'_M + \gamma_F$. Once we have identified the rotation of the flange, we can simply find out its position by stepping back for the length of the shaft l in negative z_F-direction, starting at the known inflexion point L. As we have explained above, L marks the link between wrist and shaft. If we take α_F for the z_F-rotation, β_F for the y'_F-rotation and γ_F for the z''_F-rotation, we get the transformation matrix ${}^0_FT$:

$$\begin{pmatrix} c(\alpha)c(\beta)c(\gamma) - s(\alpha)s(\gamma) & -c(\alpha)c(\beta)s(\gamma) - s(\alpha)c(\gamma) & c(\alpha)s(\beta) & L_x - {}^0_FT_{02} \cdot l \\ s(\alpha)c(\beta)c(\gamma) + c(\alpha)s(\gamma) & -s(\alpha)c(\beta)s(\gamma) + c(\alpha)c(\gamma) & s(\alpha)s(\beta) & L_x - {}^0_FT_{12} \cdot l \\ -s(\beta)c(\gamma) & s(\beta)s(\gamma) & c(\beta) & L_x - {}^0_FT_{22} \cdot l \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where, for example, $\cos(\alpha_F)$ is abbreviated by $c(\alpha)$. We will use this matrix as an input for the inverse kinematics of the corresponding robot. The inverse kinematics of individual robots is described in detail in the appendix (A.3).

3.3.5 Application of Forces

For a proper feedback of forces, it is required to transform the sensor readings of the strain gauges (cf. chapter 2) to the PHANToM system. Since the sensors are mounted in perpendicular arrangement onto the shaft, they produce force vectors normal to the shaft and normal to each other (cf. fig. 3.16). The orientation of force vectors relative to the basis system can be found by applying matrix ${}^{0}_{F}T$, which describes the posture of the flange. Note that both the flange and the attached instrument share one

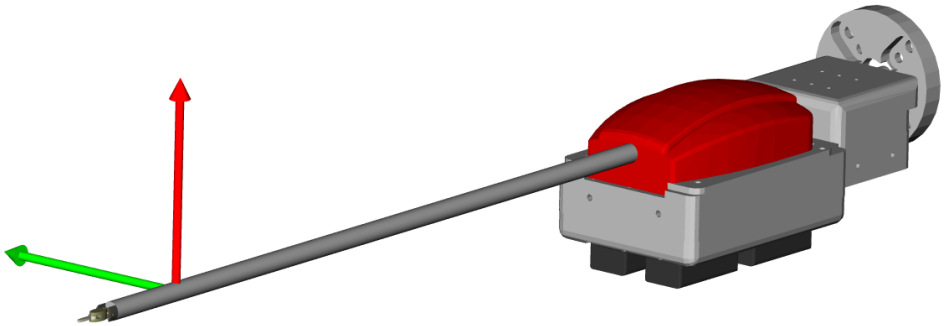


Figure 3.16: Orientation of force vectors

axis of rotation, and force vectors ${}^{G}\mathsf{f}_x$ and ${}^{G}\mathsf{f}_y$ are collinear to ${}^{0}_{F}T_x$ and ${}^{0}_{F}T_y$, respectively. Therefore, we can easily calculate the force vectors with respect to the base system: ${}^{0}\mathsf{f} = {}^{0}_{F}R \cdot {}^{G}\mathsf{f}$, where ${}^{0}_{F}R$ is the rotational part of ${}^{0}_{F}T$ and G is the tool frame of the gauges, which is parallel to F. The lengths of the vectors correspond to the amplified and calibrated sensor readings. It is possible to set an offset, which will be subtracted from the actual readings. This helps to compensate for longterm thermal fluctuation of the sensor values. The offset can be reset by a foot switch, which is pressed, if no forces are exerted on the gauges. After applying ${}^{P}_{0}T_y$, the inverse transformation matrix of the PHANToM's tool frame, we can display the forces to the user by means of the HAL of the feedback devices.

transformation of forces

3.3.6 System Calibration

An important feature of every robotic system is its absolute and relative accuracy. Absolute accuracy means the extent of aberration of the end effector from a desired point in Cartesian coordinates. This type of accuracy is predominantly limited by constructional variations: i.e. the dimensions of certain assembly parts may vary between different robots with the same layout. Fortunately, the absolute accuracy of the Mitsubishi robots is very exact (our own measurements have exposed an error of about 1mm). Relative accuracy refers to the aberration between two reiterations of the same trajectory with the same robot. This plays an important role in manufacturing where robots are often programmed to perform the same trajectory over and over again ("Teach-In"). In general, relative accuracy is often much better than absolute accuracy, since relative accuracy is only limited by the quality of the encoders and the accessory control loop. Anyway, relative accuracy does not play a dominant role in our scenario because most features of our software will rely on absolute Cartesian coordinates. Unfortunately, the absolute accuracy of our system is limited by additional factors: all robots are mounted on a gantry (cf. fig. 2.2) which is afflicted by several intrinsic aberrations: variations in the dimensions of the elements, errors of mounting angles and so on. For our research it is particularly important that all robots refer to the same global coordinate frame. Therefore, we will present a method for establishing such a coordinate system. We have decided to refer to the coordinate system of our simulation environment as global system. Without restrictions it is not possible to replicate this system in real world, since we do not know the exact measures of the gantry. On the other hand, the robots we use possess an absolute calibration, which is very accurate. Therefore, we decided to center the global system around a certain robot, i.e. we assume that one of the robots is mounted in perfect congruence with the underlying CAD model and the calibration of all other robots and objects are brought into line with this robot. In order to reach this, we employ the following error model:

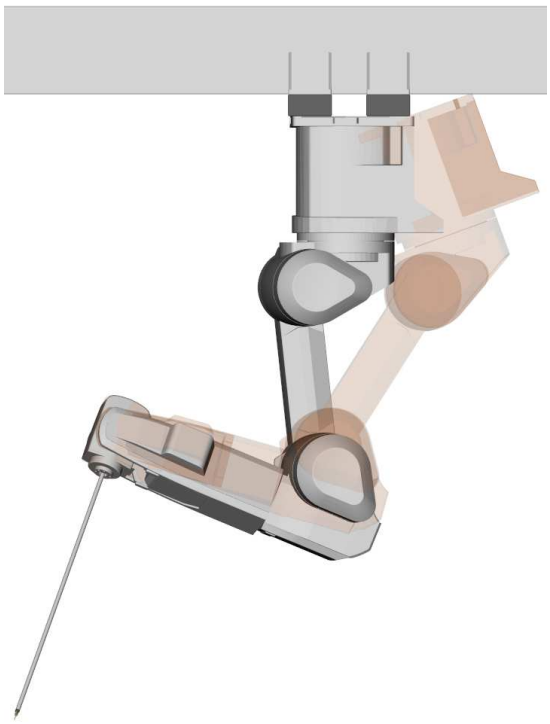


Figure 3.17: Calibration of the robots

$$ {}^{0}_{R_1}T \cdot {}^{R_1}_{C}T = {}^{0}_{R_2}T \cdot {}^{R_2}_{R_2'}T \cdot {}^{R_2'}_{C}T \tag{3.6} $$

In this equation ${}^{0}_{R_1}T$ is the position of the base of the reference robot R_1, expressed in global coordinates taken from the CAD model. As mentioned above, we assume that this position is perfectly reproduced within the real system. The position of any other robot is determined by ${}^{0}_{R_2}T \cdot {}^{R_2}_{R_2'}T$, where ${}^{0}_{R_2}T$ would have been the position of the other robot, presuming perfect assembly. Because perfect assembly is rarely reached in practice, we assume that the real position of the robot suffers from a displacement expressed by the transform ${}^{R_2}_{R_2'}T$. In order to measure this displacement we define a

calibration frame C in global coordinates, i.e. the position and orientation of this frame is given in local coordinates of the perfectly mounted robot ${}^{R_1}_{C}T$. In reality we can replicate this frame by mounting a calibration trihedron on the flange of robot R_1. Now, we can determine the position of this trihedron in coordinates of robot R_2 (i.e. ${}^{R_2'}_{C}T$) by touching its ends with the tip of robot R_2. Further transformations of 3.6 yield:

$$ {}^{R_2}_{R_2'}T = {}^{0}_{R_2}T^{-1} \cdot {}^{0}_{R_1}T \cdot {}^{R_1}_{C}T \cdot {}^{R_2'}_{C}T^{-1} = {}^{R_2}_{0}T \cdot {}^{0}_{R_1}T \cdot {}^{R_1}_{C}T \cdot {}^{C}_{R_2'}T \tag{3.7} $$

Now the error model ${}^{R_2}_{R_2'}T$ can be integrated into our planning environment. Fig. 3.17 shows the CAD model of an uncalibrated robot and its calibrated version as transparent overlay. Note that the extent of aberration is exaggerated for clarity of depiction. Rotational errors usually do not exceed a value of one angular degree. After this part of the calibration, our planning environment will produce the correct angles with its inverse kinematics for any point given in global coordinates (i.e. what you see in the simulation environment is what you get in reality).

Unfortunately, mounting displacements of the robots are not the only source of errors in our system. If we go down further, we find that also the attachment of the instruments bears certain variances. In theory, the shaft of the instrument should align with the z-axis of the robot's tool system. In practice, a perfect alignment is improbable due to the fact that the instrument is mounted with a magnetic coupling (see above) and even the original instruments themselves are not perfectly straight. This leads to a certain excentricity (cf. fig. 3.18), if the instrument deviates from the z_F-axis of the flange. For the following compensation estimate, we use an approximation which simplifies the calculation and applies only to small angles. An aberration dx and dy from the center will lead to a positional error of approx. $\sqrt{dx^2 + dy^2}$. The parameters depicted in fig. 3.18 can be found by positioning the instrument over a

compensate for excentricity

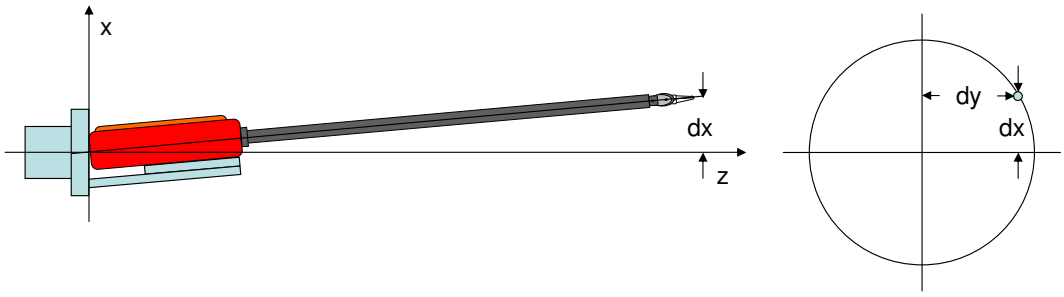


Figure 3.18: Excentricity error of the instrument

plane surface with the z-axis of the robot's tool system normal to the surface. By performing a $360°$ movement of the end effector, the relevant parameters can be determined. In order to compensate for this excentricity, a correctional transformation has to be applied to the end effector prior to the calculation of the inverse kinematics of the robot. The inverse trocar kinematics of the instruments (see above) yields the transformation matrix of the robot's end effector under ideal circumstances. If the instrument suffers from excentricity, this matrix has to be adapted by turning it about the x-axis for approx. $sin^{-1}(dx/l)$ degrees and about the y-axis for approx. $sin^{-1}(dy/l)$, where l is the length of the instrument. Afterwards, the corrected matrix will serve as input for the inverse kinematics of the robot.

3.4 User Interface

While the hardware part of the user interface has already been presented in chapter 2, we will focus here on the description of software modules, which enable the online operation of the system as well as offline planning and analysis. All modules are integrated in an easily operated GUI, abstracting from the low-level modules described above.

3.4.1 Online Control

In order to control the system directly by the input devices, we provide an online control module, which interconnects the necessary components. Postures displayed by the user are acquired by means of the PHANToM devices and transformed to joint angles by the inverse kinematics module (see above). These joints are sent to the corresponding robots and instruments in real-time, thus, changing the state of the system. This may induce forces at the end effectors, caused by interaction with objects in the environment (e.g. tissue, suture material or other instruments). Occurring forces are measured by strain gauge sensors near the distal end of the instrument's shaft. After amplification and filtering, forces are fed back to the user via the PHANToM devices. During operation, all parameters of the system (postures, forces, switch states etc.) can be recorded to a `MySQL` data base [162].

Before starting the online control, users can activate the desired features by means of the GUI (cf. fig. 3.19). The combo box at the top provides a selection of predefined settings. These settings describe what is going to be performed after pressing the start button. For example, the setting selected in fig. 3.19 engages bimanual operation of the system, i.e. the left and right PHANToM is connected to the left and right robot in the front, respectively. Connections refer to the flow of information, i.e. which PHANToM provides input for which manipulator (and whose force measurements will in turn be fed back to the connected PHANToM). In addition, the functionality of some foot switches is determined here. If a set-up uses more than two robots, a dedicated foot switch is defined for changing the connections between the input devices and different manipulators (cf. section 3.4.2). These preassembled settings offer a shortcut to the functionality provided by the context sensitive menus (cf. section 3.4.3). For example, the program "2 arms: manual" could have also been activated by starting the robots separately and connect them to the PHANToMs, which can be started afterwards.

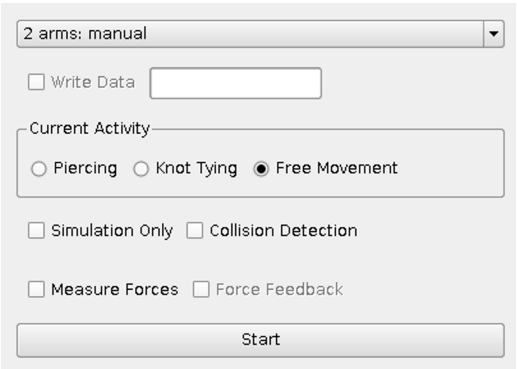


Figure 3.19: Online GUI

If the check box *Simulation only* is activated, the manipulators will stay offline and all movements are exclusively carried out in the simulation environment. This is quite helpful in order to test new arrangements of the robots for feasibility. An optional collision detection (cf. section 3.2) can be activated by the corresponding check box. If *Measure Forces* has been activated, a UDP-connection to the computer will be established, which interconnects to the amplifiers of the sensor gauges by CAN bus. Now, all forces will be redirected to the control computer for further processing, e.g. writing them to the data base. In addition, if the check box *Force Feedback* is activated, forces will be fed back to the user. In contrast to all other check boxes of this widget, this one will still be active after pressing *Start*. Therefore, force feedback can be switched on and off during operations. This feature was pre-

dominantly implemented for evaluation reasons (cf. chapter 4). If the data should be recorded for later processing, users can provide a name for the data set, which is going to be created, and activate the corresponding check box *Write Data*. After all settings are arranged, the system is switched to online mode by pressing *Start*. In a nutshell, the following cycle of operations describes the online mode of the system:

1. Determine all necessary parameters of the system (or select a preassembled program)
2. Press the *Start* button in the GUI: The real-time interface of all robots, which have been selected by the program, will be started and moved into the situs. The posture of the instruments is set to an initial state in order to synchronize them with the PHANToMs in the next step.
3. Press the *Start / Stop* foot switch in order to start the PHANToMs: The PHANToMs will be started and connected to the manipulators. After this point, movements of the PHANToMs will be mapped onto movements of the end effectors. Now, the surgeon can perform operations or interact with the system (e.g. switch between different instruments).
4. Press the *Start / Stop* foot switch again: Connections between input devices and manipulators will be interrupted, i.e. the surgeon cannot move the instruments any longer. Afterwards, the instruments are retracted from the situs and ready for restarting the cycle.

During operation, the current trajectory can be annotated for further evaluation. This is done by the option box *Current Activity*, which provides different choices of comments, which will be mapped to an integer and saved to the database.

3.4.2 Snap Mode

Since the master console provides only two input devices, but up to four manipulators can be controlled in our system, we have to provide a possibility to switch between them. One option is “indexing” as it is implemented by the *daVinci* system: the user can actuate a clutch to disconnect the control devices from the manipulators (cf. fig. 3.20 Nr. 2). Afterwards, the control devices can be relocated and

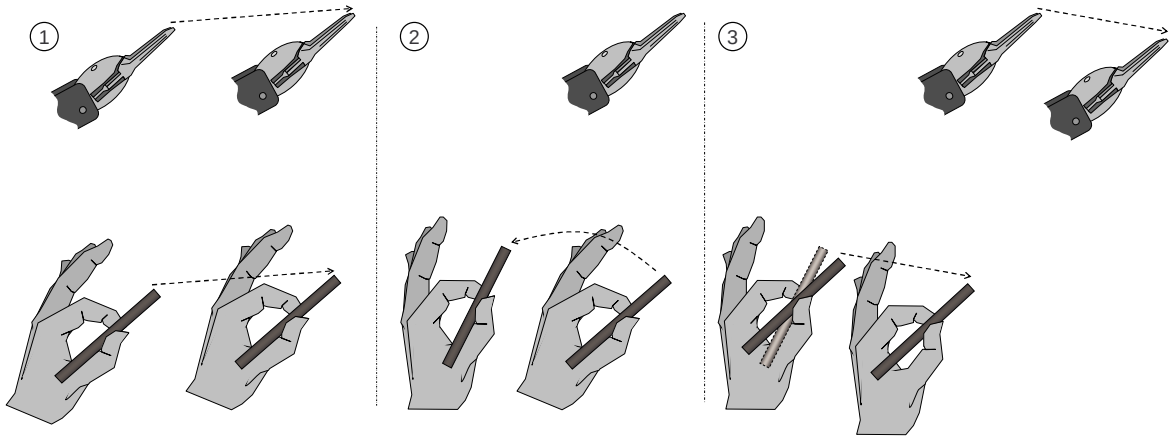


Figure 3.20: Illustration of the indexing procedure

clutched in again, possibly to control a different manipulator. In this moment the postures of the stylus and the end effector intended for action are synchronized in order to maintain a reasonable hand-eye

coordination (cf. fig. 3.20 Nr. 3). In order to accomplish this, the input devices have to resemble the posture of the end effector (transition from grayed-out posture to target posture in fig. 3.20 Nr. 3). This can be achieved, only if the input devices can be controlled actively, like those of the *daVinci* (or if a PHANToM with 6DoF force feedback is employed). Since the input devices of our system lack this ability (they are only capable of 3DoF translational feedback), we have to find another way to synchronize them with the instruments. Therefore, we have introduced the concept of a “snap mode”. This means the user has to move the stylus actively in order to synchronize it with the end effector. It is clutched in automatically as recently as the difference of both transformations (input stylus and instrument) falls below a certain threshold. This difference is determined as the rotation about the equivalent angle axis. Since it is difficult for the user to estimate the rotational difference between the stylus and the end effector, we have evaluated different options of providing an optical cue. One is to present both the stylus and the tip of the instrument in the simulation environment. The disadvantage of this procedure is the need of looking at another monitor, which disturbs the feeling of immersion. In order to compensate for that, we have tried to overlay the 3D view of the scene (including the end effector) with an appropriately placed model of the stylus. Due to distortions of the endoscope and calibration issues, this has not profen convenient. Therefore, we have implemented a simplified version of the procedure: Once the surgeon intends to change the manipulator, declutching can be performed by a foot switch. After a second actuation of the switch, the manipulator, which is going to be engaged, returns to an initial posture that can be easily resembled with the stylus. Now, the surgeon can place the input device in its initial position and clutch in again. During this process, the PHANToM devices are reset, therefore, providing an optimal synchronization with the rotation of the end effector, whose posture has also been reset. We have already evaluated this method in realistic experiments with our system (cf. section 4.2).

3.4.3 Simulation Environment

Apart from the manual interface described above, our system also comprises an interface for offline and real-time trajectory planning. The central part of this tool is a virtual emulation of the system where the user can easily manipulate its state. In fig. 3.21 a robotic arm of the system is selected. Items in the scene can either be selected by directly clicking on them, or by choosing them from the scene browser at the upper right side of the GUI. The scene browser can be used as a basic CAD program. It is possible to insert new basic geometries (like cones, spheres, cubes etc.) or VRML objects. If an item is selected in the scene browser, a context menu of the corresponding object is displayed. In fig. 3.21 a context menu for adjusting the different joints of the robot is displayed. In addition, each context menu of an object contains functionality to translate or rotate the object. By means of the scene browser it is also possible to aggregate objects to groups, which can be manipulated on their part. Objects can be moved in the hierarchy of the scene graph or removed completely. Also copy and paste operations on objects are provided in order to reuse preassembled parts. The scene graph, or parts of it, can be stored to disk in order to archive them safely. This provides an intuitive interface for users to manage different scenes and apply defined modifications. All operations on the scene graph are implemented by means of the open source API *Coin3D* from *Systems in Motion AS*. This is a high-level graphics language based on OpenGL.

The GUI provides different modes for interacting with robots or surgical instruments. As mentioned above, the robots can be moved by means of sliders in the context menu: one slider for each joint of the robot. In addition, the robot’s flange can be moved in Cartesian space (i.e. linear translations in x,y and

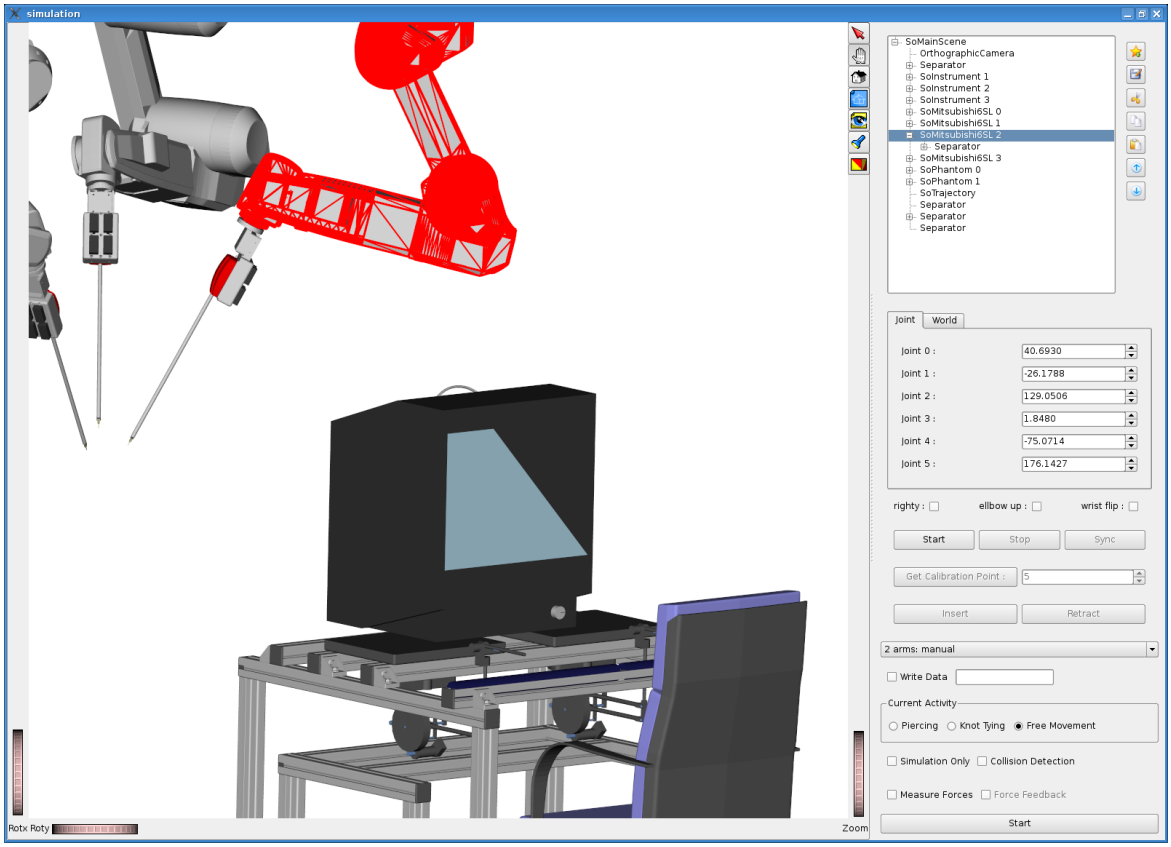


Figure 3.21: Graphical user interface (GUI)

z direction and corresponding rotations about these axes). Every time the configuration of the robot is changed, Cartesian movements will be mapped onto joint angles by the inverse kinematics. The same applies to the minimal invasive instruments, which require a special kind of inverse kinematics if moved in Cartesian space (so-called port or trocar kinematics [17]). As explained in section 3.3.4, trocar kinematics arranges for movements of the instruments about small incisions in the patient's body and it is indispensable for robotic applications in minimally invasive surgery. Since each instrument is linked to a dedicated robot, any input changing the position of the instrument's base will consequently induce corresponding movements of the robot to which the instrument is attached. During manual operations in online mode, all movements of the input devices and end effectors are additionally displayed in the simulation environment.

3.4.4 Trajectory Planning with Key-frames

So far, we can use this interface to move the robots or instruments from one posture to another. This can either be performed in real-time or offline. In real-time mode, the robot directly follows the movements, which are instructed by the GUI sliders. Since this is quite dangerous (particularly in Cartesian mode: small slider changes can result in wide-ranging robot movements), we have disabled this feature during normal operation. In contrast, offline operation provides more safety. After adjusting the posture of the robot by the sliders in offline mode, the robot will not move until the user has acknowledged the new stage. For more sophisticated trajectories, like they may occur in robotic knot-tying, this

kind of interface for point-to-point movements will not suffice. Therefore, we have developed a planning interface based on key-framing. This functionality can be accessed by selecting the “Trajectory” item in the scene browser and switching to tab “Animation” in the corresponding context menu (cf. fig. 3.27 left side).

key-framing interface

If we speak of key-framing regarding trajectory planning, we refer to moving the robots to certain consecutive positions, which are saved as key-frames. Afterwards, we apply a certain policy (e.g. linear or spline interpolation) to generate all other points of the trajectory between those key-frames (cf. fig. 3.22). There are two different modes of moving from key-frame to key-frame: one is to stop at

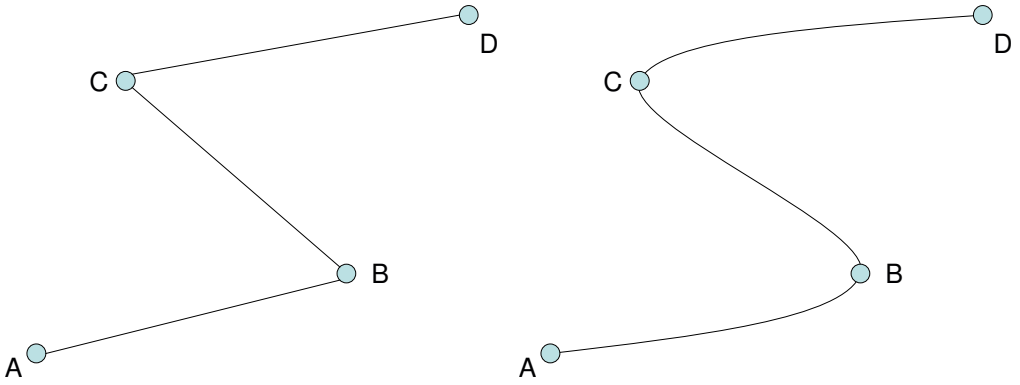


Figure 3.22: Linear and spline interpolation between key-frames

each key-frame, the other, more complex possibility is to make a continuous movement through all key-frames between start and end. Being the most intricate option, we restrict ourselves to the description of continuous movements via spline interpolation. We have to take into account that every robot needs a certain time for acceleration after starting, and for deceleration before it can be stopped. Otherwise, it is not possible to achieve jerk-free motions. Another prerequisite for our application is key-frames occurring at certain points in time, which have to be met exactly: i.e. if the robot starts in point A (cf. fig. 3.22) it will first accelerate to a certain speed, which depends on the time when point B has to be reached. Accordingly, there is a fixed time to move from B to C. Therefore, the robot will have to adapt its speed after leaving point B. For calculating the speed during acceleration and decelerations, we have employed the functions $v_a(t)$ and $v_d(t)$, respectively:

$$v_a(t) = \frac{v_c}{1 + e^{n\left(1 - \frac{2t}{t_a}\right)}} \qquad v_d(t) = \frac{v_c}{1 + e^{-n\left(1 - \frac{2t}{t_d}\right)}} \tag{3.8}$$

Those are sigmoid functions shifted along the positive t-axis. The factor n adjusts the acclivity of

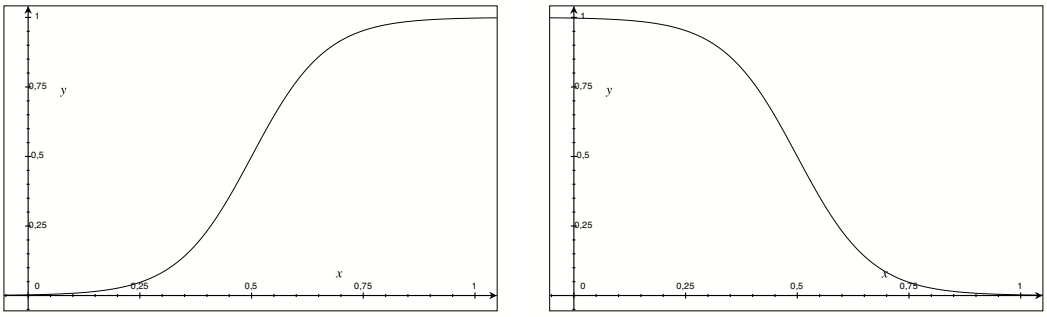


Figure 3.23: Sigmoid functions for acceleration and deceleration

the curve, which reaches its maximum at $t = \frac{t_a}{2}$ (cf. fig. 3.23). The time needed for acceleration and deceleration is denoted as t_a and t_d, respectively. Another nice feature of these functions is that the area underneath the curve (i.e. the traveled path) amounts to $\frac{1}{2}t_a v_c$. Therefore, given a certain path length and frame time, we can easily determine the residual speed v_c (the same holds for deceleration). Determining the path length is easy for linear key-frame interpolations, but linear interpolation yields the adverse effect of trajectories exhibiting discontinuities at the key-frame positions (cf. fig. 3.22 left side). In order to overcome this disadvantage, it is favorable to apply a spline interpolation to these points (cf. fig. 3.22 right side). We have chosen Hermite splines for interpolation. They arrange for smooth transitions at intermediate key-frames. Given a starting point $\overrightarrow{a}$ and an end point $\overrightarrow{b}$, and the corresponding tangent vectors $\overrightarrow{a}'$ and $\overrightarrow{b}'$, each intermediate point can be determined by

$$p(s) = h_1(s) \cdot \overrightarrow{a} + h_2(s) \cdot \overrightarrow{b} + h_3(s) \cdot \overrightarrow{a}' + h_4(s) \cdot \overrightarrow{b}' \tag{3.9}$$

where the Hermite weightings are calculated as follows:

$$h_1(s) = 2s^3 - 3s^2 + 1 \tag{3.10}$$
$$h_2(s) = -2s^3 + 3s^2 \tag{3.11}$$
$$h_3(s) = s^3 - 2s^2 + s \tag{3.12}$$
$$h_4(s) = s^3 - s^2 \tag{3.13}$$

The progression of this function is depicted in fig. 3.24. However, determination of the path length of

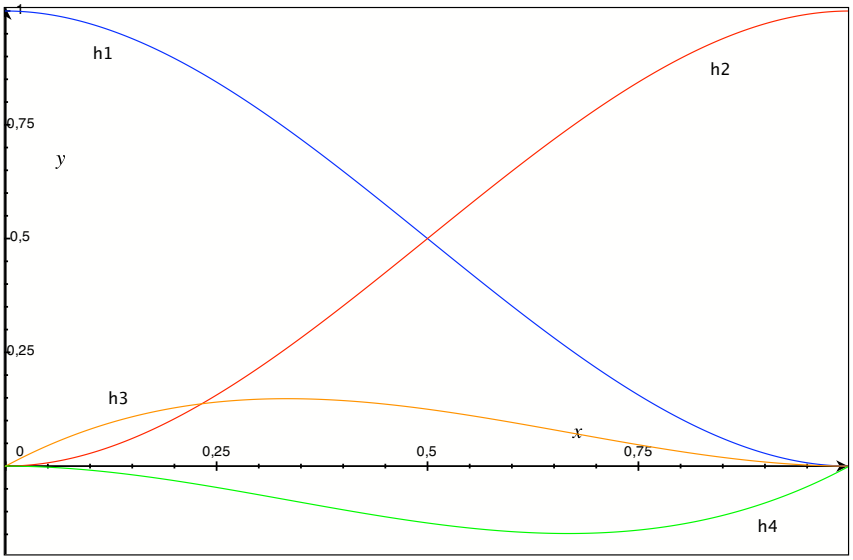


Figure 3.24: Weightings for Hermite splines

a spline curve is analytically not feasible. The only possibility is piecewise discretization, but this will cause inevitable aberrations. Therefore, we decided to use another method, so-called Hermite non-stop interpolation based on TCB-Splines [125]. The basic idea is to interpolate between the tangent vectors and use their length as manipulator speed. Thus, the length of $\overrightarrow{a}'$ is the initial speed of a section while the length of $\overrightarrow{b}'$ is the terminal velocity. Intermediate velocities can be determined by

differentiating equation 3.9:

$$v(s) = \dot{h}_1(s) \cdot \overrightarrow{a} + \dot{h}_2(s) \cdot \overrightarrow{b} + \dot{h}_3(s) \cdot \overrightarrow{a}' + \dot{h}_4(s) \cdot \overrightarrow{b}' \tag{3.14}$$

where the derivatives of the Hermite weightings with respect to s are determined as:

$$\dot{h}_1(s) = 6s^2 - 6s \tag{3.15}$$
$$\dot{h}_2(s) = -6s^2 + 6s \tag{3.16}$$
$$\dot{h}_3(s) = 3s^2 - 4s + 1 \tag{3.17}$$
$$\dot{h}_4(s) = 3s^2 - 2s \tag{3.18}$$

Using these formulas, it is guaranteed that the next key-frame will be reached at the right time and with the right velocity: i.e. the important parameters for robot movements will be automatically determined by the system. For the user, the only thing left to do is setting the key-frames on a timeline. This can be achieved by the interface depicted in fig. 3.25. By means of this interface, the user can add and

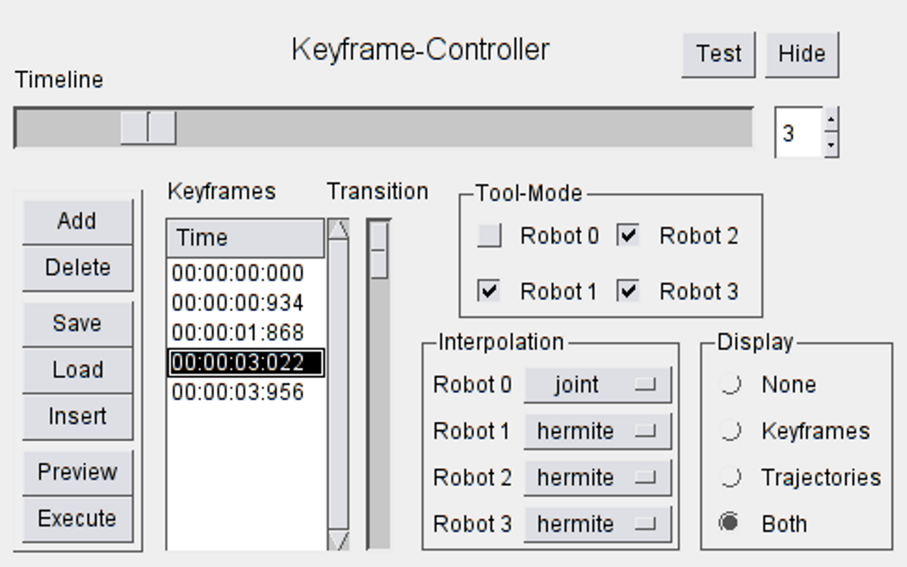


Figure 3.25: GUI elements for key-frame adjustment

delete key-frames on a timeline. The time for each key-frame can still be altered afterwards with the timeline slider. The user can select, which robot should be integrated into the program, and how the corresponding trajectory will be interpolated. In addition, trajectories can be stored and saved, and it is even possible to insert stored parts of trajectories into new ones. All features can be displayed in the simulation environment and a preview of the assembled trajectories is possible. Anyway, two bad things still can happen. The first is a robot exceeding its speed limit because the user has scheduled too little time for its movement. In this case, the user gets an error message, which recommends to extend the time between the corresponding key-frames. The other issue is that instruments or robots may collide if the planned trajectory is executed. In order to avoid this, all trajectories are processed by the inverse kinematics and collision detection unit (see above). If collisions occur, it is not possible to execute the trajectory, unless it is safely replanned.

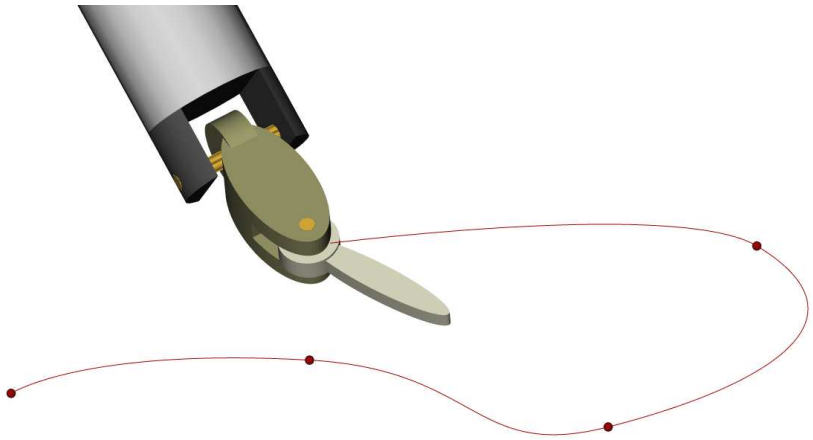


Figure 3.26: Display of key-frames and calculated trajectory in the planning environment

3.4.5 Learning Interface for Human-Machine Skill Transfer

The interface to the skill transfer functionality is realized as a context menu, which can be activated by selecting the item "*Trajectory*" in the scene browser of the simulation environment (cf. section 3.4.3). The corresponding menu provides access to the `MySQL` data base, which contains all recorded trajectories. After selecting a trajectory in the table view, a spline representation will be displayed in the 3D view of the simulation environment (if the corresponding check box is marked). The selected trajectory can be executed by pressing the *Replay* button. This implements a "Teach-In" functionality, which has already been realized in the previous version of the system, EndoPAR. If a trajectory is intended as demonstration of an abstract skill, it can be checked for validity and decomposed into primitives by pressing the *Get Primitives* button. Once this button was pressed, the selected trajectory

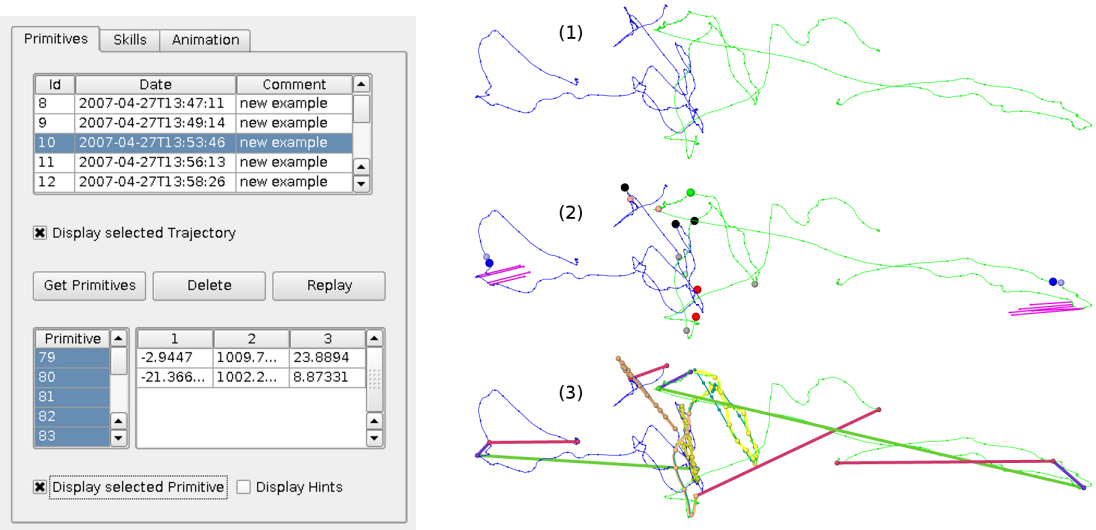


Figure 3.27: Trajectory GUI (left) and different steps of the algorithm (right)

is examined for being an instance of the currently selected skill. If this validation yields a positive result,

the trajectory is decomposed into primitives by means of abstract pattern matching. The underlying procedure is explained in detail in section 5. If any primitives are detected in the currently selected trajectory, a second table view is displayed in the GUI (cf. fig. 3.27 left side). This view is capable of multiple selections, i.e. one or more primitives will be displayed in the 3D view, if the corresponding check box is marked. If at least one primitive is selected, its essential features are displayed to the right of the primitive list. In case of multiple selections, the features of the primitive with the lowest ID are displayed. In addition, it is possible to display all detected events, which have been used for validation and decomposition, if "Display Hints" is selected. The right side of figure 3.27 visualizes the consecutive steps of the employed algorithm:

1. Raw trajectory (user demonstration selected for further processing).
2. The same trajectory augmented with detected events.
3. All primitives of this trajectory are selected for display.

3.5 Software Engineering Aspects

We want to conclude this chapter by presenting some software engineering aspects of our control architecture. First of all, we have tried to design the software to be independent from a specific operation system. Currently, our system is based on a Linux 64bit platform, but it may be compiled for other platforms without major changes as well. All modules, including the HAL and all superordinate parts, are written in plain C++ using only standard extensions like STL, which are available for most platforms. The code contains only types, which are independent from a specific word-length, and therefore, will run on both 32bit and 64bit systems. We have experienced that our software profits from running on a 64bit system, since the accumulated errors within the inverse kinematics are significantly reduced. All libraries used in our software are available as public domain source code and may be compiled for various operating systems. In detail, we use *MySQL* as database, *Qt* for constructing the GUI, *Coin3D* for visualizing the 3D models and *GNU GSL* provides some of the scientific functions like SVD (cf. chapter 5). The only parts of the software, which are platform dependent, are the drivers for the PHANToM devices and the CAN-bus interface. However, both are also available for other platforms.

software engineering

In order to guarantee scalability and quick response times, we have based the architecture of our software on multi-threading. Thus, all important control loops of the software are implemented as threads (platform independent *POSIX* threads). Every manipulator added to the scene has to provide its own control thread, which interacts with the corresponding robots and instruments. These threads are set to real-time priority in order to guarantee accurately timed transmission of joint angles. They are scheduled to be executed every $3ms$. Therefore, new joint angles arrive at the robot at least once in a control cycle of $6.8ms$. If more than one set of joints arrives during a cycle, additional values are simply omitted by the controller and can cause no unexpected behavior. The same strategy applies to the control threads of the PHANToM devices, which arrange for proper acquisition of postures and realistic force feedback. Although being implemented as real-time threads, they are set to a lower priority as the control threads. This is due to the fact that a violation of the timeliness of the control threads will lead to a system crash, while a reduced timeliness of the PHANToM threads will only lead to minor jerks regarding the movement of the controlled robots and the force feedback. The thread for interactions with the user interface is set to non-real-time priority, since no interactions are required during manual operation of the system (with all other threads activated) and a timely display of the 3D

models is not critical. Since the threads have to interchange data with each other (e.g. the PHANToM threads provide data for the control threads of the robots, or the collision detection must be able to stop all robot threads), we have implemented a central class called `ControlUnit` providing shared access. In addition, this class also provides an interface to the MySQL data base in order to store trajectories and other features. Due to the one-way flow of data, mutual exclusion of data access is not necessary. This leads to a significant increase of speed and intrinsically avoids starvation of threads.

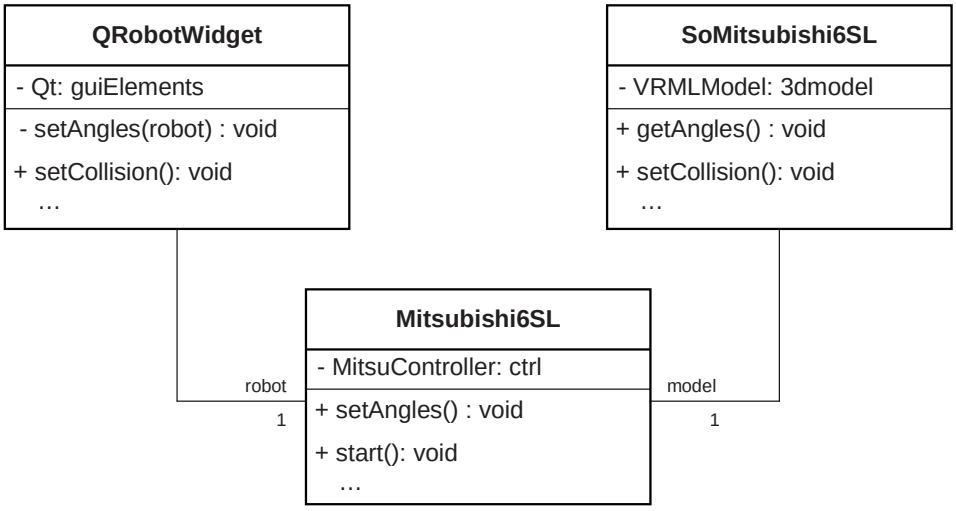


Figure 3.28: Model-View-Controller (MVC) architecture of the robot interface

The architecture of our software is mostly based on the model-view-controller design pattern. We will explain this concept on the basis of the classes, which have been implemented in order to interact with the *Mitsubishi* robots. The model is realized by the class `Mitsubishi6SL` and contains all functionality for on- and offline interaction with the robot (e.g. establish a connection to the controller, start servos, set joints ...). The view is implemented as a 3D model, which is going to be displayed in the simulation environment. This is the only part of the robot's interface depending on the *Coin3D* API. Therefore, the corresponding class, `SoMitsubishi6SL`, contains a VRML model of the *Mitsubishi MELFA 6SL*, functions for changing its angles and methods to visualize the collision detection. This class contains a pointer to the model (`Mitsubishi6SL`) in order to update the angles in the 3D view. In contrast, the model contains no pointers to the view class in order to allow for implementations of the control software without 3D graphics, or even without any GUI. Thus, the model also contains no pointer to the controller class, `QRobotWidget`. The capital "Q" indicates that this (and only this) part of the software was implemented with *Qt*, a widespread API for implementation of GUIs. This class contains the context menu for robot interaction (see above), thus, providing sliders for adjustment of angles, Cartesian control, or movements in tool frame. In addition, the menu exhibits some buttons to activate functions of the model (e.g. starting and stopping the servos of the robot). The `QRobotWidget`-class contains no direct pointer to the model (`Mitsubishi6SL`), but a pointer to an abstract class. Therefore, it is easy to exchange the robot model without greater modifications.
Once an element of the GUI, implemented in `QRobotWidget`, was activated, the corresponding event is dispatched to the model. For example, if the user intends to change an angle of the robot by dragging a slider, the `setAngles()`-function of the corresponding object of type `Mitsubishi6SL` is called (cf. fig. 3.29). In our implementation, no direct connection between the controller and the view is established.

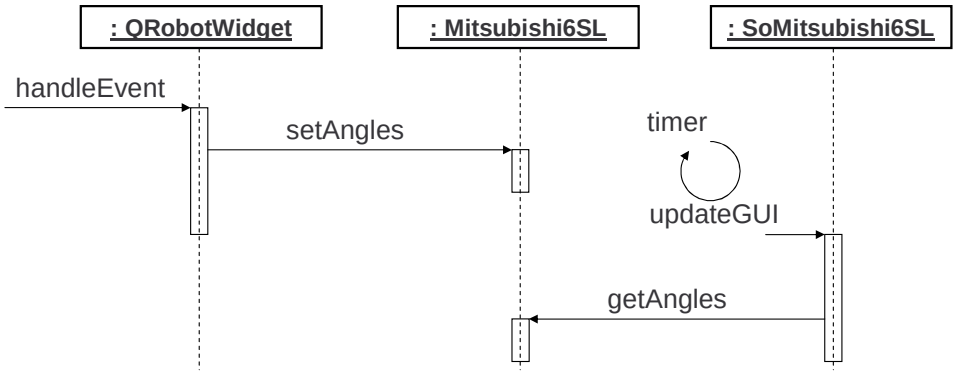


Figure 3.29: Sequence diagram of changing an angle

Instead, a central GUI class, `QSimulation`, which contains the context menus as components, handles the updates by periodically calling the `updateGUI()`-function of all views. After this function is called for `SoMitsubishi6SL`, this view fetches the angles of the robot by calling `getAngles()` of the model. Now, the angles of the 3D model can be displayed appropriately.
As mentioned above, `QRobotWidget` contains only a pointer to an abstract robot class, which can be instantiated by various types of robots. While the robot-specific parts (e.g. low-level communication with the controller) are hidden within these classes, they have to provide standardized functions for interaction with certain GUI elements. For example, every class implementing `Robot` has to provide functions for starting and stopping the servos, and for the adjustment of angles (cf. fig. 3.30). A similar concept is used for the classes implementing the view of a robot. They are all derived from `SoRobot`.

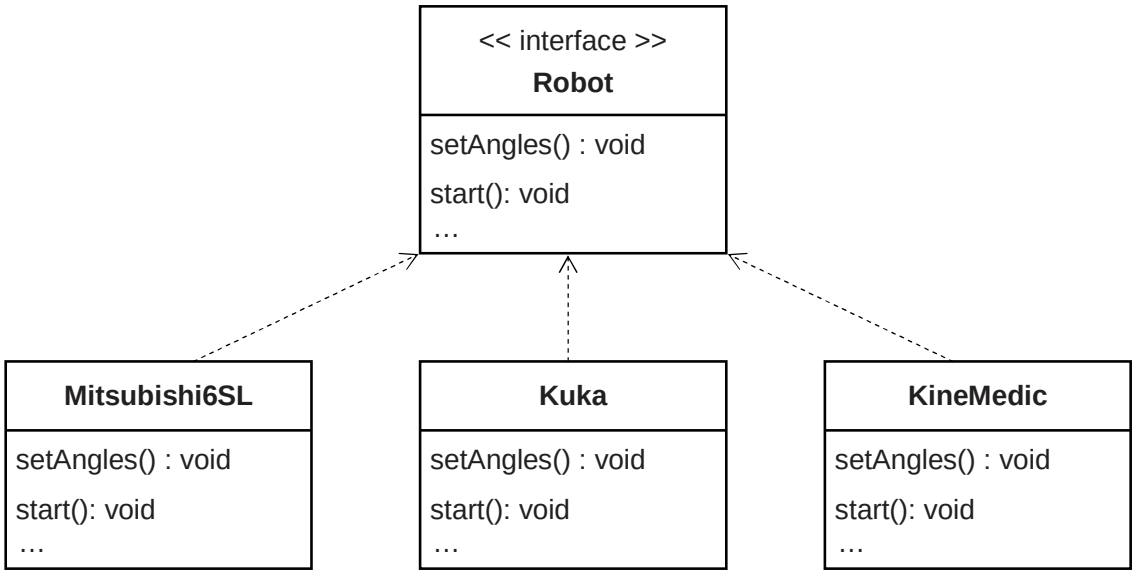


Figure 3.30: Different models of robots are implementations of the same abstract class

All these concepts, MVC and abstract classes, are used for most of the objects in our system. The MVC classes of the *PHANToM* device are called `SoPhantom`, `Phantom` and `QPhantomWidget`, while those for the instruments are named `SoInstrument`, `Instrument` and `QInstrumentWidget` etc. The framework is completed by some aggregation classes: As mentioned above, the class `ControlUnit` contains all

existing models of the MVC architecture in order to serve as central class for data storage. In addition, interaction sequences, comprising different models, are also implemented in this class. The views of the elements are basically *Coin3D*-nodes, which are inserted into a central tree structure whose parent node is `SoMainScene`. Furthermore, `QSimulation` aggregates all context menus, i.e. all controllers of the MVC architecture, and some additional interfaces, e.g. the window for displaying the scene graph. Finally, everything is put together by the main routine in `simulation.cpp`. Here, the instantiation of all classes is initiated, the scene graph is initialized, the widgets and windows for visualization are constructed, and the main loop of the controller is activated.

Chapter 4

Applications

By means of the described setup, we have already performed automated procedures like knot-tying [28, 4]. Focus of this section is on the evaluation of the impact of force measurement and feedback. Our hypothesis, which has been proven correct [36, 33, 1], was that haptic feedback contributes to a better performance of systems for robotic surgery by preventing force-induced damages. Examples for such harms are the breakage of thread material, tissue fissures and the strangulation of sutures. A similar, more general hypothesis was recently proposed by Gersem et al. [92], but they conducted no evaluation in order to proof their thesis. Recently, Wagner and Howe [49] conducted experiments on the influence of force feedback on surgical work and they came to similar conclusions like we did.

4.1 Evaluation of Force Feedback (EndoPAR)

The major goal of any telepresent application is providing as much immersion as possible: i.e. in an optimal telepresent scenario the operator should not be able to tell whether it is virtual or not, since all senses are supplied with adequate sensory substitutions. Currently, relevant systems are still far from reaching this optimum. While high-quality optical substitution is provided by sophisticated HMD systems yet, haptic substitution of the actual environment is still in an early stage of development. The haptic sense can be further divided into two categories: force feedback and tactile feedback. In higher animals, the former is realized with touch-sensitive receptors under the skin, while the latter is provided by proprio-receptors in muscles and joints. In this section we focus on force feedback in telepresent systems, while neglecting tactile feedback. Currently, commercially available systems like the *daVinci* manipulator (see chapter 2) do not provide any force feedback. There is an active discussion in the corresponding literature, regarding the influence of force feedback on telemanipulated surgery [63, 81, 105, 139]. Therefore, the hypothesis that force feedback in form of sensory substitution facilitates performance of surgical tasks, was evaluated on the experimental platform described above (on the EndoPAR version). In addition, a further hypothesis was explored: The eminently high fatigue of surgeons during and after robotic operations may be caused by visual compensation of missing force feedback [205].

first evaluation

4.1.1 Materials and Methods

Subjects of this study have been 25 heart surgeons of different levels of surgical training and age (cf. fig. 4.1). We have divided them into three groups regarding their skill level: One group of eight young surgeons, another group of twelve experienced surgeons and a third group of five surgeons with robotic experience. All subjects had to perform typical surgical procedures: surgical knot-tying,

Table 4.1: Random sample of the human subjects

	all participants n=25	young surgeons n=8	experienced surgeons n=12	robotic surgeons n=5
surgical experience (years) mean value standard deviation	7.4 (±6.9)	0.4 (±0.3)	11.5 (±7.6)	9.0 (±5.2)
age (years) mean value standard deviation	36 (±8)	29 (±7)	39 (±8)	39 (±5)
robotic experience (years) mean value standard deviation	0	0	0	2.4 (±2.0)

breakage of suture material and detection of arteriosclerosis. In the following sections the consecutive execution of these procedures will be called task sequence. All task sequences have been repeated at three different levels of force feedback: absence of feedback, feedback of actually measured forces and amplified force feedback (factor two). Each surgeon had to perform three subsequent task sequences. The corresponding level of force feedback of a certain task sequence has been chosen randomly. In detail, each task sequence consisted of the following three subtasks (cf. fig. 4.1):

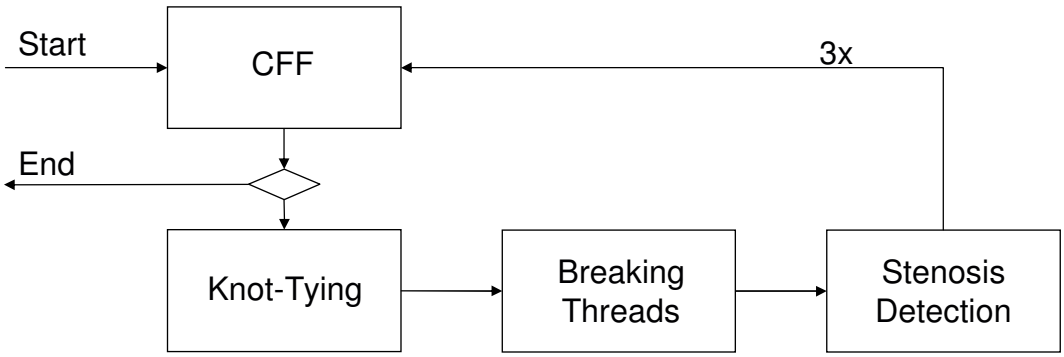
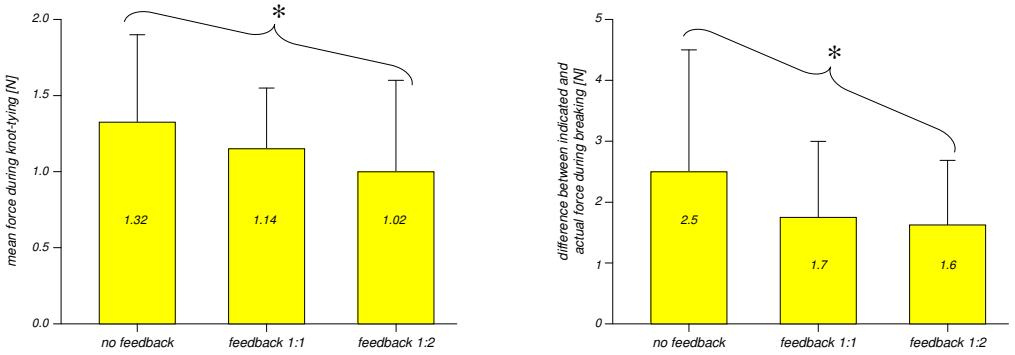


Figure 4.1: Course of evaluation for each single subject

- Surgical knot-tying:
  Surgical knots have been tied with two minimally invasive instruments as described above (see chapter 1). Within ten minutes, the surgeons had to perform as much knots as possible in alternating arrangement (winded clockwise and anti-clockwise, respectively). The total number of knots, their quality, the applied forces, the breakage of suture material and the speed of motion during knot-tying have been recorded for subsequent analysis.

- Breakage of suture material:
  For this experiment, a surgical thread (ETHICON Prolene^{TM}6-0) was clamped in between two instruments. The surgeons had to strain the thread and pull until it breaks. They were asked to indicate the point of imminent breakage. At this moment the marginal forces have been recorded and were compared to the actual breaking force. Cases where the thread was broken before indication have been neglected, since the corresponding incidences were equally distributed across all force levels.

- Detection of arteriosclerosis:
  The subjects had to detect a possible stenosis with one force feedback instrument (dominant hand). Stenoses have been emulated by metal wires of different lengths, which have been inserted in artificial arteries made from polymer. The surgeons had to indicate whether there is any object inside the tube, and if so, how long it was. This detection task had to be completed precisely and also expeditiously. Detection errors, regarding existence or size of the stenosis, have been counted. In addition, the applied forces during detection have been recorded and the time needed for detection in seconds, respectively.

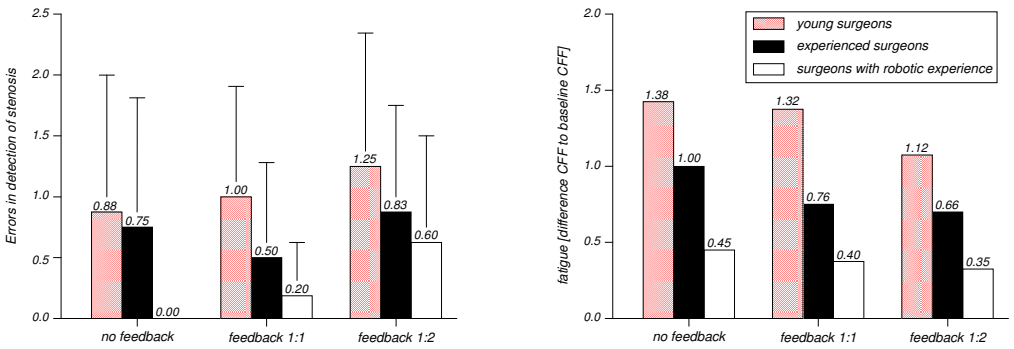


(a) Forces during knot-tying

(b) Forces during breakage of suture material

Figure 4.2: Statistical plots of evaluation of force feedback (the asterisks indicate that significance has been proven) [*original data recorded by H. Mayer, statistical evaluation by C. Haßelbeck and E.U. Braun*]

At the beginning of the experiment and after each sequence, subjects have been tested for fatigue. As an indicator we have used the critical flicker fusion frequency (CFF). The CFF-device is an individual part of the “Wiener Testsystem” (Schuhfried GmbH, Mödling, Austria) which was designed for analysing the progression of fatigue [218]. The subject is asked to look into a box where a LED is flickering. Above a certain frequency, the human visual system cannot discern a flickering light any more, but perceives a steady light. As a person gets more and more tired, this frequency decreases [113]. The CFF is defined as the median of fusion frequency, determined after five presentations of a flickering red LED at increasing and decreasing frequency, respectively.



(a) Detection of arteriosclerosis

(b) Progression of surgeon's fatigue

Figure 4.3: Comparison of surgeons with different skill levels [*original data recorded by H. Mayer, statistical evaluation by C. Haßelbeck and E.U. Braun*]

4.1.2 Results of the first evaluation

The evaluation data has been analyzed by E.U. Braun during work on her master thesis and by C. Haßelbeck during work on her MD thesis. The corresponding results, which are presented here for the sake of completeness, have already been presented at various international conferences [35, 6, 32, 12, 34, 5, 9, 11]. Increasing the amplification of force feedback during knot-tying leads to a significant reduction of the applied forces ($p < 0,05$ cf. fig 4.2(a)), while there was no significant change in the performance time. Some of the subjects also asserted that they feel less comfortable in performing this task under force feedback. This is most probably due to stress-strain forces induced by the driving wires. However, there have been no such disturbances during the remaining tasks, since they did not require extensive movements of the grippers. During the breakage of suture material, the corresponding differences of forces have been calculated as described above. Increasing the magnitude of force feedback leads to a decrease of this difference ($p < 0,05$ cf. fig. 4.2(b)). This militates for the precision of the estimated force when the thread was breaking and the high grade of immersion the telemanipulator system provides. Similar results have been found for the detection of arteriosclerosis. Haptic feedback significantly influences the amount of applied forces: an increase in the amplification of force feedback leads to a significant decrease of applied forces. However, force feedback has no significant influence on detection errors ($p = 0,05$ cf. fig. 4.3(a)) and performance time. This is most probably due to optical cues, which play a predominant role in this detection task. Similar results have been indicated by Tonet et al. [47]. Finally, regarding fatigue, we came to the conclusion that operating with force feedback leads to a significant decrease of that phenomenon ($p < 0,05$ cf. fig. 4.3(b)). In order to check for significance, a Friedman test has been applied, since the data exhibits no Gaussian distribution. In case of significance, a Wilcoxon test has been applied for pairwise comparison.

4.2 Evaluation of Operation Tasks

The evaluation mentioned above was carried out with the first version of the system (EndoPAR). Since the reliability and precision of this system is limited and allows only for performance of simple tasks, we have undertaken a second evaluation with the new version of the system (ARAMIS). This time we

second evaluation

performed more complex tasks, which are more in step with surgical practice.

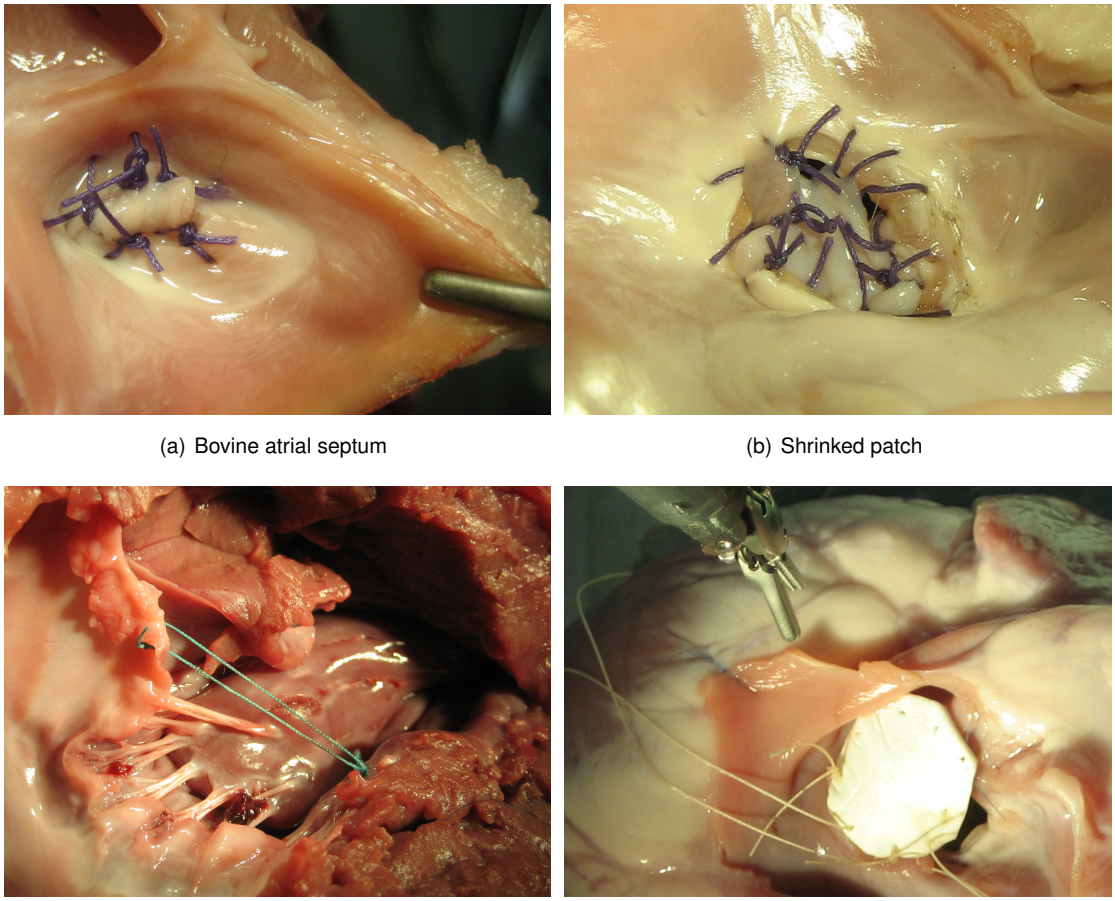
4.2.1 Materials and Methods

Before starting the evaluation with surgeons, we have assessed different operation tasks, which are typically applied in heart surgery, for their feasibility with our system. The demand on the experiments was to keep them below a total execution time of two hours and to perform them with a fixed camera view. Although the employment of a maneuverable camera would have been possible, we refrained from this feature since switching between instruments and camera is still very time-consuming and probably confuses unexperienced users.

We have performed first trials with human cadaver tissue, but this turned out to be inappropriate. This was mostly due to the formalin (which is used as preserving agent) having changed the elasticity of the tissue. Even with self-cutting needles it was very complicated and time-consuming to pierce the tissue, which was cured this way. In addition, handling human cadaver tissue involves more effort on acquisition and disposal. Therefore, we have conducted all subsequent experiments on animal hearts (cf. fig. 4.4): We have tried to close an atrial septum defect (ASD) of a bovine heart. We have prepared the heart with an artificial defect and closed it with a patch made out of the atrioventricular valves of the same heart (cf. fig. 4.4(a)). During several performances of this experiment, the patch turned out to shrink once exerted to warm air and bright lights (cf. fig. 4.4(b)). Therefore, we have tried to use patches made of GoreTex^{TM} material 4.4(d). While this patch did not shrink, it was very tough, and therefore, hard to pierce with minimally invasive instruments. Also, the tissue of the bovine heart was very tough, which has made this intervention very time-consuming.

Another experiment, which we have adopted from heart surgery, was the replacement of a papillary tendon by a surrogate made of surgical silk (cf. fig. 4.4(c)). Since the papillary muscle, constituting the attachment point, is an elastic, convex structure, it was quite easy to pierce it with the needle. The same applies to the mitral valve, which had to be fixed by this procedure.

Thereafter, we have carried out the same experiments on pig hearts. Since those are smaller, working on them is more delicate, but size and tissue properties are more similar to human hearts. Also, the higher elasticity of the tissue has fostered our experiments. While the conductance of the replacement of the papillary tendon yielded no differences, it was much easier to perform the closure of the atrial septum defect on pig hearts. In fig. 4.5(a) the preparation of the heart is depicted: a small hole of approx. 0.5 cm in diameter is cut into the atrial septum of the heart. From one of the atrioventricular valves, a patch for closing this hole is dissected and prepared. Based on the experience with the bovine heart, we shaped the patch to cover more than twice the size of the hole in order to take shrinkage into account. For convenience reasons, we were working with three instruments to perform this intervention (cf. fig. 4.5(b)). Two grippers (the assistant and one of the active grippers) are used to strain the patch, while the other active gripper wields the needle in order to pierce the tissue. We have used six stitches to close the hole (cf. fig. 4.5(c) and 4.5(d)). Afterwards, the suture was tested for permeability. It turned out to be leakproof under flowing water, but it still was permeable under pressurized water.



(a) Bovine atrial septum

(b) Shrinked patch

(c) Surrogate papillary tendon

(d) Artificial patch

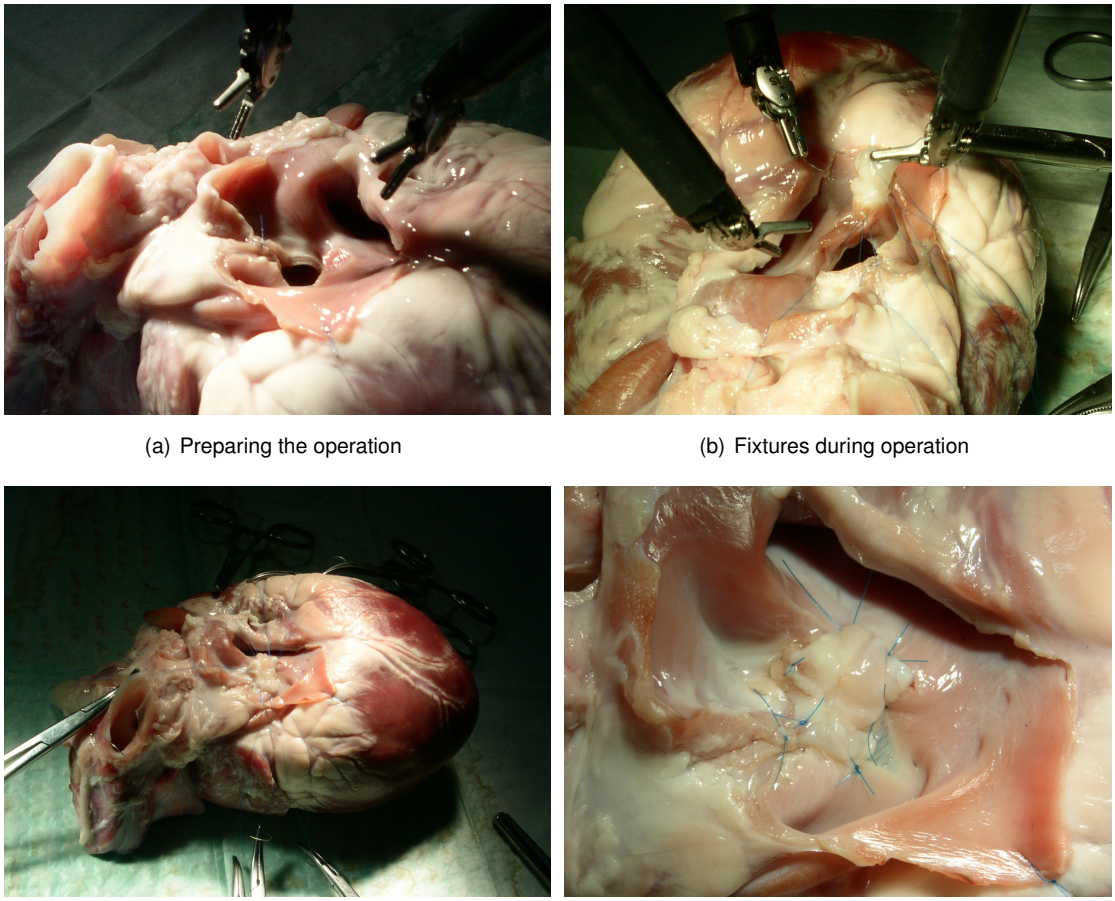
Figure 4.4: Different experiments of the planning stage

Since performance time of the ASD intervention still was too extensive, we have further simplified the procedure and finally we have chosen the following run of experiments for the evaluation:

1. **Close a small cut in the heart**: This initial exercise has no direct correspondence with cardiac surgical procedures. Its purpose was to familiarize the subjects with our system, before they proceeded with the ASD closure, which is comparably complex.

2. **Closure of a simplified atrial septum defect**: Since the original procedure turned out to be too time-consuming, we have simplified the intervention to the closure of a small cut in the atrial septum, which can be achieved by two stitches. Anyway, this experiment still was complicated to perform since the atrial septum constitutes a concave structure. In order to ease stitching, the assistant gripper can be used to strain the septum, which was lain open for this experiment.

3. **Replacement of papillary tendon**: This procedure already worked fine for the bovine heart. Therefore, we have added its performance on a pig heart to the list of experiments. Since the areas of operation do not intersect, we were able to reuse the same heart as for the closure of the ASD.



(a) Preparing the operation

(b) Fixtures during operation

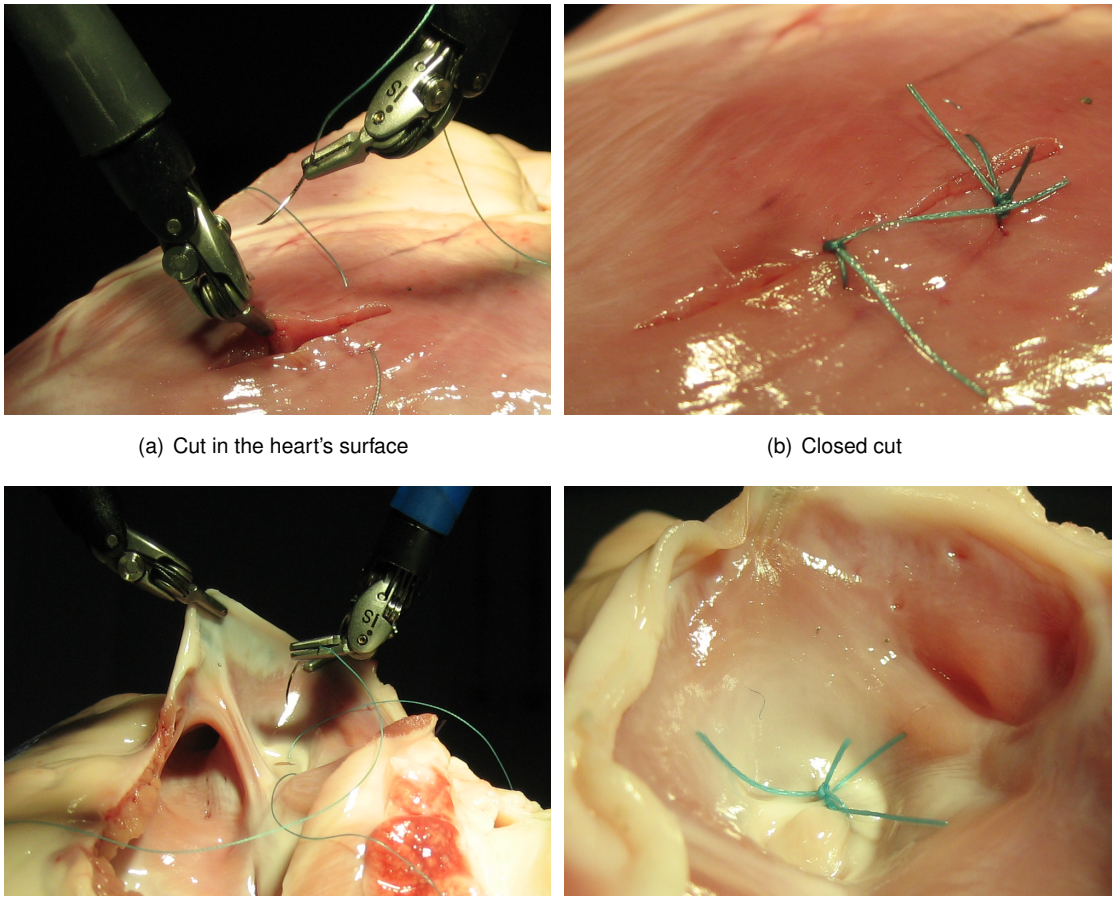
(c) Septum defect closed

(d) Closeup of the suture

Figure 4.5: Closure of an atrial septum defect (ASD)

4.2.2 Results

The experiments described above have been conducted by 30 subjects (15 students or residents and 15 surgeons). All of them were able to perform all experiments. There was a significant difference in performance times between both groups. While the surgeons needed between 45 min. and 110 min. (avg. 75 min.), the students and residents needed considerably more time (between 70 min. and 170 min. with an avg. of 105 min.). Although none of the surgeons has ever worked with our system, general surgical experience seems to play a significant role for the adaption to new surgical devices. This was not clear from the beginning, because one of our hypothesis was that the adaption to this teleoperated technique might be influenced by experience with computer games (particularly joystick-based games) and this kind of ability is more often found in young people than in experienced surgeons. We have also asked the subjects for their individual assessment of the force feedback. Most of them deemed it helpful. Interestingly, most surgeons with experience on other devices for robotic heart surgery (e.g. the *daVinci*), found the force feedback disturbing.



(a) Cut in the heart's surface

(b) Closed cut

(c) Simplified septum defect

(d) Simplified ASD closed

Figure 4.6: Experiments of the evaluation

Chapter 5

Skill Transfer

Originally, skill transfer refers to the teaching of new abilities and is usually restricted to humans. Human beings start their lives with a rather small “basic configuration” of inborn behaviors, restricted to some reflexes and evolutionary knowledge (e.g. human babies will stop at the edge of a table, without ever having made the experience of falling down). While most animals (especially presocial species) are fit for action shortly after birth, humans have to acquire most of their abilities in a long learning process during youth. If this process is omitted, humans will not be capable of normal participation in society and even suffer from physical deficiencies (so-called Kaspar-Hauser syndrome). Therefore, skill transfer is a basic key in human evolution. It is a powerful technique to pass knowledge cross generations and to generate culture and traditions. However, some animals are also capable of basic skill transfer (e.g. crows, apes or elephants), but this is mostly limited to skills for food acquisition [101].

Apart from an effective brain, speech and sophisticated sensori-motor abilities are the basic keys to skill transfer. Without speech, skill transfer will be restricted to learning by showing, which is quite cumbersome - i.e. without annotations, much more demonstrations of a skill will be required for generalization. On the other hand, without a sensori-motor system, learning is restricted to an abstract manipulation of symbols. Some authors even argue that intelligence cannot exist without a body [67] and learning of skills has to be situated in a real physical environment - i.e. teacher and learner are parts of the same environment and possess similar sensori-motor abilities. Later in this text we will seize this suggestion in order to develop a novel methodology for human-machine skill transfer. Before we do so, we review some other methods of learning and human-machine skill transfer, and assess their applicability on our scenario.

5.1 Learning Techniques for Robotic Systems

Since our scenario, automated knot-tying, mostly deals with the acquisition and reproduction of trajectories, we review a variety of approaches for learning of movement patterns. We will restrict ourselves to basic concepts, which pursue the goal of reproducing certain trajectories or fulfill well-defined tasks like the well-known peg-in-a-hole task. Higher level concepts like learning by demonstration and sensori-motor skill transfer, which require an in-depth analysis of the underlying task, will be introduced in a dedicated section. Methods, which are reviewed in this chapter, are basically independent from any specific task and can be applied to complete trajectories or on parts of them. Therefore, we will

start this section with techniques employed for the decomposition of trajectories.

5.1.1 Task Decomposition

A central paradigm of engineering is breaking up complex tasks into multiple subtasks, which are easier to solve, and reintegrate the results at the end. Therefore, robotic engineers developed the idea to decompose complex trajectories into smaller segments, which will serve as input for their methodologies. Though addressing the same concept, they have used different terminologies for naming these segments. In most publications these segments are called primitives, or more specific, motion primitives and visuo-motor primitives, respectively. Other authors have called them actions or even symbols. There exists a broad variety of methods for decomposing trajectories. In the following section we will introduce some of the established concepts.

In most applications, motion primitives are used to decompose complex user demonstrations, and therefore, these techniques are a first step towards learning by demonstration. The term “motion primitive” is not used consistently throughout literature. There are some authors referring to motion primitives in terms of a spatial decomposition of trajectories, while a prevailing part of authors mean a temporal segmentation. Examples for the former are Sanger [186] and Lim [136], who define primitives on the joint level of motions, and Xu et al. [222] who has decomposed complex trajectories into simpler primitives comprising the velocity profile of only one coordinate. A similar technique has been proposed by Slutski et al. [200]. They have introduced a method to analyze human pointing movements. Such decompositions of a trajectory can also be found in combination with temporal decompositions as it was proposed by the hierarchical learning model of Hasegawa et al. [100]. As mentioned above, a more common definition of motion primitives is exclusively based on a temporal decomposition of trajectories. A motion primitive is applied at a certain point in time after a certain state of the system has been determined [117]. Application of the action rules associated with this primitive will transfer the system into a desired final state. Therefore, primitive motions can be seen as transitions of a finite state machine, which enables the application of sophisticated learning techniques like Hidden Markov Models (see discussion below). Another method for employing primitives is to compose complex tasks out of different predefined primitives. This was employed by Morrow et al. [156, 155] in order to perform a peg-in-a-hole task, where primitives have been used as transitions between different contact states of the system. A similar method was recently proposed by Hwang et al. [107] in order to manipulate objects with only two fingers of a robotic hand. A related approach for multiple fingers was developed by Matsuoka et al. [145]

different types of motion primitives

While the authors referenced above restricted themselves to the application of primitive motions, there are much more publications about the derivation of primitives from user demonstrations. This is a first step towards learning by demonstration or imitation learning, which will be subject of the next sections. Approaches for primitive derivation differ from author to author. Some techniques require multiple user demonstrations in order to extract generalized primitives. This applies to the proposal of Volz [199], where similarities in trajectories acquired from user demonstrations are analyzed by means of singular value decomposition. Multiple modalities (geometry and contact states of the trajectory) are incorporated in the process. Anyway, experimental results are restricted to rather simple subtasks like force-guarded, linear movements along a fixed axis. These primitives were reassembled again to solve a peg-in-a-hole task, which was an important benchmark at that time. The same benchmark was sub-

ject of the research of Tominaga et al. [206] and Voyles et al. [210]. While these approaches are still based on predefined primitives, Newman et al. [168] tried to derive primitives from user demonstrations. However, this segmentation still was performed manually. An automated derivation of primitives was proposed by Skubic and Volz [198]. They have introduced the concept of so-called single-ended contact formations. These contact formations are characteristic for a certain primitive and can be classified by an algorithm for fuzzy cluster analysis. They have conducted experiments with force-based primitives like aligning faces of objects. Other authors have used geometric approaches to derive primitives automatically. Bentivegna et al. [61] and Mann et al. [142] have defined discontinuities in the trajectory as demarcation points of primitives. A related approach was proposed by Nakazawa et al. [166] and Inamura et al. [110]. Instead of direct reference to distinctive features in the shape of the trajectory, they have used discontinuities in the velocity profile. However, both authors desisted from giving any details about the workflow of their algorithms. Their proposals also differ regarding the classification of primitives. While the former uses an approach with dynamic programming to analyze the distance between classes of primitives, the latter employs singular value decomposition like Volz (see above). Some authors like Friedrich et al. [89] and Ogasawara et al. [170] have merged both steps, trajectory segmentation and primitive classification, into one single procedure. They segment user-demonstrated trajectories by means of a predefined library of primitives, which serves also as basis for classification. While both of them use contact information to classify the segments, Kimura et al. [122] have proposed an approach, which is based on a fuzzy filtering of the geometric information of the trajectory. A more abstract method was proposed by Nakamura et al. [165]. They have tried to emulate the sensor readings of user demonstrations with the help of combinations of previously acquired primitives.

5.1.2 Basic Techniques for Trajectory Learning

Our first approach was to learn the shape of a trajectory itself, without respect to any interactions with the environment so far; i.e. given an initial sequence of a trajectory, we want the system to provide a continuation based on a previously demonstrated example of this trajectory. Or in other words, the system should be able to predict the trajectory of the user and automatically continue the intended sequence. The complete sequence was demonstrated several times during the learning process. During this stage, the system has to generalize the trajectories in order to discern previously unknown instantiation of them. Although this challenge seems to be simple, it raises a bunch of demanding requirements for the underlying learning algorithm:

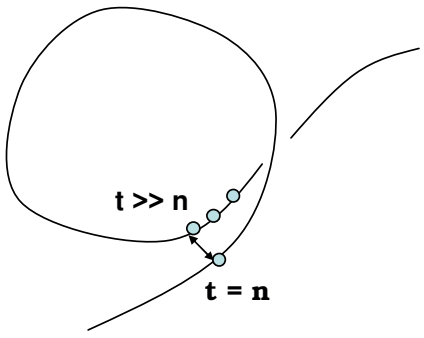


Figure 5.1: Long-term dependencies

1. **Learn long-term dependencies in the trajectory**: Any trajectory will be represented as a set of points occurring at a distinct point in time. This timestamp is necessary to reproduce the original speed from the stored data. Given this set of points, one of the main requirements on the algorithm is to generalize spatio-temporal dependencies between these points. An example for such a long-term dependency is depicted in figure 5.1. The important feature of this loop-making trajectory is its helical shape: i.e. if the loop starts at a point with timestamp $t = n$, it is necessary to return to a proximity of this point in time step $t \gg n$ in order to complete the helix. The

expression $t \gg n$ indicates that there are many other (less relevant) points between these highly significant starting and end points of the helix. The significance of the points has to be detected by the learning algorithm during the learning process. This is particularly difficult in the case of long-term dependencies as for the helix. We will explain beneath why this requirement limits the choice of algorithms severely.

2. **Consume as few training examples as possible**: This requirement is an important contribution to the usability of the system. As mentioned above, we want to implement learning by demonstration in a way that users from other domains can easily program robots without knowledge on technical details. This implicates that the learning procedure itself is easy to use and time-saving. Therefore, a learning algorithm, which needs too much training data in order to derive a generalization, would conflict with this goal. In our case of automation of knot-tying procedures, the algorithm should not require more than ten training examples of the knot-tying task. Otherwise, the assistance of the system would be rather recognized as a burden than as a help. This issue has already been indicated by other authors like Fuentes et al. [90].

3. **Fast performance**: Besides the reduction of demonstration data, the time complexity of the training algorithm should be contained within certain limits, too. The important parameter is not the relative time complexity depending on the size of the training set (O-notation), but the absolute time consumption of the algorithm. Again, the acceptance of the whole system will be at stake if there is a notable time delay between training and execution. It should be possible to perform the learned task immediately after demonstration. At most a time delay for computing of about five minutes will be acceptable.

4. **Robustness**: Another important requirement is the robustness of the results. This is especially important for automated knot-tying, since minimally invasive surgery is a sensitive application domain where no errors should occur. Most of the learning methods mentioned beneath tend to produce generalization errors if applied to environments not covered by the training set. Therefore, it is indispensable to avoid, or at least to detect generalization errors. Otherwise, the results will be useless.

Concerning these requirements we have evaluated different types of learning algorithms which have been employed recently to solve similar tasks. First of all, we will have a closer look on Hidden Markov Models and Kalman filters, both instantiations of Bayesian networks (confer [93]). A Hidden Markov Model (HMM) is an abstract state machine with external observations O_t, hidden states X_t and probabilities P_n for transition between internal states. Each internal state can produce any observation with a certain probability. The overall process implements the so-called Markov property: i.e. the current state X_t is only dependent on its predecessor X_{t-1}, but independent from all other states. This methodology is particularly useful to represent procedures with observable outputs where the underlying model is not known from the first. This was recently exploited for natural speech analysis. In this case, the observations are so-called phonemes, which are the basic entities of human language. Given a recorded speech consisting of a progression of phonemes, the question is which model, i.e. which written sentence, has given raise to this sequence. In similar fashion HMMs are

Hidden Markov Model

exploited for optical character recognition. The interesting question is now, whether this technique can be used to learn complex spatio-temporal interdependencies like the trajectories for knot-tying. Pilot surveys of this issue have been conducted in [160]: their challenge was the other way round: to recognize surgical motions like a knot-tying trajectory. According to the author, the recognition rate with basic HMMs was rather small. This is due to limitations of the HMM approach: HMMs are only capable of handling one time-dependent variable, but much the worse, the Markov property already limits temporal dependencies to one time step. Therefore, HMMs in their original form can only be employed for rather simple tasks with negligible spatio-temporal dependencies, like peg-in-a-hole [104]. Some extensions to basic HMMs have been proposed, which can deal with more than one variable and look back several time steps, but they are computationally expensive and only suitable for short-term dependencies. However, HMMs have been employed for offline analysis of surgical tasks [99] and the evaluation of surgical skills. Well-known extensions are hierarchical HMMs [169, 158], which have in common that they combine several variables to a super state, which leads back to basic HMMs, but complexity rises by the power of n, where n is the number of different variables.

Another instantiation of Bayesian networks is the Kalman filter. Its basic purpose is the tracking of rather simple trajectories like the movement of a plane. Like the HMM, the Kalman filter was originally designed to track one time-dependent variable. In addition, the estimation of a subsequent state is dependent on a control law that incorporates an error model derived from previous comparisons of the estimated value and the actual value. Therefore, the Kalman filter is only suitable for predictable, physical processes. It is not possible to describe arbitrary movements as they are known from telemanipulation systems with the help of a linear control model. There exists an extension of the Kalman filter for non-linear control models. But its application is computationally expensive and still cannot cope with arbitrary movements. So far, this technique has only been employed for feasibility studies in rather theoretical applications (e.g. movement of a 2DoF planar robot [167]). Therefore, we will not pursue this approach any further.

Kalman Filter

Another recently adopted methodology for learning tasks is support vector machines (SVMs). Their basic field of application is classification, i.e. decomposition of the task space into meaningful subspaces (e.g. character recognition). The basic idea is to transform the data from feature space into a so-called kernel space where its classification is simplified (kernel trick [192]). For example it is possible to design a kernel where ellipsoid subspaces in feature space can be represented by hyperplanes in kernel space. The optimal hyperplane, which separates different classes of data, will be determined by quadratic programming. According to Mercer's theorem [149], the dot product in this optimization can be replaced by a kernel function designed for the corresponding feature space. Therefore, the transformations of data from feature space into kernel space need not be carried out explicitly. However, one of the main drawbacks of SVMs is exactly the application of this kernel trick, which is only possible if the dimension of the kernel space (i.e. the partitioning of the feature space) is already known. Therefore, SVMs are predominantly employed for static classification tasks. A good example is optical character recognition, where the partitioning of the feature space (26 letters) is already well-defined. Though SVMs provide remarkable results in this application domain, it is very difficult to apply them to spatio-temporal data sets like trajectories. According to experts in this field a proper application of SVMs to time series is computationally extensive and has been applied only to benchmark examples [157] or has required a huge training set [176]. Apart from that, there have been attempts to combine SVMs with HMMs to a hybrid model [69]. In this application, the outputs of SVMs

Support Vector Machines

have been observed by HMMs in order to segment telemanipulation tasks. It has been stated that this procedure is particularly interesting for complex tasks in robotic surgery, but so far, there have been no evaluations on this proposal.

While both previously mentioned techniques, HMMs and SVMs, are methods for supervised learning, there exist also some methods for unsupervised learning. Unsupervised learning is especially attractive, since it requires less user interaction and a-priori knowledge. On the other hand, unsupervised algorithms are only applicable to rather simple learning tasks. This can be demonstrated by means of Self Organizing Maps (SOMs [126]). Unlike SVMs, they work in a reverse fashion: while objects in feature space are transformed to a higher-dimensional kernel space by SVMs, they are projected to a lower-dimensional map by SOMs. The biological archetype of SOMs is the planar topology of certain regions in the brain. Their main applications are in the field of preprocessing, data mining and cluster analysis. This concept has also been adopted for basic robot control (mappings between sensor input and motor commands [215]) or the implementation of motion primitives [54], but since the processing of spatio-temporal data like trajectories is difficult and cumbersome, only little research exists regarding issues similar to ours. Owens and Hunter [177] have examined the application of SOMs to human motion trajectories. Their results have been promising only with respect to the classification of large scale movements. In addition, they have generated only two groups for classification (normal human movement and suspicious behavior). According to their evaluation, SOMs are not capable to assess complex trajectories with greater deviations. The description of higher level features of a trajectory, like start/end points or the curvature of a trajectory, is intractable with SOMs, since it is difficult to include a-priori knowledge into the learning process. Therefore, this approach is certainly not applicable to the processing of delicate surgical movements.

Self Organizing Maps

The last methodology for learning motion trajectories we want to mention here is neural networks. While they come in a variety of different types, their basic functionality can be reduced to directed graphs with at least one layer for in- and output. Depending on the type of network, an arbitrary number of hidden layers can be inserted between in- and output. Each layer consists of several nodes, which contain rules to process numerical data at their input edges in order to generate some output. Each edge of the graph applies a certain weight factor to the values and forwards the result to the destination node. The function for processing the inputs (activation function) may vary between different types of networks, but in most cases a logistic or $tanh$ function is chosen, which maps all inputs to a range between 0 and 1. Results of all input functions are merged at the destination node. In most cases, this is achieved by means of a weighted sum over all inputs. During a training stage, the network is faced with pairs of inputs and compatible outputs. In order to produce a desired output given a certain input, weights of the edges are adjusted accordingly. The procedure of achieving this behavior for as many given in-/output pairs as possible is called generalization. Afterwards, the network should be able to produce sensible results for previously unknown inputs. In our case of trajectory learning, we demand from the neural net to provide a sensible continuation of the trajectory, given different starting points.

The basic version of neural networks is the single-layer perceptron with one in-/output layer. Though it can emulate certain boolean functions like AND and OR, it is not capable of higher level data processing (cf. XOR-problem and linear separability [79]). Extensions of this version, so-called multi-layer perceptrons, are trained by means of gradient descent algorithms with backpropagation of the error.

Neural Networks

In short this means the following: if a certain input is presented to the network we can influence the output by adjusting the weights of the edges. Initially, these weights are chosen more or less randomly. Therefore, the output of the network might most probably diverge from the expected result of the training example. Once we alter the weights, the output will change accordingly, i.e. the error will be increased and decreased, respectively. Our goal is to change the weights in away to achieve a maximum decrease of error in each step. If the size of this steps, or in other words the extent of changes applied to the weights (learning rate), is too coarse, we might miss the optimal value or get stuck in local minima of the error function. This procedure is comprehensively discussed in literature about neural networks, and therefore, we will refrain from any further details here. It is possible to represent every computable function with a neural network, which makes them Turing complete [196]. Though corresponding networks can be defined for each possible function, it is a severe problem to derive a network from training data, if the underlying function is not known. If there are only few training examples available and the topology is very large, neural networks tend to overfit. I.e. the network can perfectly reproduce the training examples, but cannot induce sensible results for previously unknown inputs (lack of generalization). Another serious issue is the definition of the network topology itself. It cannot be predicted, which topology might produce the best results for a problem. However, some heuristics are available for choosing the topology once the training set is provided. If the topology is too large, there is an increased danger of overfitting. On the other hand, if the network is underpopulated with nodes, it might not be able to reproduce each detail of the desired function. A good solution to this topology issue are recurrent neural networks. Unlike multi-layer perceptrons, which are acyclic graphs, recurrent networks contain cycles, i.e. their output may serve as input for some parts of the network. Learning algorithms for recurrent neural networks work in a similar way as for multi-layer perceptrons. Recurrence is handled by unfolding the topology during training (back-propagation through time). Another advantage of recurrent neural networks is their straightforward ability to handle time series. Thereby, the computation of the network is decomposed into snapshots at discrete time steps when new values are presented as input or outputs are fed back. This constitutes some sort of memory, which enables the network to discern dependencies between historic events and present outputs. Zegers and Sundareshan [224] have shown that it is possible to generate arbitrary trajectories with RNNs. They have proposed a method where learning is decomposed into a temporal learning part and a spatial learning part. So far, they have successfully applied this concept only for rather simple trajectories in 2D space. Kolb et al. [127] have proposed a RNN-based method for generalization of trajectories, which produces similar results like our approach based on fluid dynamics (see below). In contrast, their methodology is limited to simple trajectories without self-crossing and their learning algorithm is very time-consuming. However, recurrent networks would be an ideal tool to handle long-term dependencies as described above. Note that multi-layer perceptrons can achieve the same, due to their Turing completeness. The problem is that a corresponding conventional neural network designed for such a task would bear a huge number of nodes, which makes it very cumbersome, if not impossible, to train it without overfitting.

5.1.3 Trajectory Learning with LSTM Networks

In practice, training long-term dependencies is also a critical issue of recurrent neural networks. This is due to the fact that long-term dependencies will get lost in the feedback loops of the network during time (vanishing gradient [60]). The network forgets historic events after a while, since they are maintained only in form of results from multiple applications of activation functions. This issue has first been

tackled by introducing memory cells, which can store information without modifications and access it again after arbitrary time steps. Networks exhibiting this feature are called long short-term memory (LSTM) networks [103]. In order to prove feasibility of our trajectory learning task with LSTM networks, we have performed the following experiment [14]: We have taken a programmed version of the surgical knot (see below) as a starting point. Our goal was to learn the loop-making part of the trajectory with an LSTM network as benchmark example (cf. fig 5.2). Therefore, we have recorded 25 training examples of loop-making with our surgical system. Each trajectory consists of about 1500 points, which are sampled at a distance of $0.1mm$. For now, we have only considered translational coordinates (x,y and z) while neglecting the orientation of the surgical instrument (more precisely: the orientation of the gripper of the instrument). We took these training sets as inputs for the LSTM network. The first 50

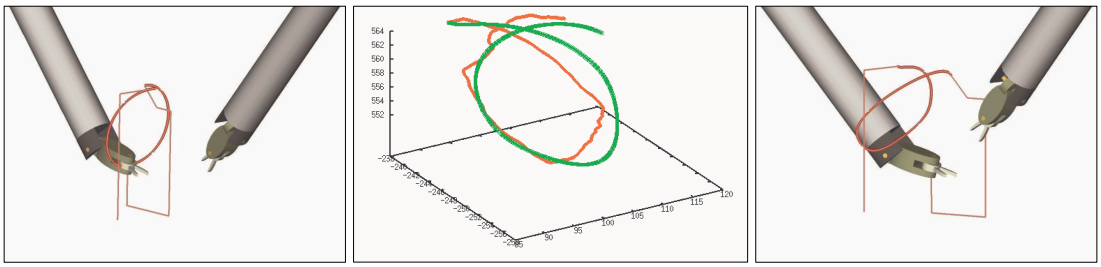
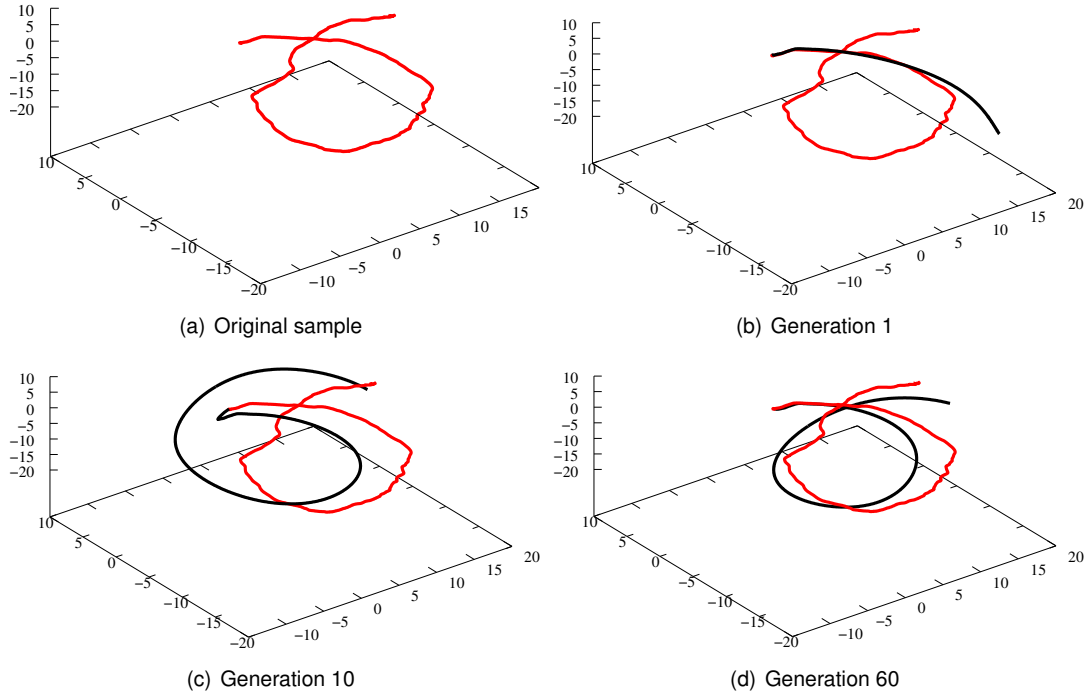


Figure 5.2: Original trajectory (left), during learning (middle), and smoothed trajectory (right) [*images published in [14], original data recorded by H. Mayer, neural network output by F. Gomez and D. Wierstra*]

coordinates of each set have been presented to the network without enabling the recurrent feedback loop (so-called washout time). At each time step the network was faced with new coordinates of one single point of the trajectory while its output produces the subsequent point. The error function, which is going to be minimized, is the sum of squared distances between the proposed points of the network and the actual points of each training set. It is also possible to train LSTM networks with a gradient descent method as it is known from perceptrons or basic recurrent networks. The disadvantage of this approach is that the error function of our task is rather complex (1500 time steps!) and exhibits many local minima. Therefore, the gradient descent algorithm will probably get stuck with a suboptimal solution. In order to overcome this hitch we applied a recently published method, called Evolino (evolution of systems with linear outputs [219]). The basic idea of this algorithm is to generate different distributions of weights and memory cell activations for the LSTM network (subpopulations). The fitness of each subpopulation is evaluated by comparing the output of the network with the training data. The network, which produces the best results with respect to this function, will be selected as an ancestor of the next generation of subpopulations. This procedure has been proven to outperform gradient descent algorithms in significant benchmark applications [191]. In figure 5.3 one can see the different stages of evolutionary learning. Note that all trajectories are already normalized, which is not the case in fig. 5.2 (graph in the middle). The important fact is that results improve from generation to generation until a predefined stopping criterion is reached in generation 60. At this point, the deviation (least squared distance) of the trajectory predicted by the network from the actual trajectory taken from the examples falls below an acceptable threshold. The complete learning procedure was included into the software for trajectory planning. After the initial segment of the knot-tying trajectory was carried out, control is handed over to the LSTM network. After the network has received some starting points, it generates the loop-making part of the trajectory at the desired position (see fig. 5.2 right side). After



(a) Original sample (b) Generation 1

(c) Generation 10 (d) Generation 60

Figure 5.3: Different stages of the Evolino learning algorithm [*plots published in [14], original data recorded by H. Mayer, neural network output and plot generation by F. Gomez and D. Wierstra*]

this part is finished, the gripper is moved to a well-defined point again and the remaining part of the pre-programmed knot is carried out. Before the whole procedure is applied to the real-world scenario, it can be assessed in the simulation environment. This prevents execution of malformed trajectories, which might contingently be generated by the network. Anyway, after normalizing the training data (trajectories are transformed to start at the origin), malformed trajectories did not occur any more. As a positive side effect, the learned trajectory was smoother and carried out faster than the original part of the pre-programmed knot (3.4 seconds for the learned version compared to 11.3 seconds for the preprogrammed loop). In all cases, the trajectories (which have been carried out by the real robots) have generated a surgical knot.

5.1.4 Discussion of reviewed methods

In this chapter we have presented a variety of learning algorithms, which have been implemented by other authors in the field of trajectory planning. Most of them are not applicable to our scenario of robotic knot-tying. In the case of LSTM networks, no applications to this field were available, and therefore, we have conducted own experiments. LSTM networks turned out to be a superior method for learning spatio-temporal dependencies. To our knowledge, similar results have not been demonstrated with other methods, yet. However, there are still some issues regarding the requirements defined above. One disadvantage of the LSTM networks is their demand for a rather large training set. It was not possible to achieve sensible results with less than 20 training examples, which is still far from the desired maximum of ten. Producing 25 training examples, as we did in our experiments, would be too time-consuming for everyday practice. This will restrict the usability of the system for users from

a foreign domain and reduce their willingness to include semi-automatic features into their workflow. Another drawback of LSTM networks, in particular the Evolino method, is the time-consuming learning phase. For our examples it took up to two hours between acquisition of training data and the execution of the resulting trajectory. This will also impede the acceptance of this method.

5.2 Methods for Human-Machine Skill Transfer

In the last section we have already reviewed the decomposition of complex tasks into primitives and assessed some learning algorithms working on trajectories. Both concepts are important fundamentals for the main topic of this work: human-machine skill transfer.

First of all, we would like to give a formal definition of the term skill transfer. In biology and psychology the expression “skill” is used as an all-embracing term. On the one hand, it may refer to rather simple phenomenons like an organism moving a limb, on the other hand, it also comprises cultural acquirements of mankind like using speech or playing chess. Anyway, all skills have in common that they answer a certain purpose in the corresponding environment. A skill is a certain execution plan, which fulfills a task in order to reach a predefined goal (e.g. a bacterium has the skill to move towards a light source in order to find food, or humans may differ in their skills to reach the goal of winning a chess match). Each skill has a beginning t_0, an end t_n, and therefore, an execution time $t_e = t_n - t_0$. The purpose of the skill (the solution to a certain task) can be described by means of changes in the environment. Here, an environment is a set of objects Γ and a set of properties Θ of these objects. The properties are defined quite generally and can refer to geometric quantities (like rigid transforms or distances) as well as forces or torques exerted to the object. A task is a transformation of an environment at t_0 into an altered environment at t_n, which will be expressed by $(\Gamma_0, \Theta_0) \overset{\Phi}{\longmapsto} (\Gamma_n, \Theta_n)$. The transition $\overset{\Phi}{\longmapsto}$ constitutes the **task** Φ. In many cases Γ_0 will be equal to Γ_n as long as the set of objects remains unchanged. The set will be changed for example if objects are destroyed or new objects are generated out of existing ones. For clarification we will give a description of the well known peg-in-a-hole task with this formal description: The environment is defined by the set of objects $\Gamma_0 = \{\text{peg, hole}\}$ with Θ_0 comprising the rigid transformation T_{ph}, which denotes the position of the peg relative to the hole. Therefore, the peg-in-a-hole task can be expressed as $(\Gamma_0, \{T_{ph}\}) \overset{\Phi}{\longmapsto} (\Gamma_n, \{I\})$, i.e. the goal is to align the peg with the hole until their centers are matched, and thus, the transformation between peg and hole finally constitutes a unit matrix I. Task definitions provide the trainee with abstract information about a task, which is going to be demonstrated by the teacher. Providing this kind of hints is the very essence of **scaffolding** as it is used in our skill-transfer architecture.

formal definition of a skill

Each **skill** is a description how the desired transformation of the environment, the task, can be carried out (e.g. this description can be realized by a control program for robotic manipulators). We restrict our considerations to so-called sensori-motor skills: i.e. a manipulator can affect the environment and can determine parts of the status of the environment by means of appropriate sensors. Therefore, the manipulator is a set on its own, comprising the corresponding parameters like joint angles or speed of the end-effector. Accordingly, a skill is an algorithmic solution to a task, which can use the state of the manipulator and the environment as inputs (as long as those can be determined by the sensors of the system). As outputs, it produces new states of the manipulator. A priori, this algorithm

is a kind of a black box, which has to be implemented in some way. For the examples mentioned above [124, 117, 107], this can be achieved in a constructive manner by composing control programs out of predefined primitives (cf. 5.1.1). The major disadvantage of this approach is its inflexibility and its demand for an expert in robot programming. Today, robots are often deployed in areas where they are operated by experts from other domains (like manufacturing or surgery). Therefore, it would be desirable to program robots in a more intuitive way. An elegant way to do so is programming by demonstration: i.e. a human expert demonstrates a task in a situated domain (e.g. welding or making a surgical knot). The robotic system witnesses this demonstration by means of its sensor readings, which may include visual information, forces, accelerations or similar parameters. With the help of this information, an algorithmic solution to the corresponding skill is derived. As mentioned above, the most basic implementation of this concept is “Teach-In”. This technique was developed during the early years of robotic research as a programming modality for industrial robots. Operators need not to be experts in robot programming, but control the robot with an appropriate input device in order to solve a certain task. During this procedure all motor commands issued to the robot are recorded. Afterwards, the demonstrated movements can be reproduced arbitrarily often. Note that the trajectory will be reproduced exactly in the same way it was demonstrated and that there is no interaction with the environment, i.e. the corresponding procedure needs no inputs. Anyway, this implementation still meets the requirements on a skill as defined above. Therefore, “Teach-In” demarcates the lowest level of abstraction of skill transfer from human to robot. Although this technique serves well for industrial processes with standardized components and workflows, it will fail in applications with dynamically changing environments like surgery or service robotics. For this types of applications it would be more desirable to have a solution, which is much more abstract and flexible. “Teach-in” implements only one of many possible solutions to a task, but this solution might not be optimal, or even not work in some situations. The implementation cannot be altered afterwards in order to react to unexpected changes in the environment. Given such changes, it could even happen that it is no longer possible to solve the task with this single trajectory. Therefore, “Teach-in” can be seen as a kind of overfitting regarding the underlying task (similar to overfitting of neural networks; cf. 5.1.2).

A more flexible approach is to abandon predefined trajectories and to specify restrictions of the workspace instead. One possible implementation of this concept is potential fields [119]. Since this approach looks akin to the one we present below, we will refer to it later again in order to indicate significant differences. Another method within this context is the representation of a skill as a petri net [146] or as a finite state machine [161], which both react to certain states of the environment with an adequate output. Both approaches require a discretization of the input space and an appropriate temporal resolution of the workflow. While the former approach utilizes different contact situations as states, the latter determines state transitions by mapping inputs onto certain incidence relations by means of a fuzzy classifier (cf. Zhang et al. [227]). The same technique of fuzzy classification has also been employed in other projects, where it was utilized for reinforcement learning [225] and for the selection of control rules [85], respectively. Reinforcement learning as a means of implementing skills has also been proposed by several other authors: Malak et al. [141] and Zhao et al. [229] have proposed trial-and-error methods based on reinforcement learning in order to implement skills. Given a certain task, i.e. the corresponding transformation of the environment (see above), their algorithms try to emulate this transformation by testing different sequences of motion primitives (we will give a definition of primitives later in the text). The better the outcome, the higher the award assigned to the corresponding strategy. The aim is to find an optimal strategy for a perfect solution to the task. A re-

review on literature about skill transfer methods

lated, though more formal approach was proposed by Atkeson et al. [58]. They have implemented the skill to balance a pendulum by reinforcement learning. The optimal solution was presented by a human beforehand, which makes this solution an instantiation of the learning by demonstration paradigm. Learning by demonstration can be seen as an advanced technique for skill implementation. The goal is to generate robot programs automatically. Wang et al. [216] have described a procedure to segment a human demonstration into primitives, express each primitive by code for the robot's controller and recompose the code again in order to emulate the demonstrated behavior. Synonymous expressions for learning by demonstration are imitation learning, learning from observation, learning by doing or programming by demonstration. All of them refer to the same fact: a human user shows a skill to a robotic system, which should be able to derive a general implementation of this skill from the demonstrations. This yields the advantage that control programs for the underlying system can be composed by users from other domains, which are no experts in programming robots. As mentioned above, it is a crucial requirement for these algorithms to require as few demonstration examples as possible. The first, optional step in learning by demonstration is to **preprocess the acquired demonstration**. This presupposes that some kind of a-priori knowledge is available, e.g. in the form of thresholds or workspace limitations. Kaiser et al. [116] suggest to remove those parts of the demonstrations whose influence on the environment lies below a certain, predefined level (e.g. tiny rotations of the work piece). Chen et al. [72] recommend to omit all demonstrations, which lie outside a predefined region of the workspace. Afterwards, demonstrations of the same task have to be matched to each other in order to find similarities. This can be achieved by dynamic programming as proposed by Ogawara et al. [171]. The algorithm is based on interactions between the manipulator and different workpieces. These interactions are used as synchronization points for dynamic programming. Based on this approach, Sato et al. [187] have refined the procedure by providing a method for **trajectory segmentation**, which constitutes the next step in programming by demonstration. Their method is based on velocity events, changes in contact states and different changes in the environment. Similar techniques have already been reviewed in 5.1.1, where primitives have been derived from robot trajectories. The final, remaining step of programming by demonstration is the **application of the skill** to a possibly unknown environment. This is particularly challenging, since the system should possess the ability to generate every possible trajectory to solve a certain task in any environment it comes upon. Learning by demonstration is a popular solution to this problem and several research groups, including the ones already mentioned above, have proposed methodologies, which are instantiations of this paradigm. For example Zöllner et al. have proposed an architecture for programming by demonstration in [230]. User demonstrations are decomposed into meaningful primitives, which are called "elementary operations" in their work. A priori knowledge is included by means of defining a parameter space for each object. Each action induces the generation of a hierarchy of subtasks ending up at basic primitives. Subtasks are arranged in a precedence graph, which is derived from previous demonstrations of the user. In order to decide whether a certain action can be applied, sets of pre- and post-conditions are defined for each action. Given such a pre-condition, the system serializes the corresponding precedence graph, yielding miscellaneous alternatives for an execution plan. One of the plans is actually carried out, depending on optimization criteria (e.g. speed).

three major steps of learning by demonstration

5.3 Bio-Inspired Learning

All technical solutions to programming by demonstration mentioned above are more or less based on the well-known engineering paradigm of task decomposition: Given a complex task, break it up into subtasks, which are easier to understand (primitives) and recombine them in order to get a solution (skill). The methodology presented in this work will also follow this principle, but before we provide any details, we want to refer to some different approaches. Unlike the methodologies mentioned above, which all have a strict engineering background, some researchers are rather associated with biology or psychology and try to understand and emulate human behavior in order to solve learning by demonstration tasks. Zollo et al. have proposed a bio-inspired control model for basic manipulation tasks [231]. This approach is based on recent research on the principles of the cerebral cortex, which is modeled as a neural network. They assume that a robot has to learn its own movement potential, before it can acquire knowledge about complex tasks. Therefore, they start at a very basic level by learning the inverse kinematics of the robot by means of a neural network. The robot was taught to correlate certain postures, which are perceived visually, with certain settings of its joints. Afterwards, the robot learns how to reach different objects in the working space and how to handle them. This was reached by cooperating neural networks. Although the inverse kinematics of the system is already known, it makes sense to let the system learn its own body in order to get a deeper understanding of the underlying processes and to apply the acquired solutions to higher level tasks. There is strong evidence that humans learn complex motions in a similar way by recombining congenital motor primitives [128]. In addition, it is possible to acquire new motor primitives during learning processes [94, 220]. Similar approaches for learning the inverse kinematics of a robotic system have been proposed by Ijspeert et al. [108] and Izawa et al. [112]. Lopes et al. have proposed a new architecture for learning by imitation [138]. For posture recognition, this approach dispenses with marker-based tracking, but is exclusively based on vision. The necessary calibration is achieved via viewpoint transformations: i.e. all movements refer to a distinct coordinate system. The authors assume that hand-eye coordination (the inverse kinematics of the arm) is not inherently known a-priori. Therefore, a so-called visuo-motor map has to be generated with the help of multi-layer perceptrons. The actual learning by imitation is based on recently explored structures in the brain, so-called mirror neurons. Those are active during observation of a demonstrated task, as well as during execution of the same task by the observer. Within this architecture, mirror neurons are modeled as Bayesian networks.

5.4 Description of the Task

With the system described above we have already performed surgical tasks like cutting and sewing (cf. chapter 4). Due to the more complex handling of minimally invasive systems, an intrinsic issue of these procedures is their completion taking significantly more time and suffers from an increased rate of errors. For example 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 minimally invasive knot-tying. Our skill transfer framework is employed to supply a generic knot, which can be triggered by the surgeon in order to perform a specific knot 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 abilities (e.g. a different type of surgical knot). The first step towards this is to analyze how the knot-tying task

is performed by a human. Based on these observations, surgical knot-tying can be decomposed into the following subtasks (see fig. 5.4). After piercing, the needle with the attached thread is pulled out of the tissue by 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 the 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 complete the knot (4).

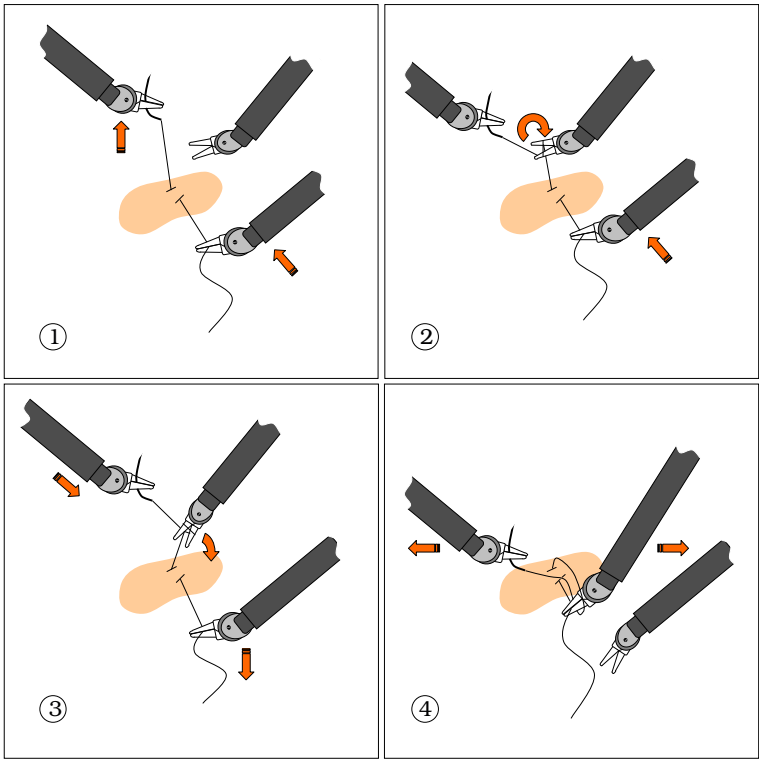


Figure 5.4: Knot-tying procedure

Although some groups tried to transfer knot-tying theory [181] into applications to robotic learning [115], there has been only little research on the particular task of automating a surgical knot. Kang has proposed a generalized version of a taught knot in his PhD thesis [118]. Thereby, the task is performed with special knot-tying instruments, while we are using multi-purpose instruments. Hynes et al. [39] and Wakamatsu et al. [214] proposed robotic setups for knot-tying, but they 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 is also some work available on analyzing the knot-tying task itself, but these publications are rather focused on surgical evaluation than on automation [68, 123, 43, 184, 46, 204]. In addition, some solutions are documented, which dispense with employment of conventional surgical instruments. Kuniholm et al. [133] have constructed a cartridge containing a preloaded knot, which can be discharged after getting into contact with tissue. As part of the *CoMe* project, a helical needle for automated surgical knot-tying was patented [148]. A general procedure for surgical assistance by using surgical fixtures has been proposed by a research group of the *Johns Hopkins Haptics Lab* [40]. This method could be used for further automation of surgical knot-tying (e.g. by providing assistance for piercing through tissue).

We have already integrated a programmed version of knot-tying into our system in order to alleviate analysis of the task. The trajectory to perform a surgical knot was manually instructed with our offline interface. Our system is equipped with three grippers, and therefore, can assume the work of both operator and assistant. We were able to tie a surgical knot in about 30 seconds, which is already an improvement compared to human performance in minimally invasive knot tying. At this point, no learning was included, i.e. this knot cannot be adapted to unknown environments, which is the topic of the following sections. Anyway, implementing this procedure has provided us with valuable hints about knot-tying with a robotic system. This was a first step towards the situated dialog described below: the user has to be aware of the sensori-motor abilities of the system, i.e. the user has to teach the knot within an environment, which can be completely comprehended by the system. Otherwise, the user will overstrain the system by assuming a-priori knowledge the system does not possess.

5.5 Layers of Abstraction

In the next sections the knot-tying task described above is formalized and prepared for integration into our skill transfer framework. The reader might find it helpful to learn about the outcome of our experiments, first (in the results section 5.8), in order to get a more practical understanding of the underlying methodology. Afterwards, it should be much easier to understand the subsequent formalizations, which are based on the definitions introduced at the beginning of this chapter.

In our application, a skill will constitute the top level of the hierarchy employed for learning by demonstration. Given a certain pre-condition, the user can interact with the system at a high level of abstraction by means of applying skills. For example, a surgical knot can be applied after the surgeon has pierced the tissue and grasped both ends of the thread (see below). As mentioned above, a skill describes the transition between a certain pre-condition (a certain state of the working environment) and the desired result. Implementations of a skill, which exhibit no further granularity, can easily induce overfitting. As already mentioned, the best-known example for such a behavior is “Teach-In”. The corresponding implementation of a skill produces a trajectory, which can only be applied to a static environment with invariant pre- and post-conditions.

In order to improve generalization, we have adopted the paradigm of dissecting complex skills into a set of sensori-motor primitives. While most authors mentioned above use a sequential dissection (horizontal dissection of the timeline), it is inevitable for our scenario to have a vertical dissection, too. The latter refers to a geometrical dissection of the trajectory in order to get one primitive for each manipulator. Therefore, the additional challenge of synchronizing different primitives with each other will emerge. Based on these considerations, we now can refine the definition of a skill and give a formal description of primitives. First of all, we will have to subdivide a task into smaller **tasklets**. While a skill is the solution to a certain task, primitives will be solutions to tasklets. Or in other words, a complex task will be broken up into smaller tasklets, which can be solved by primitives. Afterwards, those primitives can be reassembled in order to get a skill, which will be a solution to the original task. Like top-level tasks, each tasklet will be a transformation between pre- and post-conditions of an environment, denoted by a corresponding transition: $(\Gamma_a, \Theta_a) \overset{\phi}{\longmapsto} (\Gamma_b, \Theta_b)$, where $\overset{\phi}{\longmapsto}$ is the transition

definition of tasklets

of tasklet ϕ. Therefore, a task is a set of tasklets, which meet the following requirements:

$$\exists\, t_b \leq t_n : (\Gamma_0, \Theta_0) \overset{\phi}{\longmapsto} (\Gamma_b, \Theta_b) \tag{5.1}$$

$$\exists\, t_a \geq t_0 : (\Gamma_a, \Theta_a) \overset{\phi}{\longmapsto} (\Gamma_n, \Theta_n) \tag{5.2}$$

$$\left\{ \overset{\Phi}{\longmapsto} \right\} \in \left\{ \overset{\phi}{\longmapsto} \right\}^{+} \tag{5.3}$$

The first requirement determines the pre-condition of task Φ being also a pre-condition of at least one tasklet (the same accounts to the post-condition in the second requirement). The last statement refers to the transitive hull of the tasklet transitions: the transition of task Φ has to be included in the transitive hull in order to let these tasklets be a valid dissection of the task. For a complete description of the task, the transitions of the tasklets have to be ordered appropriately. Therefore, we need to extend our definition of a task by three different relations. Those are irreflexive, transitive relations defined on the set of tasklets $\phi^0 ... \phi^m$. The most important is the precedence relation "$<$", i.e. $\phi_0 < \phi_1$ self-evidently means tasklet ϕ_0 has to be completed before tasklet ϕ_1 starts. In addition, we need a synchronization relation "$=$". Accordingly, $\phi_0 = \phi_1$ indicates that the tasklets have to be carried out at exactly the same starting time, and also implies that execution times of both tasklets are equal. For integrity reasons we also define an exclusion relation "$\neq$", which prevents tasklets from being executed at the same time. An application example for this type of relation would be a situation where two manipulators cannot move to the same place at the same time, but the corresponding tasklets are not ordered by any precedence. Putting it all together we get the following **extended definition of a task**:

$$\Phi = \{\phi^*, <, =, \neq\} \tag{5.4}$$

While the transition of a task Φ is determined in quite an abstract manner (transform environment (Γ_0, Θ_0) into environment (Γ_n, Θ_n)), we will define four concrete types of tasklets: linear motion, 2D motion, force controlled motion and synchronized motion. Those four will suffice to construct our application example of surgical knot tying. A **linear motion** is constituted by the following tasklet:

$$(\Gamma_a, \{T_a\}) \overset{\phi_{\text{lin}}}{\longmapsto} (\Gamma_a, \{T_b\}) \tag{5.5}$$

Γ_a contains at least the manipulator, which is going to carry out the movement. In the pre-condition the manipulator is placed at posture T_a and finally should reach T_b in the post-condition. Both postures are interconnected by a linear movement regarding the translational part. Rotations are calculated by a spherical linear interpolation based on quaternions [195]. A more general version of this tasklet is the **2D motion**:

$$(\Gamma_a, \{T_a\}) \overset{\phi_{\text{2D}}}{\longmapsto} (\Gamma_a, \{T_b\}) \tag{5.6}$$

where both postures are interpolated by an arbitrary 2D spline, which interconnects the 3D coordinates of T_a and T_b. Again, rotations are handled by a spherical linear interpolation. Note that this definition poses no further restrictions on the spline, except embedding into a plane. The next tasklet, the **force restricted motion** is basically a linear motion:

$$(\Gamma_a, \{T_a, F_a\}) \overset{\phi_{\text{F}}}{\longmapsto} (\Gamma_a, \{T_b, F_b\}) \tag{5.7}$$

The difference is that the linear motion will be stopped before T_b is reached, if a certain force F_b is exceeded. In this case, the rest of the rotational motion will be carried out in place. The last tasklet we

want to introduce here is the **synchronized motion**. This one is special, because, as a pre-condition, it requires another tasklet ϕ_s to exist, which is not a synchronized motion tasklet:

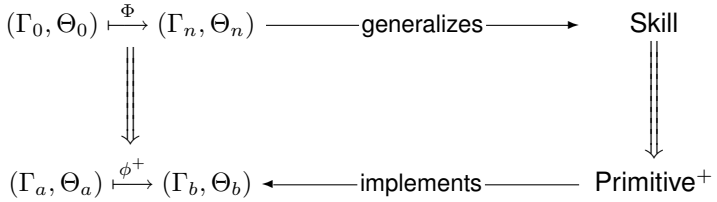
$$(\Gamma_a, \{T_a, \phi_s\}) \overset{\phi_{\text{syn}}}{\longmapsto} (*, *) \tag{5.8}$$

The post-condition is arbitrary since the purpose of this tasklet is not directed towards a certain state of the environment, but it is correlated with the trajectory of another tasklet ϕ_s. This correlation can be realized by an affine bijection. Another possibility is a mapping induced by a neural network.

All tasklets mentioned above will be implemented by **sensori-motor primitives**. These primitives are basically execution plans, which generate trajectories for the manipulators of the system. The execution of trajectories might be influenced by inputs of sensors (e.g. cease movement if a certain force is applied). Regarding our implementation, the generated trajectories can be directly applied to the system, i.e. it is possible to concatenate primitives without replanning. This can only be reached if the motion of the manipulator stops at the end of the primitive. If the velocity of the manipulator is unequal to zero, it will also influence the trajectory of the subsequent primitive. In this case a replanning of the complete trajectory is necessary once the primitives are reassembled to a skill, but in our implementation the only prerequisite for reassembling primitives to skills is the end posture of a primitive being equal to the starting posture of the next one in a row. This can already be realized at the level of tasklets.

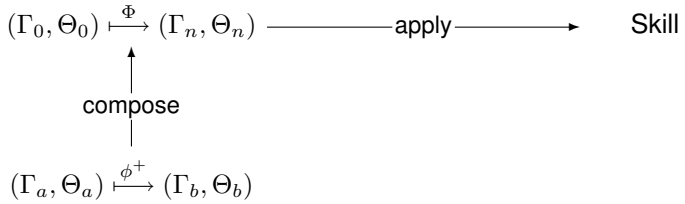
5.6 Learning by Demonstration with Scaffolding

Above, a task was defined as a transition between states of an environment (Γ, Θ). In most of the cases, this transformation will lead to an alteration of Θ, the object's properties, while the set objects itself remains unchanged. On the level of tasks and tasklets, properties are only type-defined, but contain no specific information, yet. For example, the definition of the linear movement tasklet (cf. equation 5.5) contains the assertion that the transition of the tasklet changes the posture of the manipulator ($T_a \overset{\phi_{\text{lin}}}{\longmapsto} T_b$). It is not determined what the postures look like or how the transition is performed (except that the movement is restricted to a straight line). This is first concretized when the tasklet is instantiated by a primitive, which will describe the postures by homogeneous matrices and provide an execution plan for the movement. Therefore, a tasklet is a generalization of a motion primitive as well as a task is an abstract description of a skill. Both definitions constitute the scaffolding framework of our architecture (i.e. hints in the knowledge base of the system in order to facilitate understanding of demonstrated tasks). Their interdependencies are depicted in the following diagram:

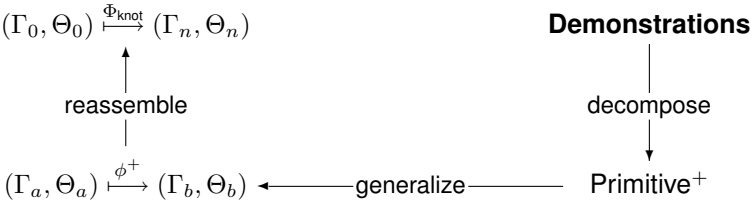


While a skill is only valid for one specific situation (one well-defined stage of the environment), a task describes the necessary transition in a generalized way. Therefore, the goal of any abstract problem solving technique is to provide the definition of the task in order to derive skills (concrete solutions to a problem) for arbitrary states of the environment. One approach, which was already mentioned

above, is the explicit construction of tasks out of predefined tasklets. This technique was presented in [107]. Note that these authors do not distinguish between tasklets and primitives, and therefore, their primitives are a mixture of both concepts:



However, constructing tasks out of tasklets requires the user to be an expert in programming of robots. In order to dispense with this prerequisite, learning by demonstration was introduced. Using our terminology, the main goal of this approach is to derive a general description (the task Φ) from a variety of user demonstrations. User demonstrations are nothing else than skills, i.e. they are specific solutions to a task, which the user has come across during demonstrations. For rather simple skills, like getting into contact with a surface, it might suffice to directly generalize the corresponding task from the demonstrations [71]. If the underlying task exhibits a more complex structure, this approach will lead to overfitting, since similarities of the demonstrations are much harder to detect. We have already tried this option for our knot-tying scenario. As mentioned above, this task consists, amongst others, of winding and grasping the end of the thread. The relative distance between the place of winding and grasping differs from demonstration to demonstration. If the system is not aware of them being different sub-trajectories, it is not possible to overlay the demonstrations and extract similarities. On the other hand, if winding and grasping are evaluated separately, it is much easier to detect significant features of these smaller sub-trajectories. Therefore, winding and grasping are modeled as different tasklets of the knot-tying task. Our first attempt was deriving this decomposition of tasklets directly from user demonstrations by unsupervised methods like cluster analysis. This would have featured the advantage of self-acting performance, which needs no further user interaction except providing the demonstrations. Referring to the diagram above, we have taken the way of decomposing demonstrations (skills) into meaningful primitives. Afterwards, the primitives were designated to be generalized by tasklets, which in turn can be used to reassemble the knot-tying task:



Unfortunately, it was not possible to achieve full-automated decomposition of skills into primitives for our application scenario. This issue has already been experienced by Lin et al. [137], who tried to automatically segment surgical motions in order to evaluate their quality (with respect to surgical experience). They have been successful after relating the recorded data with a Bayesian motion model. In our case, the major issue with unsupervised learning was the limited sensor information provided by the system. In particular, we were not able to include visual information into this first approach. Therefore, decomposition of the skills had to exclusively rely on geometric information of trajectories and occurring forces. Given these features, it was not possible to distinguish significant points in the demonstration from insignificant ones. For example, the tightening of a knot exhibits a characteristic force profile, but the same profile was often encountered due to accidental collisions of the end effectors. The same applies to the extracted features of the trajectory. We have tried to utilize changes in

the closing state of the gripper, inflexion points of the trajectories and points with low velocity as demarcation points for primitives. While important interactions of the user are actually marked by these points, the reverse does not apply: not every point detected by this algorithm is significant, e.g. the user often opens and closes the gripper without intention or without purpose. For this pilot survey we have used singular value decomposition (as proposed by Arun et al. [57]) for grouping primitives, which implement the same task. Unlike the decomposition into primitives, this part of the attempt worked quite well and we have used this technique in further approaches (see below). The deficiencies in decomposition have revealed another issue, which is also closely related to the reduced sensor information: the user is not aware of what the system (the detection algorithm) is looking at. The user has too much possibilities for interaction, which cannot be detected by the system. For example, the user can guess the curvature of an occluded thread and use this information for grasping. Since the system only recognizes the underlying trajectory it has no chance of an appropriate generalization without this information. In other words: **The major challenge for learning in this scenario arises from the fact that the user and the system do not speak the same language and act in different environments**. Therefore, we have to find a way to force the user to interact in a way, which can be comprehended by the system. A promising method to reach this goal is situated learning, or more general, communication via situated dialogs. The outline of this principle has already been introduced in chapter 1. Originally, this concept refers to the communication between two humans. The standard application of this methodology is skill transfer between a human teacher and a human trainee. The teacher has to be aware of the abilities and the restrictions imposed on the trainee. Even if the teacher has more sophisticated abilities or even more possibilities as the trainee, demonstrations must be restricted to the understanding and capabilities of the trainee. For example, this is a major issue addressed by the education of teachers for physically disabled persons [64, 114]. Needless to mention that robots outperform humans in many categories like precision and speed, but regarding their sensori-motor abilities they are still quite underdeveloped. Unfortunately, transfer of handicraft skills mostly relies on a good coordination of “sensors and actuators” of the trainee. In addition, both teacher and trainee need to have a common model of the environment where learning takes place. We have realized this concept of situated learning by our specification of tasklets, which is on the one hand powerful enough to uniquely describe significant interactions with the environment and on the other hand simple enough to be executed by control software of robots. In addition, it realizes a scaffolding framework for learning by structuring information in an appropriate way. Tasklets can be composed to complex tasks, and therefore, they provide high flexibility. Since the demonstrated skills have to be an instance of a predefined task, which serves as an abstract pattern with this respect, the user is forced to adjust demonstrations to the abilities of the robot. If demonstrations meet these requirements (i.e. they are instances of the task), they can be understood, formalized and imitated by the robot. Therefore, if the user intends to demonstrate new, previously unknown skills, a corresponding description of the task has to be provided beforehand. This can best be achieved by recombining existing tasklets. If this is not sufficient, it is also possible to define new tasklets as long as they are compatible with the sensori-motor abilities of the robot. In our software interface, we provide no possibility to define new tasklets (since this has to be done on source code level), but new tasks can be composed out of the four tasklets defined above (linear, 2D, force-controlled and synchronized movements). After providing the definition of the task as pattern, an arbitrary number of demonstrations can be recorded and fed as input to the skill transfer pipeline. This pipeline consists of the following processing steps:

- Acquire a new demonstration of the task from the user (skill acquisition)
- Check each demonstration for being an instance of the underlying task. In case of a match, the corresponding demonstration will be stored for further processing, otherwise it will be rejected.
- Valid demonstrations are decomposed into meaningful primitives, which correspond to tasklets. This procedure is prepared by unsupervised methods operating on the features of the recorded trajectory. Afterwards, a pattern matching algorithm can be applied, searching for primitives (= instances of tasklets).
- Meaningful information, which will be used to implement a robotic skill, is extracted from the demonstrated primitives and stored in the knowledge base of the system. The determination of "meaningful" is extracted from the definition of the corresponding tasklet. For example, if an instance of a force restricted motion (see above) is detected, the force and posture of the starting and end point is stored for further processing. Information not related to this tasklet (e.g. postures of stopover points) can be discarded.

As already mentioned, this method will already work and provide reasonable results after one single demonstration. Anyway, the stability of the solution (the generality of the extracted information) will usually increase, if additional demonstrations are included into this process (e.g. the model for the interdependence of trajectories, as defined for the synchronized motion, will improve by adding new points). Once the necessary information is harvested, it can be utilized to apply a skill to a previously unknown environment by performing the following steps:

- Given the extracted information and the definition of the tasklets, the robotic primitives can be instantiated. The outcome will depend on the current application environment and the implementation of the tasklets (see below), which lead to a generalization of the task even after one demonstration. This is due to the fact that tasklets and their implementations already realize a knowledge base (e.g. in form of a fluid simulation for implementing 2D motions: see below).
- The generated primitives are contingently transformed and joined to form a robotic skill
- The newly formed skill can be assessed in a preview, which is generated in the simulation environment. It is particularly important to check the trajectory for collisions. This step can also be automated as described below.
- After the optional preview, the trajectory can be executed by the system.

Given the definitions above, we now can formalize the knot-tying procedure by means of a task and provide it as scaffolding for our system. For clarity reasons, we will depict this task in the form of tracks aligned to a timeline (as it is known from video / audio processing; cf. fig. 5.5): The first track refers to the dominant right hand, the second track to the left hand and the third one to the assistant gripper. Note that only the movements of the left and right hand should be learned by the system. The behavior of the assistant arm is part of the scaffolding, which is intended for later removal. The environment (Γ_0, Θ_0) of the starting point of this task is determined as follows: Γ_0, the set of objects, comprises the three grippers, while Θ_0 contains the postures of the grippers, including translation, rotation and closing state. While translations and rotations might be set to arbitrary values (within the limits explained below), the closing states of all three grippers are already fixed. The task can only be instantiated, if a certain initial condition has been established by the user. In the case of surgical

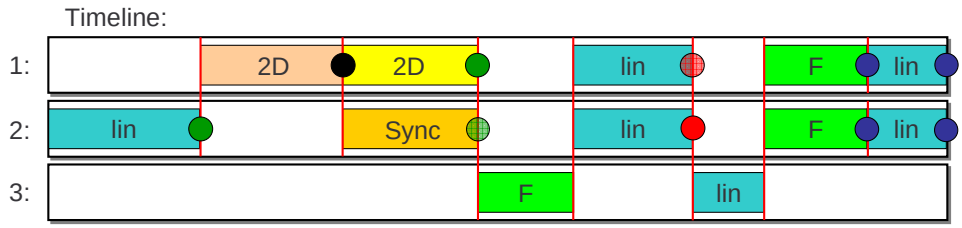


Figure 5.5: Knot-tying task

knot-tying, it is presumed that the needle is already pierced through the tissue and pulled out. In this situation the right hand holds the needle, or the proximal end of the thread, while the assistant gripper holds the loose end of the thread. Therefore, both right and assistant gripper are closed while the left hand is open. As long as the corresponding grippers remain closed (right hand and assistant), they are supposed to keep hold of the thread. The system is not able to detect the losing of threads during winding.

Given the correct starting condition, the task will proceed as follows: the left hand (track 2) moves towards a position where the thread can be wrapped around it. Afterwards, the right hand moves to a point where the first tangential contact is established between the thread and the left gripper. This has to be performed by a 2D motion, since the right hand has to sidestep the left gripper during this part of the procedure. Now, the right hand can wind the thread about the left gripper, which is also achieved by a 2D motion. This time, the left hand follows, driven by a synchronized motion. On the one hand, this allows for a comfortable method of performing bi-manual tasks: it is much harder to wind about a gripper whose position is fixed, since much of the concentration of the user is wasted to keep the gripper in a fixed position. On the other hand, the overall extension of the trajectory is reduced, since the synchronized hand always moves in a complementary direction with reference to the dominant hand. After that, the assistant gripper is used to strain the thread: i.e. it performs a force controlled linear movement. Since we restrict generalization to the right and left hand, the primitive for this motion is not derived from demonstrations, but is pre-programmed. As mentioned above, straining the thread serves the purpose of keeping the latter in a defined position (which will not be the case, if the loose thread can move randomly). Now, the left gripper can perform a linear movement towards the end of the thread. The right hand has to compensate for this by an adequate linear motion. Note that this behavior could have also been modeled by a synchronized motion, but we found that it already works fine by using a simpler solution based on two independent linear motions. By the way, using the simplest most possible decomposition of a task is very advisable, since complex solutions tend to overfit, because they pose less restrictions on the demonstrations. An extreme example would be modeling the hole task with only one or two 2D primitives. Nearly all possible movements could be subsumed under this definition of a task. Therefore, complex 2D motions should be used sparingly and only for short parts of the trajectory. Back to knot-tying, after moving towards the thread's end, the assistant is moved towards the position of the left gripper in order to deliver the thread. Now, the left gripper is closed in order to grab the thread. Afterwards, the assistant gripper can release the thread, and the left and right hand is moved toward opposed directions in order to tighten the thread. This is performed via force-controlled movements: i.e. the thread is released after a certain force is exceeded at one of the grippers. Therefore, the opening of the gripper has to be synchronized. If only one of the grippers would open after the determined force is exceeded, the force on the second gripper will

be reduced and an unbalanced knot, or even damages of the tissue, will result. Note that the forces are not explicitly defined within the task, but generalized from the demonstrations. After tightening the knot, the grippers are moved back toward their initial positions in order to enable a seamless continuation of manual control.

Usually, these patterns can be defined by users, which are no experts in programming robots. Each tasklet has to be delimited by a certain **event**. Events can be certain features of the underlying trajectory (inflexion points, movement clusters: see below), occurring forces or changes of the gripper's closing state. Each type of event is depicted by a colored circle in fig. 5.5 (e.g. red indicates a gripper open event). Given a tasklet $(\Gamma_a, \Theta_a) \overset{\phi_b}{\longmapsto} (\Gamma_b, \Theta_b)$, the delimitating event ε is a certain state of the environment at time t_b, i.e. $\varepsilon \in (\Gamma_b, \Theta_b)$. It must be possible to detect all employed events by unsupervised methods like cluster analysis or curve sketching. Since the next tasklet on the track is defined by $(\Gamma_b, \Theta_b) \overset{\phi_c}{\longmapsto} (\Gamma_c, \Theta_c)$, i.e. it starts at the same state of the environment as the previous tasklet has stopped, it is only necessary to define events to delimitate the end of the tasklet. The starting event of the first tasklet on the track is predetermined by the starting condition of the task.

events

By integrating the user into this process of composing tasks out of tasklets, the awareness of the system's capabilities will be raised, and therefore, demonstrations will be more target-oriented. The system will reject all demonstrations, which do not comply with the task. However, the demonstration process still relies on cooperative users: due to the desired generality of tasks it will be possible to deliberately generate demonstrations, which, on the one hand, will not be rejected, but on the other hand, will not produce the desired outcome. If these premises receive attention, the user can easily augment the task with meaningful information (by demonstrating instantiations of the task) and use the generalized result to produce skills, which can be applied to arbitrary locations within the environment.

5.7 Implementation

The implementation of our skill transfer architecture is divided into three major parts: a user interface for the definition of new tasks, the pipeline for processing trajectories (demonstrated skills), and the integration of skill application. Regarding our application scenario, the most important part of the implementation is the processing pipeline, which is supplied with trajectories from user demonstrations.

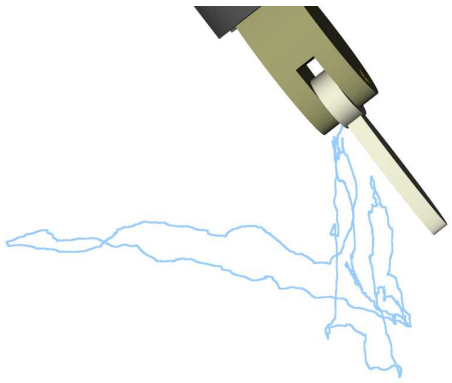


Figure 5.6: Knot-tying trajectory

The recording of trajectories is initialized directly after starting the manual user interface. Therefore, recording is transparent to the user, since it is not necessary to start it explicitly at a certain situation. The initial condition of a task, as mentioned above, will be automatically detected by the system within the demonstrated trajectory. The meaningless chunk parts at the beginning and at the end will be omitted. 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 (cf. definitions above). A task is a certain, well-defined description of how to solve a problem. A demonstration is an instance of this task performed by a human user. More generally, the term skill refers to any instantiation of a task. Finally, primitives are meaningful, non-overlapping subsequences of a skill. We apply the following steps in order to generate robotic skills based on user demonstrations:

1. **Event detection and smoothing of the trajectory**: unsupervised methods are applied in order to find significant points in the trajectory, which will be used later to delimitate primitives.
2. **Decomposition of the complete trajectory into meaningful primitives**: the demonstration is dissected into non-overlapping primitives, which have been identified as being instances of predefined tasklets.
3. **Grouping and matching of primitives from different demonstrations**: in order to extract information from congruent primitives in different demonstrations, all primitives referring to the same tasklet are grouped together.
4. **Reassembly of skills out of primitives**: If the user wants to apply the task to a previously unknown environment, a specific skill is generated with the help of the knowledge base, which has been augmented by the information extracted from the primitives.

5.7.1 Event detection and smoothing

In fig. 5.6 the trajectory of the left gripper during a demonstration of the knot-tying task is depicted. 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. As one can observe, the data acquired from user demonstrations is quite noisy and exhibits some tremor of the subject. Therefore, a first step is smoothing the data, but this is often associated with loss of information. Before we can smooth

the data we have to be sure, which parts of the trajectory are necessary for further processing, and which parts can be neglected without consequences. One idea was to use a Gaussian filter in order to remove noise, but this will also remove some important edges, which are essential to perform the task (e.g. the leftmost edge of the trajectory in fig. 5.6 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 with the help of spline approximation. Certain events are chosen as interpolation points, which demarcate important parts of the trajectory. These events will also play an important role in the decomposition of primitives. Events are detected by including additional modalities into our considerations. We have exploited spatio-temporal dependencies within the trajectory, the state of the gripper and occurring forces. Based on these features, we were able to distinguish four different types of events in the trajectories:

implementation of different events

1. **Movement clusters**: We have identified points where the user moves rather slowly. This is a strong indicator for increased precision. The applied methodology is based on profound research on the psychology of user interfaces. If the mouse is moved across the screen very fast, the user is not able to perform precise interactions with the user interface, since the focus window of the eye cannot achieve full concentration on the task [65]. Furthermore, this observation can be proven by Accot-Zhai's steering law [50], an enhancement of the well-known Fitts' law [87] for movements on constricted trajectories. In its original form it is expressed as

$$t_C = a + b \int_C \frac{ds}{W(s)} \tag{5.9}$$

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

$$\frac{ds}{dt} = \frac{W(s)}{b} \tag{5.10}$$

As one can derive from equation 5.10, 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 type of event can be easily detected by analyzing spatio-temporal features of the trajectory. Similar functions have been utilized in systems for surgical training and assessment (MIST-VR [91] and ICSAD [76]). In our application, trajectories are sampled at discrete time steps. Therefore, clusters can be found by examination of the distance of a sampling point from its predecessors. If this distance falls below a certain threshold, the corresponding point is marked as movement cluster. In order to get one distinct event, subsequent occurrences of clusters are merged. In fig. 5.5 cluster events are depicted as green circles.

2. **Closing state of the gripper**: This is another important indicator for significant interactions, which should not be smoothed out, and which might possibly delimitate primitives. Unlike the movement clusters, there is no need for a dedicated detection algorithm, since this information can be directly derived from the data. In fig. 5.5 opening and closing of the gripper is depicted as a blue and red circle, respectively.

3. **Turning points of the trajectory**: If the trajectory reaches a maximum or minimum, this event is often related with an important interaction. For example, regarding the case of knot-tying, a maximum in the trajectory is reached after the first tasklet of the right hand (aligning the thread with the left gripper).

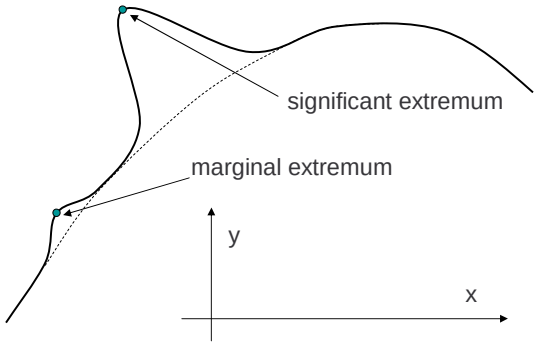


Figure 5.7: Search for extrema

While detection of turning points is comparably easy for curves defined by mathematical functions, it is difficult to detect significant turning points in trajectories acquired from human demonstrations. There are often insignificant local extrema (cf. fig. 5.7), which are caused by improper movements of the human hands. Then again, we do not want to restrict this approach to absolute extrema, since there exist only two of them for each gripper (for each track of the demonstration), and therefore, their significance is limited. In contrast, we want to detect all extrema causing a deviation from the global course of the trajectory (cf. fig. 5.7). In order to achieve this, we have applied the following method to detect extrema in trajectories: We determine all vectors connecting consecutive points within the trajectory and calculate the angles they constitute with the y-axis. Afterwards, we look for sequences of ascending angles and descending angles, respectively. If a sequence of at least ten ascensions follows after a sequence of ten descents, we mark the inflection as minimum. The inverse procedure applies to the detection of maxima. In order to filter insignificant extrema, we allow at most three disturbances within a sequence (e.g. ascensions within a descending sequence). This rather simple approach has provided reasonable results and allows us to detect all significant extrema in the knot-tying trajectory. Extrema are displayed as black circles in fig. 5.5.

4. **Synchronized events**: This event refers to the fact that a certain event in the trajectory of one gripper can also trigger certain interactions in the trajectory of another one. For example, regarding the knot-tying example, the grasping of the thread's loose end triggers the tightening of the knot, which is performed by both left and right gripper. Synchronized events can be identified, only if all other types of events have already been detected within the underlying trajectory. Therefore, detection of synchronized events is implemented by a second pass of the trajectory data. Synchronized events are displayed as transparent versions of the color of their base event (e.g. the sync close event is depicted as a transparent blue circle in fig. 5.5, while the gripper closing event itself is depicted in dark blue).

Events contain important information about locations, where the user was interacting with the environment (e.g. by grabbing some objects like the thread). Figure 5.8 shows a first evaluation of the

methodology mentioned above. Locations, where the user stayed within a radius of 1.5 mm for more than 0.2 $secs$ (movement clusters) are marked by a cone. If the trajectory is smoothed by a spline approximations, all points featuring a certain event have to be preserved, and therefore, the corresponding points have to be included as intermediate points of the spline. Besides smoothing, another important feature of the spline approximation of trajectories is resampling. This will be used only for primitives constituting a 2D motion or synced motion. Regarding linear motions, all points between start and end can be neglected. In picture 5.8, the color of the trajectory changes from green to red,

Figure 5.8: Spline approximation

once the gripper is closed. Events 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 no events have been detected on the way to the pick-up position of the thread and back. This indicates that the corresponding part of the trajectory is not too important, and therefore, a distinct smoothing can be applied (if we apply our pattern matching algorithm afterwards, this part will be transformed into a linear motion primitive).

5.7.2 Decomposition

After smoothing the trajectory and transforming it into a spline representation, the next step is to decompose it into meaningful primitives. As mentioned above, the decomposition of manipulation tasks into sensori-motor actions or motion primitives is an approved methodology to reduce complexity [116, 190]. In our case, the selection of meaningful primitives is quite complicated, because demonstrations are not normalized. Different demonstrations may be translated or rotated and primitives may appear with different parameterizations. Therefore, we cannot employ procedures proposed for normalized data (e.g. [182, 197]). Another design criterion for our primitive derivation is the segmentation of any given trajectory into a sequence of two-dimensional sections, which are instantiations of predefined tasklets. It is easy to proof that a two-dimensional decomposition is always possible, since every trajectory can be sampled to single lines, which actually have just one dimension. Then again, 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 (confer to the definition of tasklets). Regarding the decomposition into 2D primitives, we can find similarities, if we 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 is restricted to 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 sensori-motor behaviors

decomposition into primitives

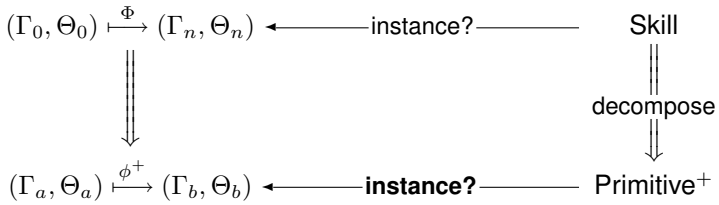
[52, 53]. Therefore, we can represent the trajectory of any Skill S in the following way:

$$S = X_0 \cup T_1 P_1 \cup X_1 \cup T_2 P_2 \cup \ldots \cup T_n P_n \cup X_n = \bigcup_{i=1}^{n} T_i P_i \tag{5.11}$$

$$P_i = p_0 \circ \cdots \circ p_{n-1} \circ \varepsilon_i \tag{5.12}$$

$$P_i \cap P_j = \emptyset \quad \forall i, j \in n; \quad i \neq j \tag{5.13}$$

where P_i is a two-dimensional primitive, which is placed at its correct position by a rigid transformation T_i. Between the primitives, chunk sections X_i might occur, which have no relevance for the task and can be replaced by a direct connection of consecutive primitives. In order to find a decomposition, which is compliant with the predefined task, we have to check whether the skill is an instance of the corresponding task, and therefore, if the skill can be decomposed into a series of primitives complying with the tasklets. Equation 5.12 says that each primitive is a sequence of points delimited by a designated point $p_n = \varepsilon_i$, which contains a significant event.



This check (whether primitives are instances of a certain task) is performed by means of the events detected in the previous step. Besides their employment for resampling the trajectory, these points are also important hints for selecting significant primitives. Usually, much more events are detected in the concrete trajectory than defined in the abstract task (cf. fig. 5.9). Therefore, it is not possible to perform an unsupervised decomposition of skills trajectory, exclusively based on events. A proper decomposition can only be achieved, if additional information in the form of task definitions is included. They assist the system in discerning primitives, and therefore, provide support in the sense of the

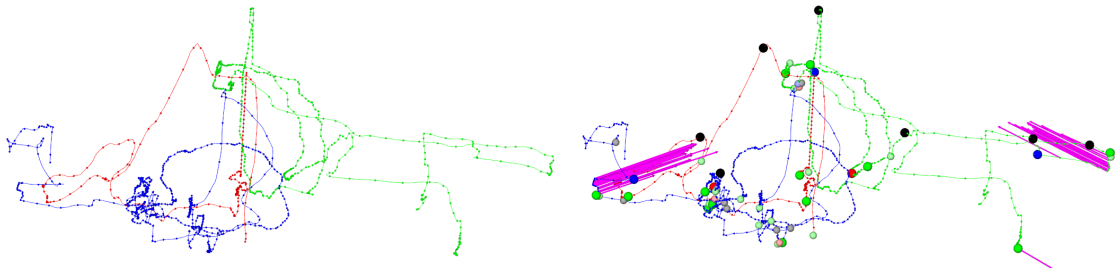


Figure 5.9: Original trajectory (left) and detected events (right)

scaffolding paradigm. As mentioned above, tasklets are separated by dedicated events within a task. We take the sequence of events from the previous step and scan it for the first event of interest. For example, in case of the knot-tying task, the first significant event is a movement cluster, depicted by a green circle in fig. 5.5. Therefore, this primitive is delimited by the beginning of the trajectory on the one side and by the first movement cluster event we come across on the other side (cf. fig. 5.9). All other events occurring before this movement cluster will be skipped. Note that the primitive starts at the beginning of the demonstration, although there is a time lag at the beginning of the first track in fig.

5.5. These pauses will be inserted in a subsequent step: after the instantiation of the tasklets they are synchronized to comply with the temporal specification of the task.
The following algorithm is employed to decompose the demonstration of the task into primitives compliant with the tasklets. On the one hand, it operates on the sequence of events derived from the demonstrations of the skill (user data), on the other hand, it utilizes the task pattern, which is part of the knowledge base of the system (internal data). Therefore, the algorithm selectively reduces the data in user demonstrations by means of pattern matching. The part of the data, which is going to be extracted, is determined by the definitions of the tasklets (see below).

$\Pi = \{\ \}$
$E = \{$`all events detected in S`$\}$
$c = 0$
$S_{\text{tmp}} = S$
`for all` $\phi_i \in \Phi$
    $\varepsilon_i \in (\Gamma_i, \Theta_i)$
    `for` $j = c$ `to` $|E|$
        `if` $\varepsilon_j = \varepsilon_i$
            $S_{\text{tmp}} = S_A \circ \varepsilon_j \circ S_B$
            $P_i = S_A \circ \varepsilon_j$
            $\Pi = \Pi \cup P_i$
            $S_{\text{tmp}} = S_B$
            `break`
        `endif`
    `next`
    $c = j + 1$
`next`
`if` $|\Pi| < |\Phi|$
    `demonstration rejected`
`endif`

The first step is to reset variable Π, which will contain all extracted primitives at the end. Afterwards, E is initialized with the set of all events (event points as defined in equation 5.12) detected in the skill S. S and S_{tmp} are embodied by a set of points as defined by equations 5.11 ff. As mentioned above, decomposition of skills into primitives can be achieved, only if the corresponding task has been defined by the user beforehand. Therefore, we assume that task Φ and its decomposition into tasklets already reside in the knowledge base of our system, and is therefore accessible by the algorithm. By the outer `for` statement, the algorithm sequentially loops over all events ε_i serving as delimiters of tasklets. The corresponding sequence of events is defined by the timeline depicted in fig. 5.5. In the inner loop we scan all events, detected in the previously unprocessed part of the skill's trajectory (ε_j), for matches with ε_i. If a match was detected, the corresponding part of the trajectory is stored as primitive $P_i \in \Pi$ for further processing. This procedure is repeated until all detected events are processed. This state of the algorithm is depicted in fig. 5.10. If the algorithm was not able to find instantiations of all tasklets in Φ, the corresponding trajectory is rejected for not being a valid demonstration of the skill. Otherwise, the significant features of the primitives can be extracted in the next step.

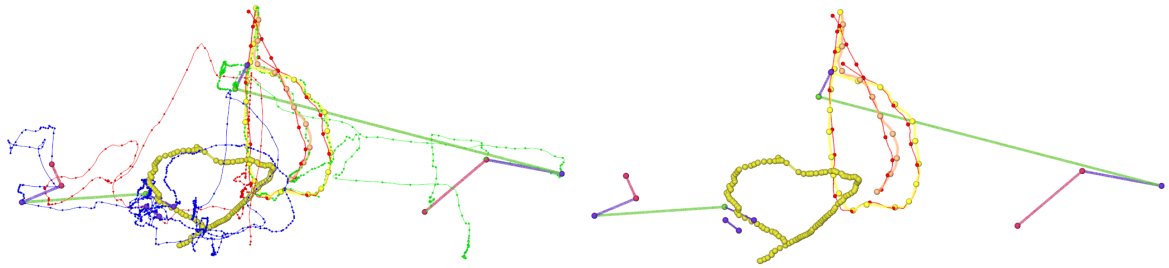


Figure 5.10: Extracted primitives with original trajectory (left) and by themselves (right)

5.7.3 Feature Extraction

Once the primitives are derived from the trajectory and assigned to the corresponding tasklets, the relevant information of the primitives can be extracted and stored in the knowledge base of the system. The structure of the knowledge base and the procedure of inserting information is depicted in fig. 5.11.

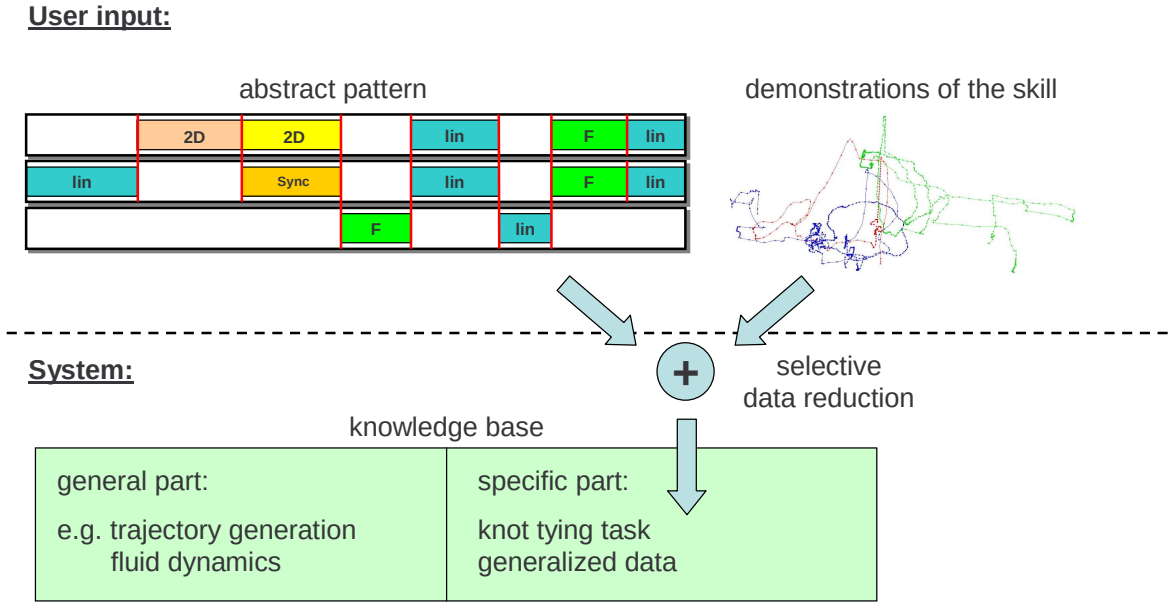


Figure 5.11: Knowledge base of the system

The knowledge base consists of a general and a specific part. The general part contains information about the execution of primitives, which can be utilized in every task. For example, the general knowledge base comprises a trajectory generator in order to implement the straight line movements of linear motion primitives. Another example is the fluid simulation for the instantiation of 2D primitives. This part of the knowledge base is hard-wired and will not be changed by user interactions. The only situation, which requires an alteration of the general knowledge base is the extension of the system by new tasklets. This has to be effected by adjusting the source code of the system. During normal operation, the user can only change the task-specific part of the knowledge base. This contains the definition of the task (in our case the knot-tying task depicted in fig. 5.5) and extracted information from the demonstrations, which is necessary to actually instantiate the task in real world environments. The corresponding task-dependent information is extracted via selective data reduction. There is a special

knowledge base

method of data extraction for each type of tasklet:

1. **2D movement**: So far, the primitives, which are instantiations of this tasklet, are simply sequences of points in 3D space. As mentioned above, they are interpolation points of a spline representation of the trajectory. The central part of this approach is the function `planefit(P)`, which calculates an optimal plane for a given set of 3D points (i.e. the primitive) Optimality is defined by means of the minimum least squares method. The regression problem is stated as follows:

$$^{2D}z_i = ax_i + by_i + c; \qquad d_i = z_i - {}^{2D}z_i; \qquad Min\left(\sum_{i=0}^{n} d_i^2\right) \tag{5.14}$$

   where (x_i, y_i, z_i) is the ith point of the corresponding primitive. Note that we are not dealing with normal distances, but with directional distances, which are easier to obtain and provide reasonable results in our application domain. The minimization is actually performed by linear regression. The return values of the function are the parameters of the plane a,b and c, and the sum of squared distances. If the mean squared error exceeds a certain threshold (i.e. the points does not fit well to a 2D plane) the whole demonstration will be rejected for not being an instance of the predefined task. Otherwise, the plane parameters and the interpolation points of the spline are stored in the task-specific knowledge base for later instantiation of the task.

2. **Linear movement**: If a linear motion primitive was detected in the demonstration, it suffices to store the start and end point of the underlying trajectory. All other points in between can be omitted.

3. **Force controlled movement**: As for the linear motion, the start and end point of the primitive is stored in the task-specific knowledge base. In addition, we store the maximum force vector during the movement. This will be used later to control the instantiation of this tasklet.

4. **Synchronized movement**: As mentioned above, the synchronized movement tasklet can only exist in connection with a 2D tasklet. The points of the corresponding 2D primitive are mapped onto the points of the synchronized movement primitive. We have applied the following transformation to describe this mapping:

$$\begin{pmatrix} x_1 \cdots x_n \\ y_1 \cdots y_n \\ z_1 \cdots z_n \\ 1 \cdots 1 \end{pmatrix} = A \begin{pmatrix} {}^{2D}x_1 \cdots {}^{2D}x_n \\ {}^{2D}y_1 \cdots {}^{2D}y_n \\ {}^{2D}z_1 \cdots {}^{2D}z_n \\ 1 \cdots 1 \end{pmatrix} \tag{5.15}$$

   In order to calculate transformation A, each point in the trajectory of the referenced 2D primitive has to correspond to a point in the synchronized movement primitive. Therefore, the number of points n used for this procedure has to match for both primitives. Finding the transformation is formulated as the following singular value decomposition problem (cf. [208]):

$$C = \frac{1}{n}\sum_{i=1}^{n} \left[\ \overrightarrow{p_i} - \overline{p}\ \right] \left[\ \overrightarrow{{}^{2D}p_i} - \overline{{}^{2D}p}\ \right]^T \xrightarrow{\text{svd}} C = USV^T \tag{5.16}$$

$$\text{where } \overline{p} = \frac{1}{n}\sum_{i=1}^{n} \overrightarrow{p_i};\ \ \overline{{}^{2D}p} = \frac{1}{n}\sum_{i=1}^{n} \overrightarrow{{}^{2D}p_i} \tag{5.17}$$

   $\overrightarrow{p_i}$ is the i-th point of the currently processed primitive, while $\overrightarrow{{}^{2D}p_i}$ is the i-th point of the 2D primitive the former is synchronized with. 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 and s_3 of matrix $C^T C$. U and V contain the eigenvectors of CC^T and $C^T C$, respectively. According to [208], we can determine the scaling factor f, the 2D rotation matrix R and the translation vector $\overrightarrow{t}$, which can be used to map ${}^{2D}p_i$ onto p_i:

$$f = \frac{s_1 + s_2 \pm s_3}{\sigma_P^2} \quad \textit{where} \quad \sigma_P^2 = \frac{1}{n}\sum_{i=1}^{n}\left(\overrightarrow{{}^{2D}p_i} - \overline{{}^{2D}p}\right)^2 \tag{5.18}$$

$$R = UV^T \tag{5.19}$$

$$\overrightarrow{t} = \overline{p_i} - f \cdot R\overline{{}^{2D}p_i} \tag{5.20}$$

Note that the sign of s_3 depends on the determinant of C, when calculating f (in addition, the sign of the last row in U has to be adapted accordingly, when calculating R). Since this method yields only a similarity transformation, we actually applied a method for linear optimization in order to get an affine transformation for A. In addition, it is also possible to learn the dependency between right and left hand motion by a neural network. The corresponding approach was developed during the work on a diploma thesis connected to this project *[T. Gebhardt: Automated Knot-Tying in Minimally Invasive Surgery]*. Although being more complex, this methodology reveals significant advantages if applied to multiple data sets (in comparison to the affine mapping: see results section 5.8).

5.7.4 Generalization and Instantiation

Once the user has demonstrated the task by 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 initial condition as defined by the task (Γ_0, Θ_0). 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 assistant. Afterwards, the task can be instantiated by pressing a foot switch. At this moment, the manual input of the user will be blocked and the skill is generated, which is adapted to the actual situation. After a preview of the skill's execution was displayed, the corresponding trajectory can be actually carried out by the robots. Therefore, the instantiation of a task at the desired position will be accomplished by the following steps: generation of the trajectories for each tasklet in the task, connecting these trajectories to form a skill, and displaying a preview of the skill in the simulation environment. There are different methods of instantiation for each type of tasklets:

1. **2D Movement**: On the basis of the extracted information in the task-specific part of the knowledge base, we could instantiate this tasklet 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 [189]. 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. So far, there has been only little research on dynamical systems towards the adaption and generation of motion primitives. Ijspeert et. al [108] 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. [173]. 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 for 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 [77, 121]. A 3D version of this so-called panel method [102] has been proposed by Zhang et al. [228] for motion planning of flying robots. Since computing power was quite limited that time, these approaches have been revitalized lately by other authors [217]. Recently, there has also been work on analyzing trajectories by means of dynamical systems. Dixon et al. [78] have proposed a method for segmenting primitives based on linear dynamical systems. Although, this method 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 conditions, e.g. in a wind tunnel. Streaklines occurring in these environments are nothing else than trajectories of particles in a fluid. To our knowledge, 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. [73]):

fluid simulation

$$\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla)\vec{u} + \nabla p = \nu \Delta \vec{u} + \vec{f} \tag{5.21}$$

$$\nabla \vec{u} = 0 \tag{5.22}$$

∇ is the Nabla operator and Δ is the Laplace operator: $\Delta = \nabla \cdot \nabla = \nabla^2$; $\vec{u}$ is the velocity of the fluid, ν its viscosity and $\vec{f}$ are external forces like gravity. As mentioned above, the tasklets we want to represent by this dynamical system are already in 2D space. Remember, during feature extraction we have stored the plane parameters of the demonstration in the task-specific knowledge base. Therefore, equations 5.21 and 5.22 can be simplified to:

$$\frac{\partial u}{\partial t} + \frac{\partial p}{\partial x} = \nu \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) - \frac{\partial u^2}{\partial x} - \frac{\partial (uv)}{\partial y} + f_x \tag{5.23}$$

$$\frac{\partial v}{\partial t} + \frac{\partial p}{\partial y} = \nu \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) - \frac{\partial (uv)}{\partial x} - \frac{\partial v^2}{\partial y} + f_y \tag{5.24}$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{5.25}$$

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 uniform

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 principle of this procedure is depicted in fig. 5.12. For clarity reasons this picture shows only a limited number of cells, which would be too coarse for practical application - 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 to a staggered grid in order to assure numeric stability. A detailed description of this methodology can be found in [96]. In addition to discretization, we have to fix the velocities at the boundaries of the simulated area (e.g. u_{00} and v_{00} in fig. 5.12). 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, and due to the nature of this equations, no particle will ever reach or even penetrate a boundary. Thus, a kind of collision avoidance is included in our system from scratch. Of course, this affects only 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 this problem of external collisions below (cf. section 5.7.5). With the help of these boundary conditions, we

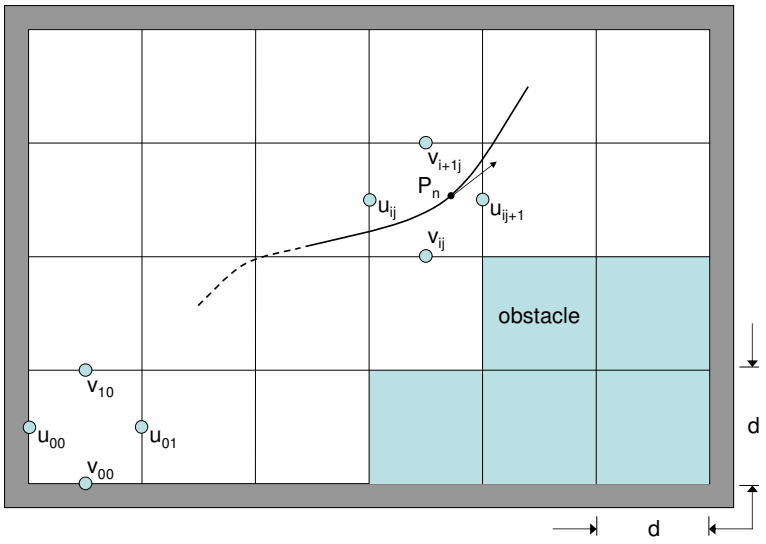


Figure 5.12: Fluid simulation

can also define obstacles within the area of simulation in order to adapt the trajectory to new environments. Each 2D movement tasklet 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 calculate the arc length of a spline, 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 5.21). 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}. If $|(S(x + \Delta x)) - S(x)| > |d| + \epsilon$, we try $S(x + 0.5\Delta x)$. Otherwise, if $|(S(x + \Delta x)) - S(x)| < |d| - \epsilon$, we try $S(x + 2\Delta x)$, and so forth. Usually, 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 $S(x) = P_n$ 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} (see fig. 5.12): i.e. we calculate a preset for these velocities at time step t_n. All other velocities within the grid are 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 tasklet. Since we know the position and orientation of the simulation grid from 5.11 (extracted plane parameters), we can generate suitable 3D points in order to instantiate the tasklet in the new environment (cf. fig. 5.13). 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.

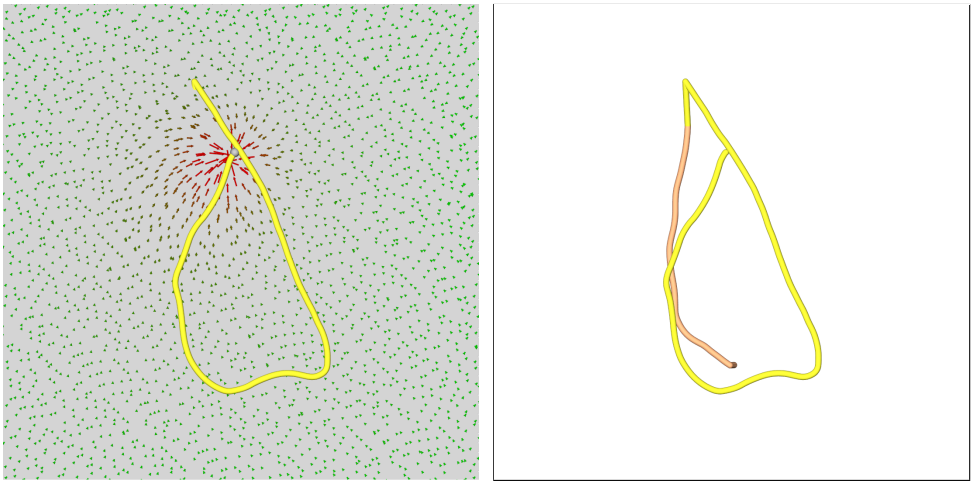


Figure 5.13: 2D primitive after fluid simulation (left) and embedded into the skill (right)

2. **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. 5.14).

3. **Force Controlled Movement**: Although this tasklet is basically a straight line movement, its instantiation yields some further issues. We have extracted the maximum force and the end points from 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, particularly during tightening a knot, it might happen that the gripper has reached the end point before the maximum force is reached. Therefore, we provide an allowance for further tightening by elongating the line by 25% of its original length (cf. fig. 5.14, 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.

4. **Synchronized Movement**: As mentioned above, a synchronized movement can only exist in connection with a 2D movement. During feature extraction, we have already stored a corresponding mapping function. This can be used now to produce the points of the synchronized motion.

Figure 5.14: Instantiation of a force controlled movement

The mapping function is applied to every point of the 2D primitive we want to synchronize with (cf. fig. 5.15). 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 points have to be concatenated by a straight line movement.

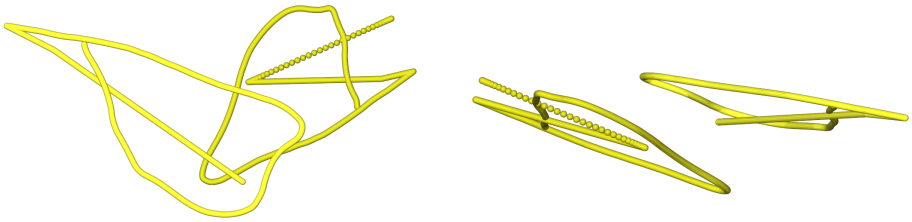


Figure 5.15: Synchronized primitive (dark yellow) and reference movement (light yellow)

5.7.5 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 instruments in their working space might induce collisions of parts of the robots. 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 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 (remember, 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 (cf. section 3.2). If any collision occurs 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. 5.16.

virtual obstacles

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 tests have 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. 5.17.

Usually 3D related programs interact with the graphics card via *OpenGL* commands, which are issued

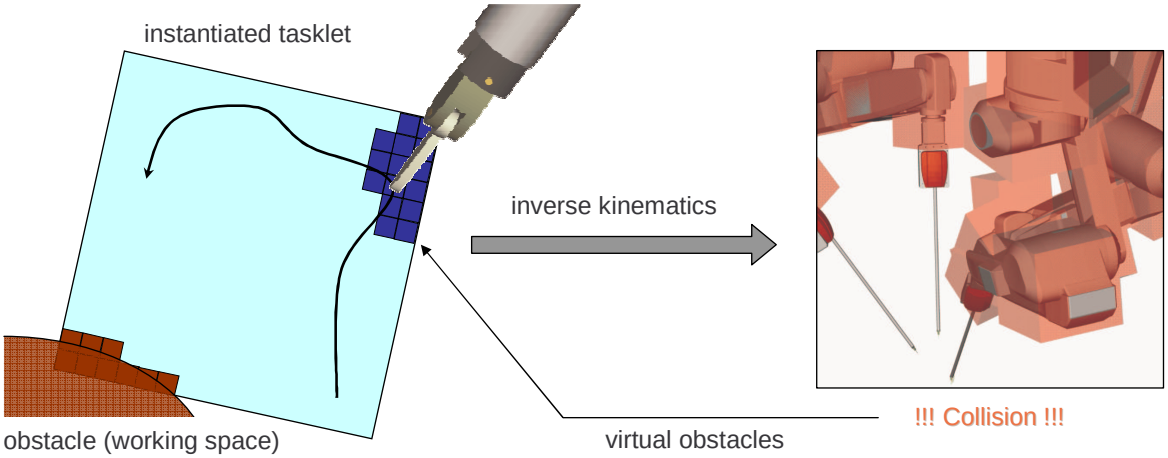


Figure 5.16: Generation of virtual obstacles for collision detection

within *C++* code segments. Basically, these commands provide the card with new geometric information about the 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 objects of the scene into pixel-based fragments on a 2D plane. Those are forwarded to the texture engine, which optionally projects predefined textures 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. 5.18). When a 2D tasklet 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 positions, 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 in practice, 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. possible positions of the end effector). Now, we can

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

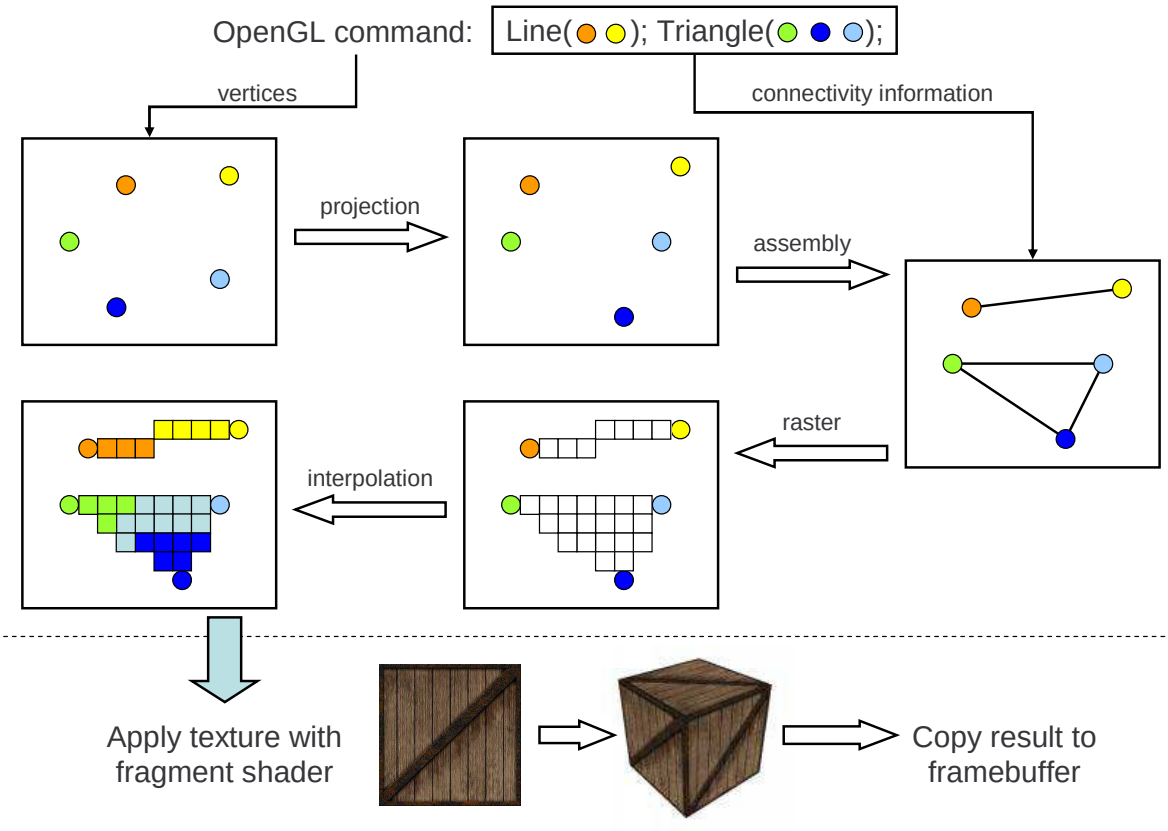


Figure 5.17: *OpenGL* pipeline for rendering of 3D objects [*source: Lighthouse3D.com*]

apply the inverse kinematics of our system (cf. section 3.3) 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 (cf. section 3.2). 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. If a collision is detected, the corresponding pixel is colored white, otherwise black. Afterwards, we can access the pixels on the frame buffer and generate virtual obstacles from it (cf. fig. 5.18). 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.

5.7.6 Application of the skill

After the tasklets are instantiated in a new environment by the methods introduced above, we have access to separated trajectories for each gripper, i.e. one trajectory for each track in fig. 5.5. The next step is to synchronize the trajectories as defined by the task. For example, before the first 2D movement of track 1 (cf. fig. 5.5) can be performed, we have to wait until the second gripper has finished its linear movement towards the winding point. Such waiting times are inserted by simply duplicating the corresponding position within the trajectory of the gripper, which has to wait. This is always possible, since the primitives of 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. Optionally, it is possible to simulate the movements of the

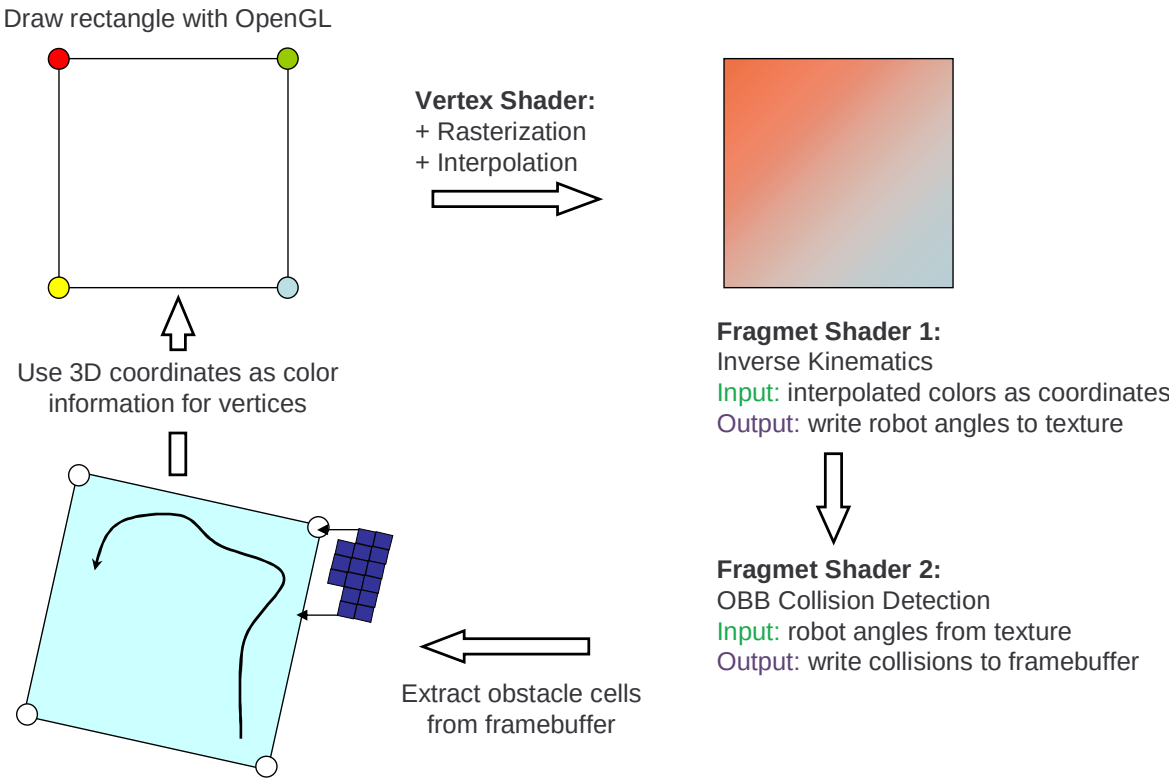


Figure 5.18: Generate virtual obstacles with OpenGL pipeline

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 by using the snap mode (cf. section 3.4.2).

5.8 Results

Before we present the results of applying the architecture mentioned above, we will review some ineffective attempts, which nonetheless contributed to our knowledge about this methodology. Our first approach to derive a decomposition of the presented skill into primitives was to exclusively apply unsupervised methods like cluster analysis. We have tried to identify certain events in the trajectory and use them as dissection points, starting with digital events, like the gripper state or force on/off, which can be uniquely defined and are easy to detect. The problem was that derivations merely relying on these events did not yield a sensible dissection of the skill into primitives (the segmentation was too coarse). In addition, results have often been flawed by improper performance of the user (e.g. unintentional opening or closing of the grippers). This, amongst others, brought us to the conclusion that the user has to follow a certain policy regarding the demonstration of a skill. This policy was finally realized by the abstract task patterns presented above. Beside the digital events, we have also utilized threshold-based events, like movement clusters or curve properties of the trajectory, as dissection points. This time, things went even worse, because the quality of dissection was dependent on the determination of the corresponding threshold (e.g. bending angle of the trajectory). In addition, these parameters have been flawed by tremor of the user. Based on

these experiences, we refrained from unsupervised dissection of the trajectories. Nonetheless, we still employ both digital and threshold-based events as hints for primitive derivation in our architecture, but their significance is only valid in connection with the tasklet definition in the knowledge base. This allows for a definition of events based on fixed thresholds. As mentioned above, we have set the thresholds of the movement clusters and turning points to a value, which guarantees detection of all significant instances (sensitivity of 100%). This comes at the price of a comparably low specificity, which leads to the detection of many insignificant threshold-based events. Therefore, events are first rendered significant after applying the corresponding tasklets as patterns.

Another problem we encountered during our experiments was the aberration of the particle from the intended path during fluid simulation. This was due to the fact that the speed of the particle has not been set appropriately and it was not possible to stop the particle at the right moment. Therefore, a particle, which was traveling too fast, was often drawn into a vortex, as depicted on the left side of fig. 5.19. Later, we have suppressed this behavior by preventing the fluid from forming small vortices. This was achieved by decreasing the Reynolds number of the fuid simulation and by exciting the fluid for a longer period at the same place. The latter refers to the simulation of stirring, which is usually concentrated to one point of the fluid (i.e. stirring with an infinitesimal small stick). Since this does not emulate a natural behavior we have used a whole path of stirring instead of a single point, i.e. the fluid was excited at many successional points on this path. This group of points was smoothly shifted along the path during simulation. The other problem, stopping the particle at the right time, was solved by defining an outflow condition at the position where the path of stirring has ended. In order to ensure smooth transitions, the outflow is faded in by an sigmoid function at the end of the stirring. The rightmost picture of fig. 5.19 shows a visualization of the outflow and the correctly instantiated trajectory.

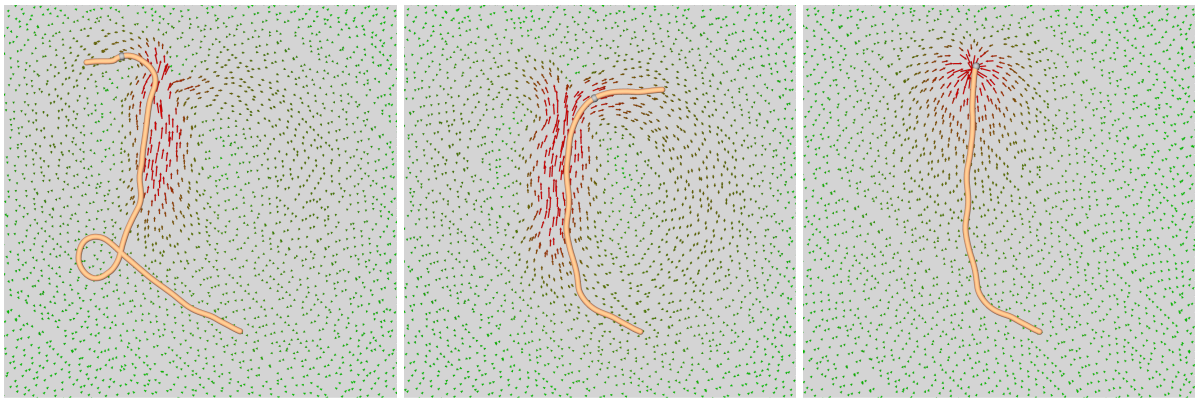
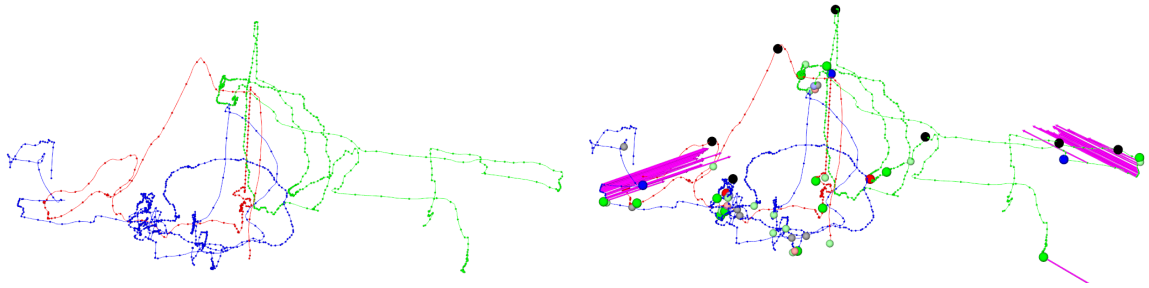


Figure 5.19: Different results of the fluid simulation for instantiation of the same tasklet

There have also been some challenges regarding the implementation of the inverse kinematics and collision detection on the GPU. The compilation and execution of shader programs is not standardized as expected from usual programming languages. For example the *atan2*-function, which is employed for the inverse kinematics, is implemented with different precisions on different graphics cards. Therefore, some cards with a low-precission implementation are not capable of providing accurate solutions. In addition, the assembler language of the GPU allows no code jumps or loops. Therefore, all jumps and loops in the high level language (e.g. GLSL or Cg) are unrolled to plain code. This enlarges the size of the assembler code dramatically, and thus, the limited code storage of the card

easily produces an overflow. In order to avoid this, we have separated the inverse kinematics from the collision detection and distributed them to two different shader programs. However, it would be favorable to have both parts in one program. Therefore, we intend to merge them again, as soon as graphics cards are available featuring enough code storage memory. Most cards also pose the limitation that the applied texture size has to be a power of two. Meanwhile, extensions of the shading language also allow for odd-sized textures, but according to our experience this feature significantly reduces performance.

After solving these initial problems, we have successfully transferred a knot-tying skill to our system. Afterwards, the corresponding task was performed within a new environment (i.e. an appropriate skill was generated out of the knowledge base of the system). The decomposition of the user demonstration starts with the detection of significant events (cf. fig. 5.20).



(a) Original trajectory (b) Detected events

Figure 5.20: Initial detection of significant events

Afterwards, the events are matched against the events delimitating tasklets in the task definition. The task definition is depicted again in fig. 5.21 where the tasklets are numbered in ascending order. The numbering is composed of the track number (track 1 = right hand, track 2 = left hand) and an ascending tasklet number for each track. These numbers can be recognized by the labels of the individual steps of the segmentation algorithm (cf. fig. 5.22). In addition, the colors used for the illustration of single primitives of the demonstration match with the corresponding colors of the tasklets in fig. 5.21.

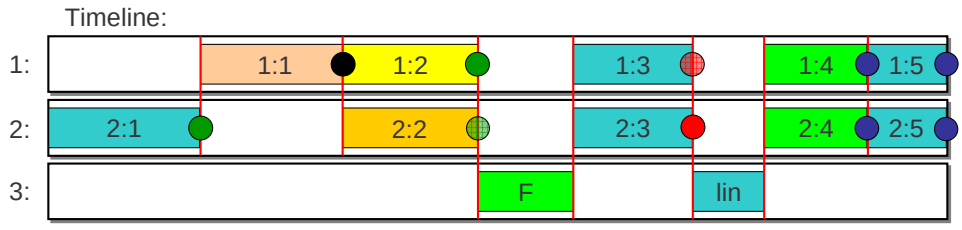
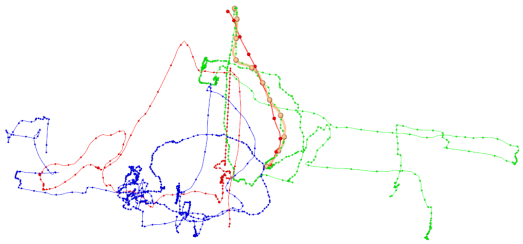


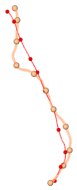
Figure 5.21: Definition of task employed as pattern for decomposition

After at least one successful demonstration has been decomposed into primitives and the corresponding information is extracted as described above, the task can be potentially instantiated at an arbitrary point in the environment. As mentioned above, the instantiation of the task is carried out step by step as prescribed by the definition (cf. fig. 5.21). Again, the colors of the illustration match with the colors of the task definition (cf. fig. 5.23). After the task is instantiated, a preview can be displayed in the simulation environment. Afterwards, the task is carried out and the user can continue with manual

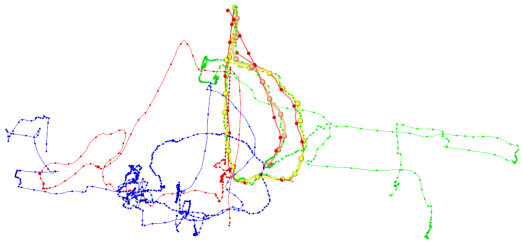
operation. The knot-tying performance depicted in fig. 5.22 ff. was taken from a series of fifteen passes of the complete procedure. Ten of them have yielded a completed knot, while five of them failed at a certain stage of the process. The reasons of failure are summarized in section 6.2. However, given the circumstances of the experiments (custom-made system and novel methodology) the success rate is comparably high. If we compare the initial user demonstration to the outcome of the instantiation, the significant contribution of this methodology becomes clear (and its advantage over mere teaching). The unstructured information contained in the trajectories of the demonstration has been transformed into a compressed, structured and adjustable representation.



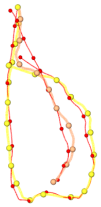
Track 1, primitive 1 (with trajectory)



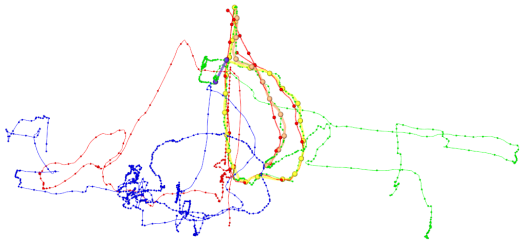
Track 1, primitive 1 (stand-alone)



Track 1, primitive 2 (with trajectory)

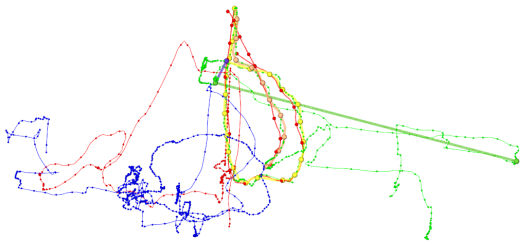


Track 1, primitive 2 (stand-alone)

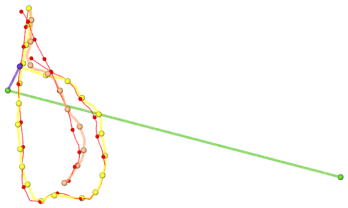


Track 1, primitive 3 (with trajectory)

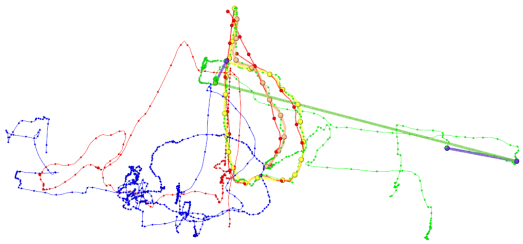
Track 1, primitive 3 (stand-alone)



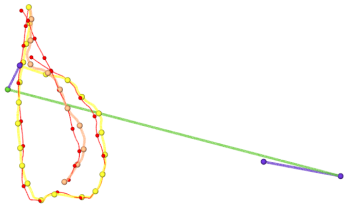
Track 1, primitive 4 (with trajectory)



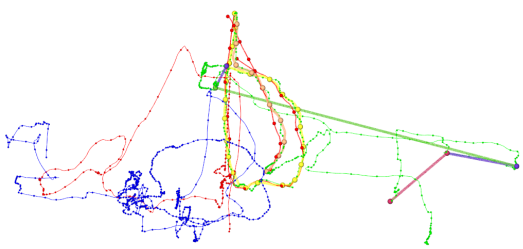
Track 1, primitive 4 (stand-alone)



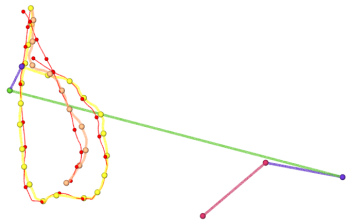
Track 1, primitive 5 (with trajectory)



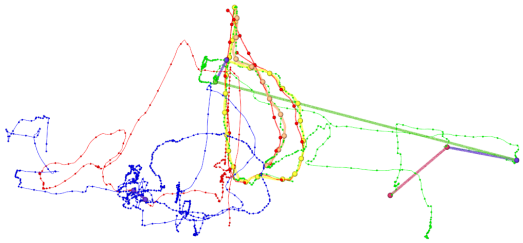
Track 1, primitive 5 (stand-alone)



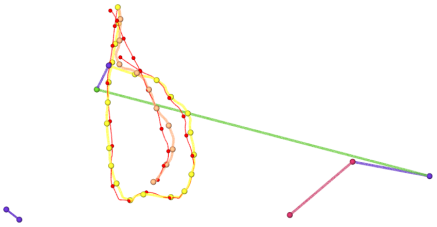
Track 1, primitive 6 (with trajectory)



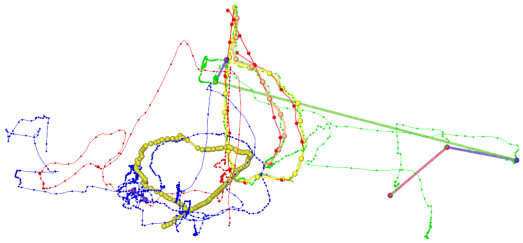
Track 1, primitive 6 (stand-alone)



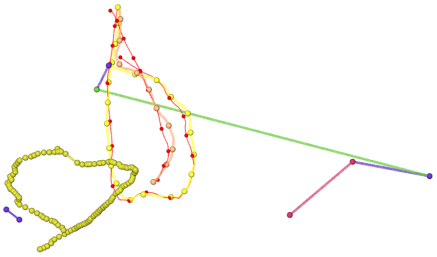
Track 2, primitive 1 (with trajectory)



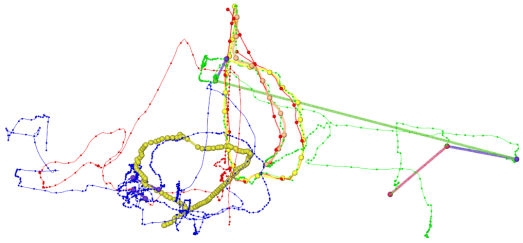
Track 2, primitive 1 (stand-alone)



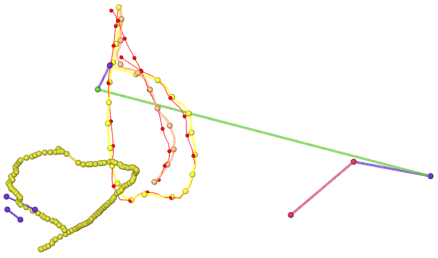
Track 2, primitive 2 (with trajectory)



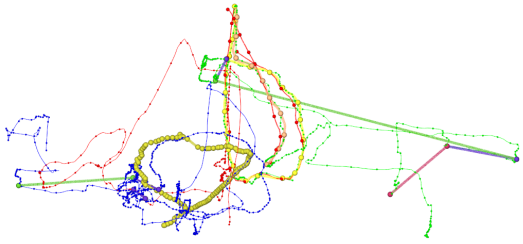
Track 2, primitive 2 (stand-alone)



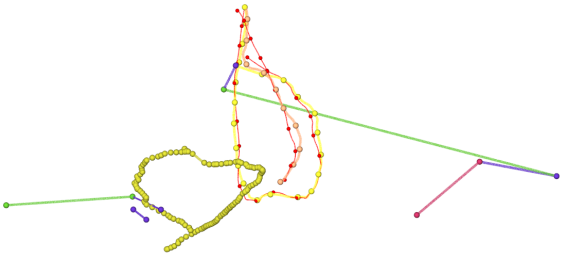
Track 2, primitive 3 (with trajectory)



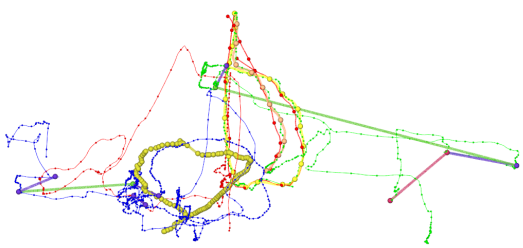
Track 2, primitive 3 (stand-alone)



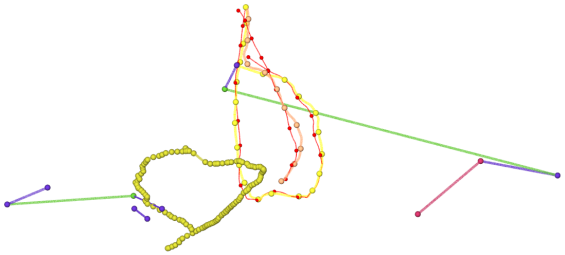
Track 2, primitive 4 (with trajectory)



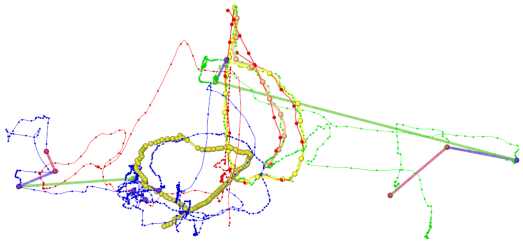
Track 2, primitive 4 (stand-alone)



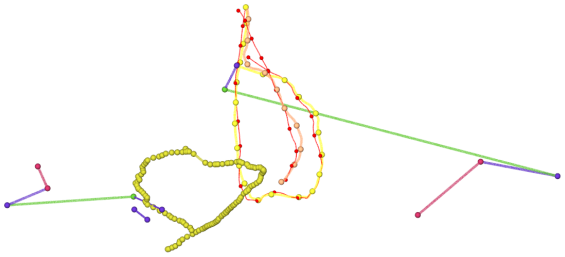
Track 2, primitive 5 (with trajectory)



Track 2, primitive 5 (stand-alone)

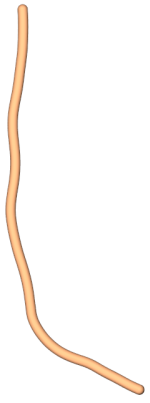


Track 2, primitive 6 (with trajectory)

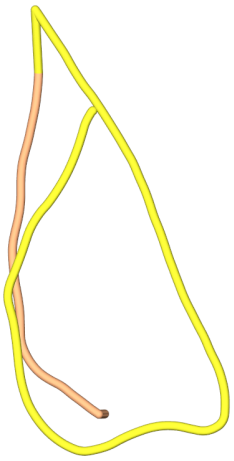


Track 2, primitive 6 (stand-alone)

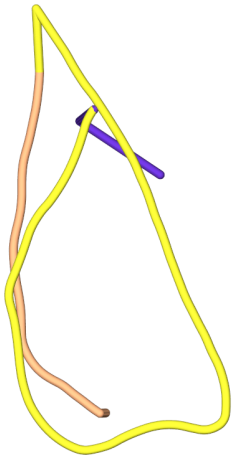
Figure 5.22: Illustration of subsequent steps of the segmentation algorithm



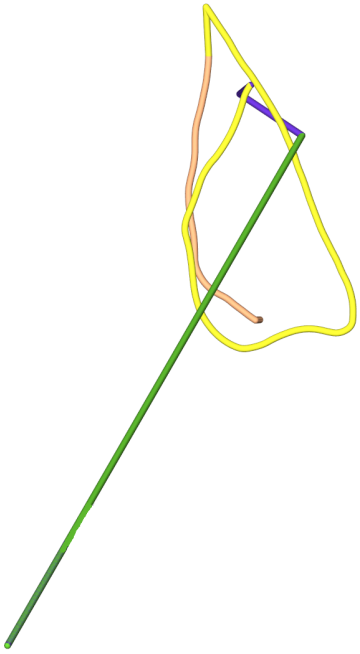
first 2D primitive (right hand)



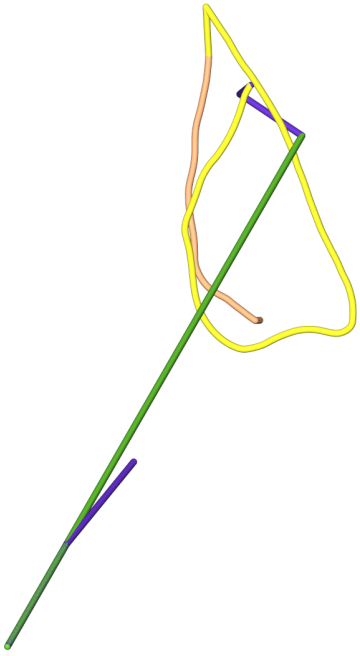
second 2D primitive (winding)



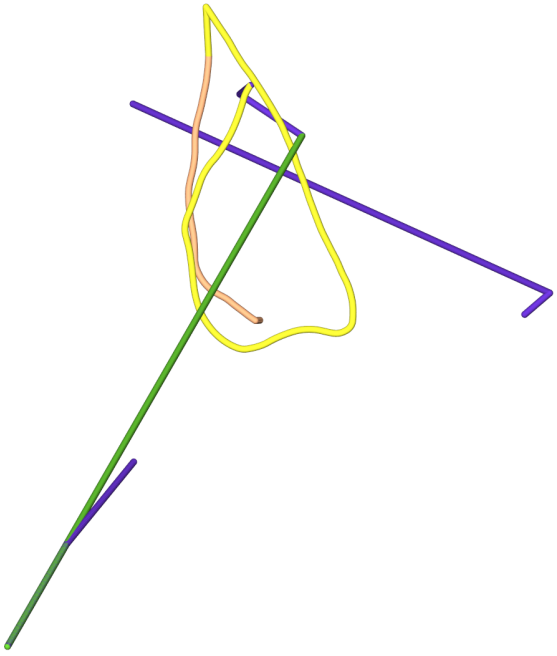
linear movement



force controlled movement (tighten knot)



linear movement (to end position)



first primitive of the left hand

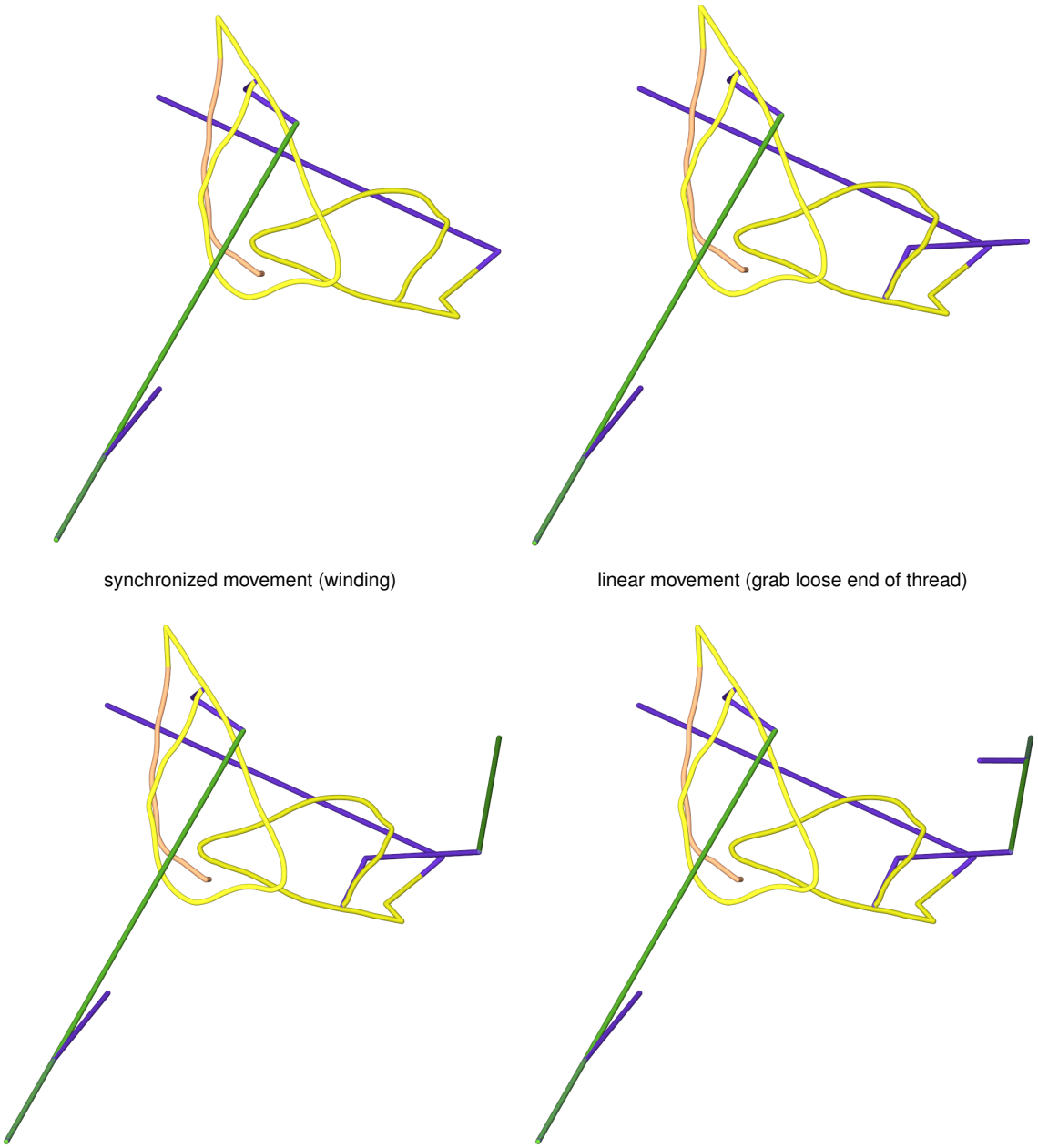


Figure 5.23: Application of the task in new environment

Chapter 6

Conclusion

In this last chapter we want to review again the presented work, and point out its novelties and its contribution to the research on the field of human-robot skill transfer. In addition, we discuss current limitations of our methodology and clarify their reasons. Finally, we specify some options of how to improve future versions of the system.

6.1 Summary

By this work we have presented an experimental device for robotic heart surgery. The main purpose of the system was to evaluate integration of force feedback into surgical procedures and to assess, to which extent automation can be transferred to such systems. The hardware consists mainly of ready-made parts, which have been adapted to our needs. The patient-side slave comprises four manipulators, each of them composed of a *Mitsubishi* robot and an *Endowirst* instrument, which was originally deployed with the *daVinci* system. The surgical instrument is connected to the robot 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 be equipped with a stereo camera instead of an instrument. The manipulators are mounted onto a gantry on the ceiling. The master console consists of a 3D display based on polarized screens. The input modality is formed by two *PHANToM* devices, which can also feed back forces to the user. The exerted forces can either be generated within the system by haptic rendering or they originate from amplified measurements at the end effectors. Therefore, we have equipped the minimally invasive instruments with strain gauge sensors, which are attached to the distal end of the instrument's shaft via adhesive bonding. Forces are measured in three spatial directions. Due to the high axial stiffness of the shaft, sensor readings measured along the shaft have been omitted from further examinations. The quality of force feedback and the applicability of the system for real-world operations has been tested in two large-scale evaluations (the first evaluation was attended by 25 surgeons, while the subjects of the second one have been 15 surgeons and 15 students - this number is comparably high regarding this type of evaluation).
Besides evaluation, another object of research, and the main focus of this thesis, was the development of software for robotic surgery. The software mentioned in this thesis can be chiefly divided into three categories. The first is predominantly concerned with control strategies for such systems, comprising a low-level interface to the devices, the inverse kinematics and a real-time interface. A second part is concerned with offline path planning, which is realized via a key-framing module and a simulation en-

vironment embedded into a sophisticated graphical user interface. The mentioned software modules are interconnected with the third part, which is the most important one regarding the relevance of the thesis, because it contains the implementation of skill transfer strategies for teleoperated systems.

Our application scenario, regarding automation and skill transfer, was minimally invasive knot-tying. This task yields an extent of complexity, which requires new strategies for structuring and transferring information. We have reviewed several established methodologies including Hidden Markov Models, Kalman filtering, neural networks and support vector machines, but none of them has met all the requirements we posed on the desired method. All those mentioned are powerful methods on their part and would have been certainly able to solve some parts of the challenge, but we refrained from constructing a patchwork solution, which contains a variety of interconnected methods. Instead, we wanted to present one consistent approach. We have based this approach on current psychological research on human communication and teaching. A methodology, which is particularly interesting for human-robot skill transfer, is situated learning, or more precisely, the scaffolding method. One of the fundamental ideas of situated learning is the teacher and trainer sharing the same environment. This also includes having the same sensori-motor capabilities in order to interact with the environment. For example, a skilled master can successfully teach an apprentice, only if he is aware of the probably restricted capabilities of the apprentice. This has always been a particular issue in human-robot skill transfer. Tasks, which are easily performed by humans, are major challenges to be realized on robotic platforms. For example, rotating an object with the fingers of one hand yields no difficulties for humans, but the same task can hardly be implemented into robotic hands. This is mostly due to the lack of sensor information and to an inflexible sensori-motor interconnection in present day robots. Therefore, situated learning (i.e. the awareness of different capabilities of teacher and trainee) is a reasonable starting-point for human-robot skill transfer. Closely linked to that paradigm, the basic idea of scaffolding, is the teacher utilizing his superior knowledge about the task to provide a framework of hints in order to assist the trainee. On the one hand, this framework should be powerful enough to significantly increase the learning rate. On the other hand, the trainee should not be spoiled, but forced to make own efforts to generalize the teacher's demonstrations.
We have realized situated learning and scaffolding by means of a skill transfer architecture, which is based on abstract descriptions of tasks. Those are provided by the user and become an intrinsic part of the knowledge base of the system. The actual skill transfer is carried out via learning by demonstration, which by itself is an instantiation of situated learning. Our first experiments have shown that learning by demonstration alone cannot solve the underlying problem of properly structuring information (decomposing the demonstration into meaningful generalizable primitives). Therefore, this method was augmented by a pattern matching algorithm, which works on trajectories. The provision of this algorithm and the corresponding patterns, namely the task definitions, constitute the scaffolding in our architecture. The most complex tasklet in this scaffolding framework is the 2D movement. We have implemented its instantiation by means of fluid simulation. The movements of the demonstrations are interpreted as the stirring of a fluid. If the corresponding tasklet is applied to a previously unknown environment, a particle is thrown into the stream, generating a path for the end effector. In addition, obstacles occurring in the new environment are inserted into the simulation. In order to allow for collisions outside the actual simulation area, we have introduced the concept of virtual obstacles. Since this concept is computationally expensive, we have realized an implementation, which runs on the GPU of the graphics card. Finally, we have successfully applied this skill transfer architecture to the real-world application of robotic knot-tying.

6.2 Limitations of the proposed methodology

As mentioned in the results section 5.8, we have successfully performed several instances of a surgical knot. The experience acquired during these procedures has disclosed some of the limitations of the currently applied methodology: It can only be employed for rather short action sequences, which can be decomposed by the user into meaningful sections. If the framework should be applied to longer, more complex skills, e.g. a complete surgical intervention, the user might not be able anymore to structure the task in a granularity, which is required for a transfer to our framework. Concerning applicability, the most important prerequisite is a deterministic decomposition of the skill into rather simple primitives. Unfortunately unambiguousness can hardly be reached in complex interventions. For example, there might be thousands of possibilities to perform a complete resection of an organ. In addition, it would be very cumbersome to decompose such complex sequences into 2D movements or other simple primitives. In order to process long sequences with our framework all the same, it would be necessary to introduce an intermediate layer of abstraction. At least, this layer must provide a possibility for user inputs on a higher level of granularity. For example the user could interact with the system verbally. Afterwards, the spoken information, e.g. “make a knot”, must be automatically transferred into a lower level description based on tasklets. However, as discussed above, it is hard to find a method for fully automatic decomposition of skills into meaningful primitives.

Apart from applications of our framework to other domains, there are also some limitations regarding the knot-tying procedure. As mentioned above, the success rate for completely performed knots was approx. 65%. Most of the failures occurred while the knot was instantiated comparably far away from its original demonstration. In practice, this point of instantiation is restricted by the calibration of the system. Unfortunately, many error sources contribute to a comparably bad calibration of our system. First of all, the absolute accuracy of the robots themselves is limited to 1-3 mm, depending on the individual robot. In addition, the mounting posture of the robots is not known exactly and can hardly be measured. Particulary the coplanarity of the robot’s base relative to its attachment cannot be guaranteed. Another error source is the mountings of the instruments and the instruments themselves. As depicted in fig. 6.1, the possible play of the end effector is quite large ($\pm$ 1.5 cm). The biggest contribution to these errors is due to the flexibility of the instruments’ shafts. In addition, the magnetic clutch exhibits some play, which is amplified by the long shaft, and the grippers at the distal end of the shaft exhibit some play, too.
While the skill application described above was derived from one single demonstration, which actually was the goal of our research, we have also tried to base the skill transfer on multiple demonstrations. This has yielded no great difficulties or surprising outcomes regarding the elementary primitives like linear movements or force controlled movements. It simply leads to an expected averaging of the corresponding parameters (force, start and end points). We have also tried to base the application of 2D movements on multiple demonstrations, but stopped the experiments at an early stage, since the necessary dependencies of subsequent tasklets were no longer preserved, but disturbed by averaging. For example, the geometrical dimension of the second 2D primitive (winding) is dependent on the preceding placement movement. If this dependency is destroyed by averaging, a knot will no longer be produced as an outcome of the movements. Therefore, we have restricted further research to the application of the synchronized primitive. As mentioned above, we have initially employed an affine transformation to perform the mapping between right hand and left hand. For a single demonstration this procedure worked quite well. Afterwards, we have tried to derive the interdependency from multiple

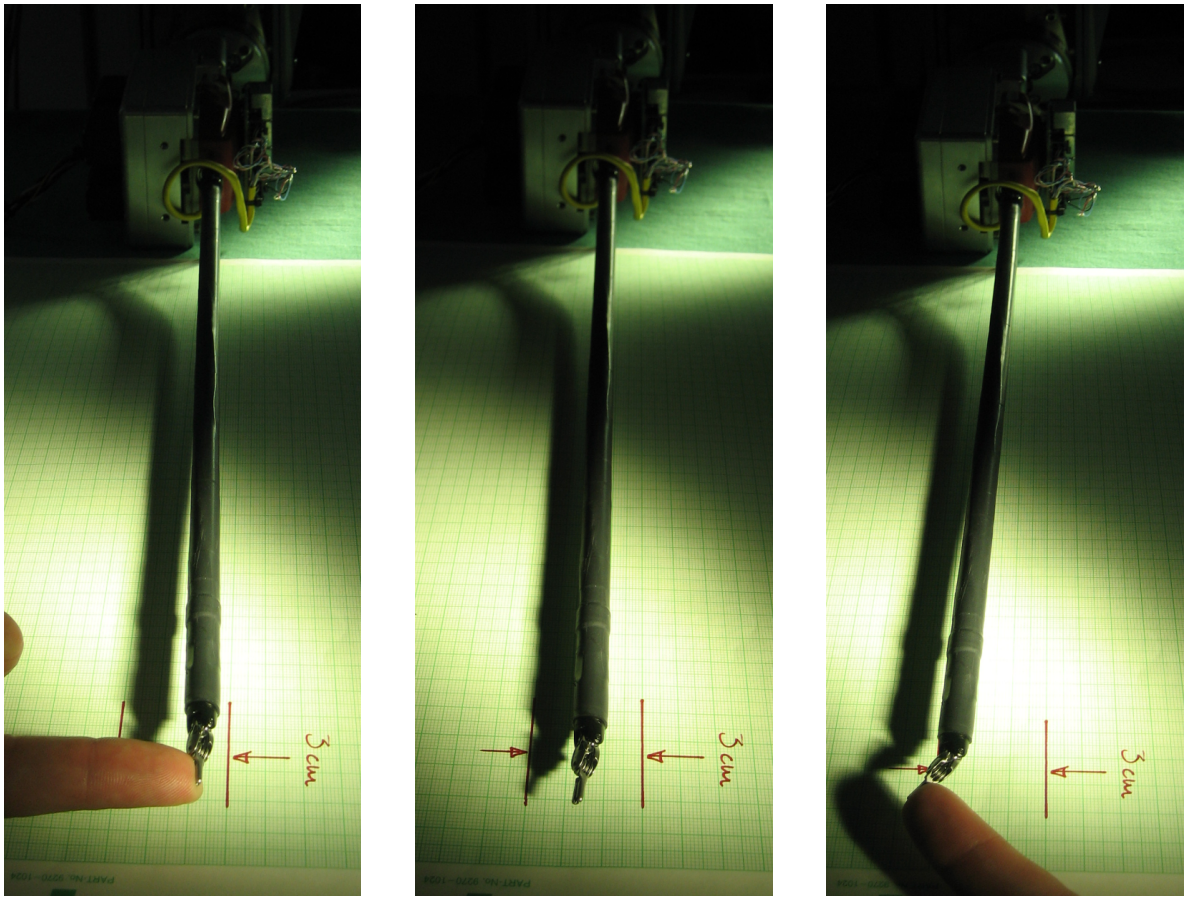


Figure 6.1: Aberrations due to flexibility and play

demonstrations and compared the outcome to the ones of a neural network. As one can see from fig. 6.2 and fig. 6.3 the result of the affine transformation gets quite bad after using two demonstration examples, but recovers as soon as the number of demonstrations is increased. However, the outcome of the neural network (we have employed an LSTM network as described above) outperforms the affine transformation by a factor of ten regarding the mean square error (MSE) between original data and the output of the mapping function. This is most probably due to the mapping between right hand and left hand being nonlinear.

6.3 Contribution of this thesis

Some parts of the work mentioned in this thesis have already been submitted to international conferences in order to get an external peer review, but major parts, in particular the skill transfer architecture, the GPU computing of virtual obstacles and most parts of the control software, are unpublished so far. The first version of our system set-up has been published in [15] and, more detailed, in [27] and [19]. The second version of our hardware was first introduced to the research community in [26] and [10]. Different aspects of the force feedback technology has been presented in [25, 29, 22]. The evaluation of haptic feedback in robot-assisted minimally invasive surgery was published in [7, 3, 12, 13] and, more detailed, in [2]. The inverse kinematics, the only part of the low-level control software published so far, was presented in [16], while the graphical user interface

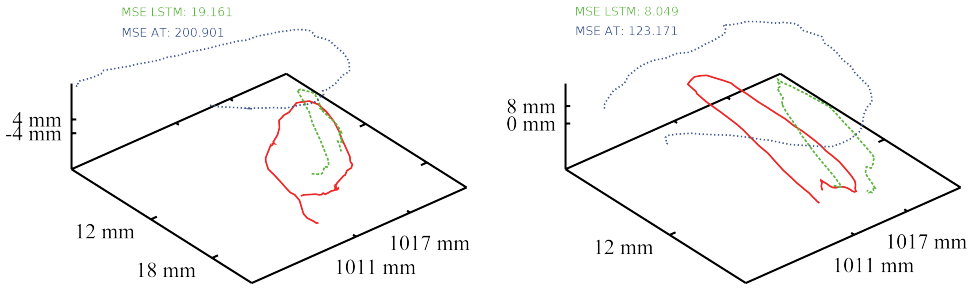


Figure 6.2: Generalization based on two demonstrations (red: recorded trajectory of left hand, dotted blue: affine transformation, dashed green: neural network) [*original data recorded by H. Mayer, neural network output by T. Gebhardt*]

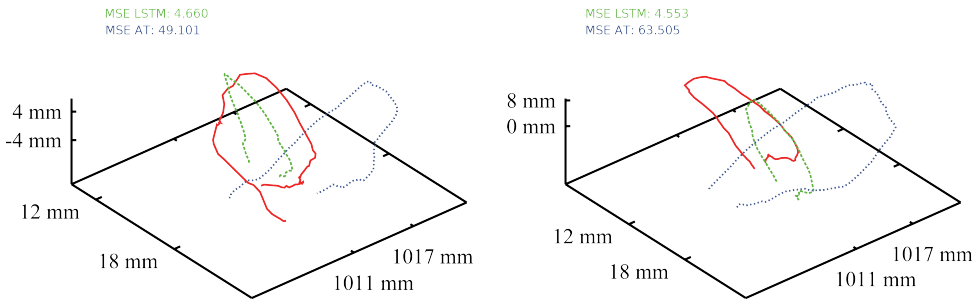


Figure 6.3: Generalization based on six demonstrations [*original data recorded by H. Mayer, neural network output by T. Gebhardt*]

was introduced at the IASTED HCI conference [18]. The rest of the software (cf. chapter 3) has not been published so far. Closely related to this work are some publications on medical image processing by I. Nagy [163, 31]. First approaches to human-robot skill transfer have been published in [23, 30, 21]. A proposal for an improvement of the force feedback, which has not been realized so far, was presented in [24] (decoupling of force measurement and driving the instrument).

In our opinion, the major contribution of this thesis is the utilization of scaffolding (which is an established paradigm of human learning) for human-robotic skill transfer. To our knowledge, a comparable extension to learning by demonstration has not been published so far. Indeed, some other authors have already referenced scaffolding with respect to robotic learning, but they have employed it in a completely different context. For example Saunders et al. [188] have utilized scaffolding to weight state vectors which a mobile robot is acquiring during exploration of its environment. Scaffolding is used to tell the robot that in some situations the input from certain sensors is more important than the one of others (e.g. dominance of visual input over range sensors). In [185] and [66] the authors refer to scaffolding with respect to the social environment where learning takes place. In contrast, our approach proposes the utilization of task definitions as patterns for the classification and structuring of data in human demonstrations. As presented above, this yields a consistent method to incorporate user data into the system's knowledge base in order to enable instantiations of tasks in previously unknown envi-

ronments. The instantiation of tasks was implemented, amongst other methods, by application of fluid dynamics. While some authors already have employed static streamline functions [77, 121], this was, to our knowledge, the first time to utilize present day's computing power for a dynamic fluid simulation, which can adjust itself to different environments and which can be used to generalize user demonstrations of various skills. In comparison to other path planning methods, namely obstacle avoidance by potential fields, this methodology yields the advantage of non-oscillatory behavior. In addition, path planning will not get stuck in dead ends and allows movements parallel to obstacles. While some of these advantages have already been included into the static solutions of the authors mentioned above, our approach is also capable of generalizing self-crossing trajectories, a feature not possible with static streamline functions. Our principle idea has been published in 2007 at the International Conference of Robotics and Automation (ICRA) and was awarded with the *Best Manipulation Paper* prize [20]. By this thesis, the real-world application of this principle is published the first time, together with a novel approach of speeding up generation of virtual obstacles (see above) by employing the GPU.

best paper award

6.4 Outlook

During the second evaluation of our system we have tried to perform more complicated tasks like suturing a cut, closing an atrial septum defect (simplified version) and replacing a papillary tendon. Although these tasks have been successfully performed with our system, they have also revealed some deficiencies. Most subjects complained about the hypersensitive reaction of the input devices, i.e. most of them would have favored a more inertial behavior. Therefore, we think of implementing an artificial inertia, which could be achieved by means of the force feedback capability of the input devices (at least for the first three joints, which are connected to servo motors). Another shortcoming, which was exhibited during evaluation, was the digital closing of the gripper. Needless to mention that it would be more benefiting to provide continuous closing. In order to achieve this, we are planning to include a haptic gripping system, which is currently developed at the German Aerospace Center (DLR).

Regarding the software of the system, there are also some unresolved issues. One is concerning the fluid dynamics, which is currently realized by one single thread of the program. Due to the nature of the algorithm, it would be possible to reach quasi arbitrary parallelization by subdividing the simulation grid. Implementing fluid dynamics by means of a multi-threaded application would speed up computation times significantly. The planning and interaction tools of the system are currently more or less specialized to our application. Therefore, we want to generalize it towards a multi-purpose planning platform, which can be employed for a variety of robotic applications. Generalization is also an issue regarding the skill transfer architecture. Although in principle being designed to handle arbitrary skill transfer tasks, we have only tried our application example of automated knot-tying so far. It would be exciting to use the platform for entirely different applications like automotive welding tasks or intelligent part feeding. Another improvement can be applied to the scaffolding framework of our software. The original definition of scaffolding also comprises the option to gradually remove hints from the knowledge base, once the system is smart enough. A first step towards this could be the removal of the assistant gripper, which is part of the scaffolding for the transfer of the knot-tying task.

Appendix A

Appendix

A.1 Conversion between Coordinate Systems

The conversion between coordinates of an object X, denoted relative to the basis frame 0 of our system, and the same coordinates given in an individual robot frame R can be found by:

$$ {}^{R}_{X}T = {}^{R}_{R'}T \cdot {}^{R'}_{0}T \cdot {}^{0}_{X}T; \qquad {}^{R}_{R'}T = \begin{pmatrix} 0010 \\ 1000 \\ 0100 \\ 0001 \end{pmatrix} $$

where ${}^{0}_{R}T = {}^{0}_{R'}T \cdot {}^{R'}_{R}T$ is the position of the robot relative to the basis system, and therefore, ${}^{R}_{0}T$ is the corresponding inverse. This method is provided in function `setWorldMatrix` of class `Mitsubishi6SL`. Accordingly, the conversion from a given robot frame into the basis frame is calculated by:

$$ {}^{0}_{X}T = {}^{0}_{R'}T \cdot {}^{R'}_{R}T \cdot {}^{R}_{X}T $$

A.2 Forward Kinematics

The forward kinematics of a robot describes the mapping of its joint angles onto a Cartesian position of its end effector. Starting at the initial robot frame, this result can be calculated by subsequently multiplying transformations, which are derived from the Denavit-Hartenberg (DH) parameters of the robot. The table below shows the DH parameters of the *Mitsubishi MELFA 6SL* robot:

i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	350	θ_1
2	-90	85	0	θ_2
3	0	380	0	θ_3
4	-90	100	425	θ_4
5	90	0	0	θ_5
6	90	0	85	θ_6

Therefore, the posture of the end effector, with respect to the robot frame, can be easily determined

by the following transformation:

$$ {}^{R}_{F}T = {}^{R}_{6}T = {}^{R}_{5}T \cdot T_{\alpha 5} \cdot T_{a5} \cdot T_{d6} \cdot T_{\theta 6} \tag{A.1} $$

where $T_{\alpha 5}$, T_{a5}, T_{d6} and $T_{\theta 6}$ are the transforms derived from the table entries and ${}^{R}_{5}T$ can be determined by applying the right hand side of equation A.1 recursively.

A.3 *Kuka* Inverse Kinematics

Since low-level Cartesian control of the *Kuka* robots is not possible, but only manipulation of joint angles, we have to determine the inverse kinematics of the *Kuka* robots. Therefore, given the homogeneous transform matrix of the robot's flange, we have to find a mapping to extract the joint angles. This is best be done by addressing the first three degrees of freedom separately.

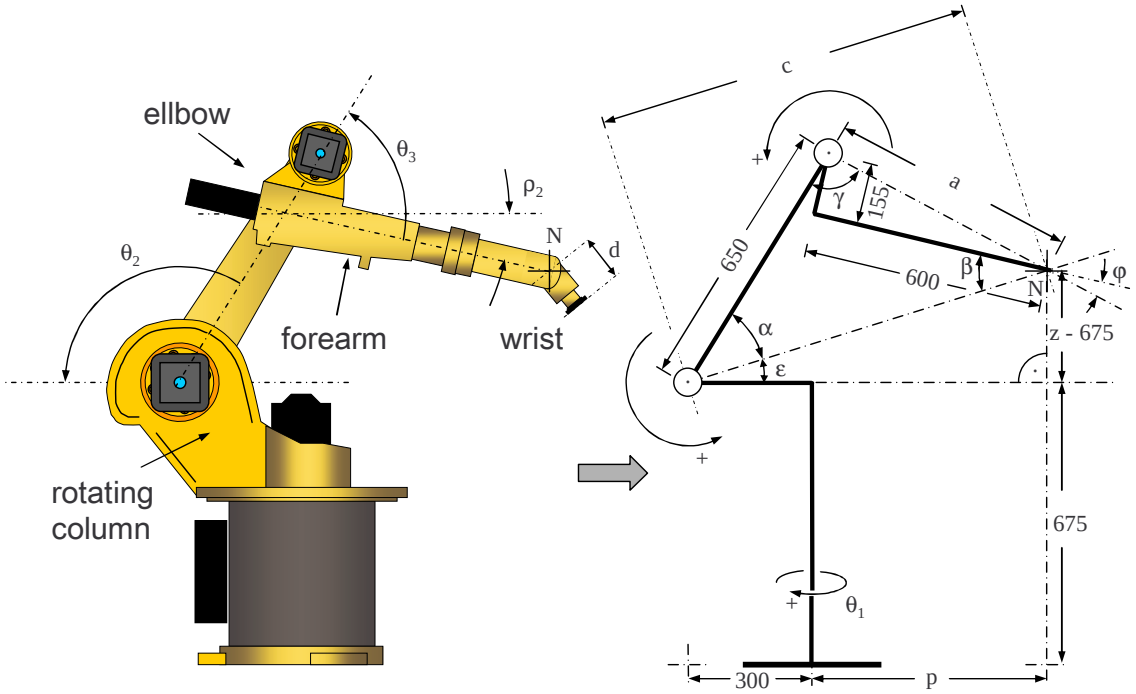


Figure A.1: *Kuka* robot geometry

The joint angles of the first three hinges uniquely determine the position of point N (cf. fig. A.1). Note that uniqueness does not apply for the other direction, since the position of N can be reached by several joint angle configurations: We can choose to turn the rotating column by $180°$ or flip the elbow. For our setup it was most convenient to turn the rotating column by $180°$ (left-handed configuration) and move the elbow up. This can be seen in figure A.1. Given the transform of the robot's flange, we have to move back in z_F-direction (i.e. normal to the flange) by the length d in order to get the position of N (see figures 3.15 and A.1). Consequently, we get:

$$ N = {}^{K}_{F}T \cdot \begin{pmatrix} 0 \\ 0 \\ -d \\ 1 \end{pmatrix} $$

$$\begin{aligned} N_x &= -d \cdot {}^K_F T_{02} + {}^K_F T_x \\ N_y &= -d \cdot {}^K_F T_{12} + {}^K_F T_y \\ N_z &= -d \cdot {}^K_F T_{22} + {}^K_F T_z \end{aligned}$$

Given the position of N, we can derive the values of the first three joint angles $(\theta_1, \theta_2, \theta_3)$. The angle which is easiest to determine is θ_1. It can be calculated by:

$$\theta_1' = atan2(N_y, N_x)$$

As mentioned above, we additionally want to turn the rotating column by $180°$. Therefore, in order to get θ_1, we add $180°$ if $\theta_1' < 0$. Otherwise, if $\theta_1' \geq 0$, we subtract $180°$. Angles θ_2 and θ_3 are more difficult to figure out. We need a detailed plan of the robot's geometry (see figure A.1 right hand side: all measurements are given in millimeters). First of all, we identify angle ϵ:

$$\epsilon = atan2(z - 675, p + 300);\ p = \sqrt{N_x^2 + N_y^2}$$

For calculating α and γ we will employ the three-side formula for triangles with oblique angles. Therefore, we need to know all three sides (a, b and c) of the triangle first. One side is already predefined: $b = 650$mm. The others can be found as follows:

$$a = \sqrt{155^2 + 600^2};\ c = \sqrt{(N_z - 675)^2 + (p + 300)^2};$$

Additionally, we introduce s (the half of the triangle's outline) and the radius of the inscribed circle r:

$$s = \frac{a + b + c}{2};\ r = \sqrt{\frac{(s-a)(s-b)(s-c)}{s}}$$

Now, we can determine the interesting angles of this triangle:

$$\alpha = 2\tan^{-1}\left(\frac{r}{(s-a)}\right);\ \gamma = 2\tan^{-1}\left(\frac{r}{(s-c)}\right);$$

Because θ_2 is measured between upper arm and forearm, and not between upper arm and line a, we need an additional angle for correction $\varphi = \tan^{-1}\left(\frac{155}{600}\right)$. Given all these angles, we can combine them to θ_2 and θ_3:

$$\theta_2 = \alpha + \epsilon - 180°;\ \theta_3 = \gamma + \varphi - 180°$$

Note that the direction of rotation is indicated by "+"-signs in figure A.1. There is no danger of any flip-over for the second and the third joint, because θ_2 and θ_3 always lie in between $0°$ and $-180°$.
In order to determine the remaining angles, we need to know the transformation matrix of the forearm in point N. We get this homogeneous transformation matrix ${}^K_N T$ by rotating the robot frame about its z_K-axis by $\rho_1 = -\theta_1$. For orientation of the robot frame see 2.1. This gives use the transform of the robot frame after application of θ_1 (Note that the negative sign is due to the left-hand rotation of the first joint). After that, we turn the z_K'-axis, which still points to the ceiling, into the direction of the forearm. As one can derive from figure A.1, this can be achieved by a rotation of $\rho_2 = \theta_2 + \theta_3 + 90°$ about the y_K'-axis. Given ρ_1 and ρ_2, we can determine ${}^K_N T$ by using the formula for arranging homogenous transform matrices from Z-Y-X-Euler angles. In this case, the rotation about the x_K''-axis is zero.

$$ {}^{K}_{N}T = \left(\begin{array}{ccc|c} \cos(\rho_1) * \cos(\rho_2) & -\sin(\rho_1) & \cos(\rho_1) * \sin(\rho_2) & N_x \\ \sin(\rho_1) * \cos(\rho_2) & \cos(\rho_1) & \sin(\rho_1) * \sin(\rho_2) & N_y \\ -\sin(\rho_2) & 0 & \cos(\rho_2) & N_z \end{array} \right) $$

For further steps, we need to determine the orientation of the flange in relation to the orientation of the forearm. In other words we are looking for ${}^{N}_{F}T$. The easiest way to do this, is multiplying the inverse of the forearm transformation with the flange transformation: ${}^{K}_{N}T^{-1} \cdot {}^{K}_{F}T = {}^{N}_{K}T \cdot {}^{K}_{F}T = {}^{N}_{F}T$. Note that after applying ${}^{K}_{N}T$, the N_z-axis of the forearm system points towards the distal direction of the forearm. The rotation axes of the last three joints intersect in one point (see figure A.2). In their initial orientation, the rotation axes of the fourth and sixth joint are identical to z_N, while the fifth one is identical to y_N. Therefore, we can extract the corresponding angles from the transformation matrix by the rules of Z-Y-Z-Euler angles:

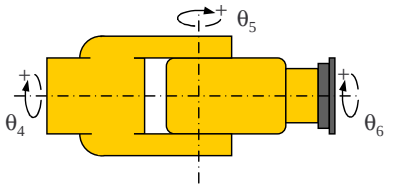


Figure A.2: Top view of the wrist

$$ \begin{aligned} \theta_4 &= -atan2({}^{N}_{F}T_{12}, {}^{N}_{F}T_{02}); \\ \theta_5 &= atan2(\sqrt{{}^{N}_{F}T^2_{20} + {}^{N}_{F}T^2_{21}}, {}^{N}_{F}T_{22}); \\ \theta_6 &= -atan2({}^{N}_{F}T_{21}, -{}^{N}_{F}T_{20}); \end{aligned} $$

The negative signs of θ_4 and θ_6 come from the left-handed rotation of the corresponding mechanical axes! Due to the fact that we operate the robot in a ”headlong” position, we have chosen a configuration with a reverted wrist in order to get smaller angles. Therefore, we additionally turn the fourth and sixth joint by $180°$ and drive θ_5 towards the opposite direction.

A.4 *Mitsubishi* Inverse Kinematics

Given the inverse kinematics of the *Kuka* robot, the inverse kinematics of the *Mitsubishi* robot can easily be derived by replacing the corresponding geometry (cf. fig. A.3). The distance d (cf. fig. A.1) between the plane of the flange (tool system) and the fifth joint of the robot amounts to $77mm$ for the *Mitsubishi* robot.

A.5 *PHANToM* Inverse Kinematics

After all, the *PHANToM* device constitutes the same kinematic chain as a standard industrial robot. A particularity is the length d between the flange and the fifth joint: Since positions refer to the fulcrum of the stylus, this parameter is set to zero. In addition, if the PHANToM is used upside-down, the Cartesian input has to be flipped about the z_P axis of the PHANToM frame ($180°$ rotation). This is simply achieved by inverting the signs of the entries in the first and second row of the corresponding transformation matrix. Afterwards calculations continue in a similar fashion as for the inverse kinematics of the Kuka, except for the assignment of the values depicted in A.4.

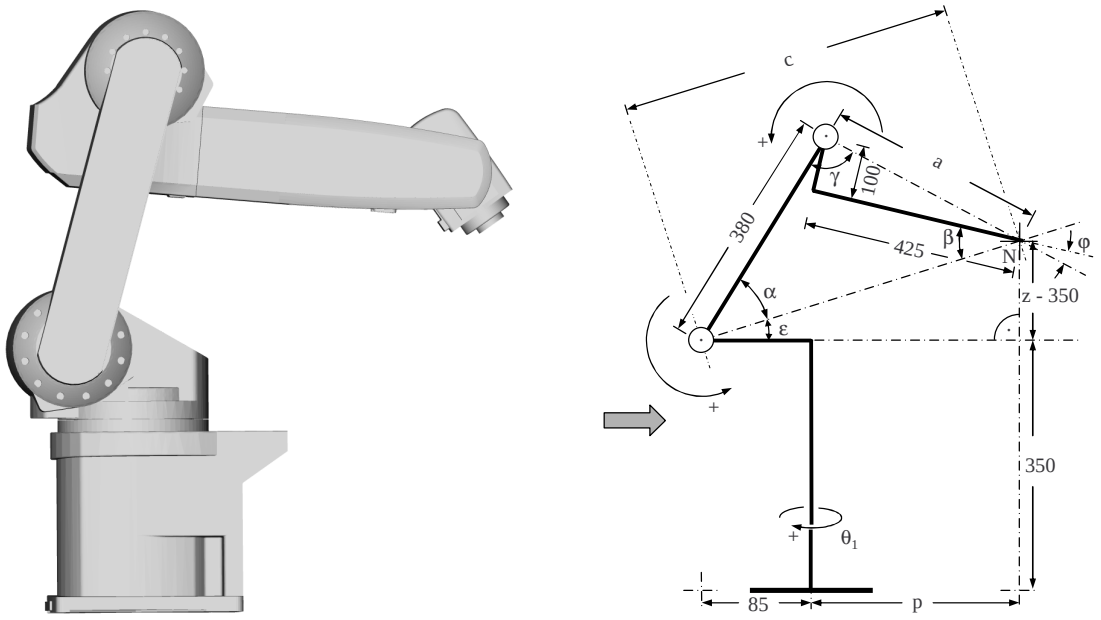


Figure A.3: Mitsubishi robot geometry

A.6 *PHANToM* Low-Level Interface

Since our control computer runs on 64bit Linux 2.6, the driver of the *PHANToM* device, which only supported 32bit Linux with kernel version 2.4 at that time, had to be ported to the newer 64bit version. In the new version of the Linux kernel the device file system is deprecated. It is supported only if the corresponding kernel option is selected and the corresponding layout of the driver has been modified. These modifications concern the functions `devfs_register` and `devfs_register_chrdev`, which are no longer defined in the kernel headers. In order to maintain the old driver code, we provide dummy versions of these functions (also for unregistering). In addition, the structure `pci_dev` is not valid any more and has to be replaced by the manually altered structure `pci_dev_new`. The driver now works with kernel version 2.6, but the modifications have a negative side effect: the physical address of the devices can no longer be detected automatically. They can be found by acquiring device information directly from the PCI bus. The most convenient way for doing this is the command `lspci -v` from the PCI utilities package for Linux. Running this command yields a list where all PCI devices and their properties are listed. Regarding the *PHANToM* cards, the PLX PCI brigde chip appears in the list. The entry of one of the chips will read as follows:

```
04:03.0 Bridge: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 01)
        Subsystem: PLX Technology, Inc. PCI-I04 PCI Passive PC/CAN
        Flags: medium devsel, IRQ 9
        Memory at b5affc00 (32-bit, non-prefetchable) [size=128]
        I/O ports at 8c00 [size=128]
        Memory at b5aff800 (32-bit, non-prefetchable) [size=256]
        Memory at b5aff400 (32-bit, non-prefetchable) [size=256]
        Memory at b5aff000 (32-bit, non-prefetchable) [size=256]
        Memory at b5afec00 (32-bit, non-prefetchable) [size=256]
```

The leading number code `04:03.0` says our chip is the third device on bus four. Therefore, we can

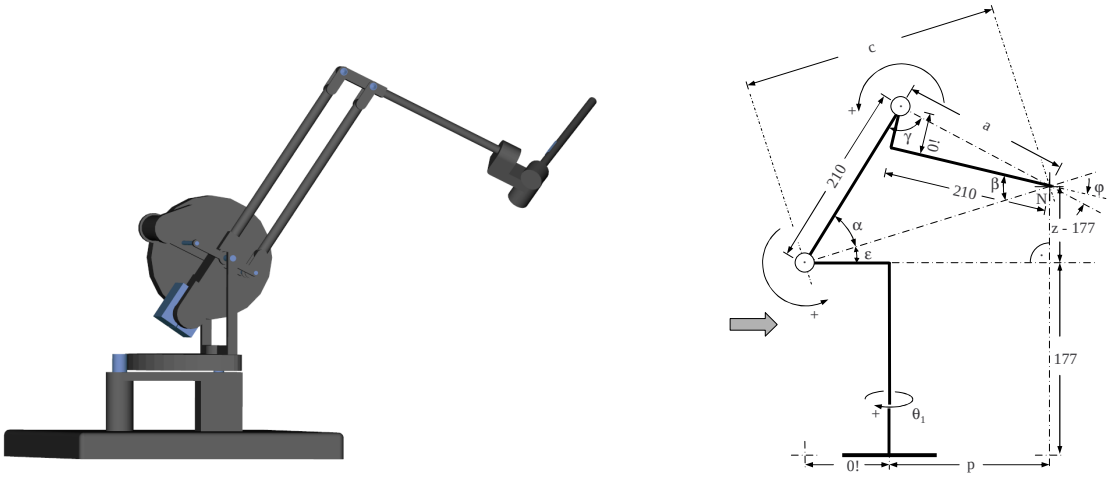


Figure A.4: *PHANToM* geometry

map the physical addresses of the first three memory regions of the chip when the driver is loaded, i.e. in function `phnpci_probe`:

```
if (dev->devfn / 8 == 3) phnpci->config = ioremap(0xb5affc00, 128);
if (dev->devfn / 8 == 3) phnpci->base[0] = ioremap(0xb5aff800, 256);
if (dev->devfn / 8 == 3) phnpci->base[1] = ioremap(0xb5aff400, 256);
```

where `(dev->devfn / 8 == 3)` selects the entries for the calling device (the third on the bus). We need to insert code in a similar way for the second *PHANToM* device. This resolves the first problem of migrating to kernel version 2.6, but we need additional modifications to get the driver working on 64bit systems. The first thing is to replace all word-width-dependent types like int and $long$ with unambiguous versions like $u32$. In addition, we have to provide the function `phnpci_compat_ioctl` in order to render the 64bit system able to interface with the original 32bit driver. Now both *PHANToM* devices can be located by the driver, and interrupts of the device will be handled correctly. The interface to the driver is implemented in the file `Phantom.cpp`. Data is transferred between hardware and driver by means of interrupts. Communication between driver and superordinate software is handled via `ioctl` calls, which will directly manipulate the device file (i.e. basically read encoder values from it or write force values to it). The function `getEncoderReading` uses the `ioctl` function of the driver to get the values of all six encoders of the device. These values are converted into joint angles by `getEncoderAngles` and those can be transformed into Cartesian positions by the inverse kinematics of the *PHANToM* (see above). Setting forces is explained in detail in chapter 3.

Authored and Co-Authored Publications

[1] R. Bauernschmitt, E.U. Braun, A. Knoll, H. Mayer, I. Nagy, and R. Lange. Implementation of force feedback into telemanipulated surgery: Assessment of surgical experience. In *Proceedings of the 33st Annual Meeting of Computers in Cardiology*, pages 301–304, Valencia, Spain, September 2006.

[2] R. Bauernschmitt, A. Knoll, H. Mayer, I. Nagy, E.U. Schirmbeck, S. Wildhirt, and R. Lange. Towards robotic heart surgery: Introduction of autonomous procedures into an experimental surgical telemanipulator system. In *Proceedings of the 3rd Annual Meeting of the German Society for Computer and Robot Assisted Surgery, CURAC [CD-ROM]*, Munich, Germany, October 2004.

[3] R. Bauernschmitt, E.U. Schirmbeck, C. Haßelbeck, F. Freyberger, H. Mayer, A. Knoll, S. Wildhirt, and R. Lange. Evaluation of haptic feedback in an experimental telemanipulator system. In *Proceedings of the IEEE 3rd European Medical and Biological Engineering Conference [CD-ROM]*, Prague, Czech Republic, November 2005.

[4] R. Bauernschmitt, E.U. Schirmbeck, A. Knoll, H. Mayer, I. Nagy, N. Wessel, S. Wildhirt, and R. Lange. Towards robotic heart surgery: Introduction of autonomous procedures into an experimental surgical telemanipulator system. *The International Journal of Medical Robotics and Computer Assisted Surgery*, 1(3):74–79, 2005.

[5] R. Bauernschmitt, E.U. Schirmbeck, H. Mayer, A. Knoll, S. Wildhirt, and R. Lange. Effects of force-feedback in robotic heart surgery: Dependency on the level of surgical skill. In *Proceedings of the Automed Workshop [CD-ROM]*, Rostock, Germany, March 2006.

[6] R. Bauernschmitt, E.U. Schirmbeck, H. Mayer, I. Nagy, A. Knoll, N. Wessel, S. Wildhirt, and R. Lange. Telemanipulator enhanced heart surgery: Implementation of autonomous procedures. *Biomedical Engineering*, 50(1):1260–1261, 2005.

[7] R. Bauernschmitt, E.U. Schirmbeck, I. Nagy, H. Mayer, and A. Knoll. Self-acting coronary artery detection as navigation aid in endoscopic heart surgery. In *Proceedings of the 10th Mediterranean Conference on Medical and Biological Engineering [CD-ROM]*, Ischia, Italy, July 2004.

[8] E.U. Braun, C. Haßelbeck, H. Mayer, F. Freyberger, A. Knoll, S. Wildhirt, R. Lange, and R. Bauernschmitt. The significance of haptic feedback for telemanipulated heart surgery. In *Proceedings of The World Congress on Medical Physics and Biomedical Engineering [CD-ROM]*, Seoul, Korea, August 2006.

[9] E.U. Braun, C. Haßelbeck, H. Mayer, A. Knoll, S. Wildhirt, R. Lange, and R. Bauernschmitt. Assessment of surgical experience for telemanipulated heart surgery. In *Proceedings of The World Congress on Medical Physics and Biomedical Engineering [CD-ROM]*, Seoul, Korea, August 2006.

[10] E.U. Braun, H. Mayer, I. Nagy, A. Knoll, S.M. Wildhirt, R. Lange, and R. Bauernschmitt. An instrumentation system with force feedback, automatic recognition and skills for cardiac telemanipulation. In *Proceedings of the 33st Annual Meeting of Computers in Cardiology*, pages 553–555, Valencia, Spain, September 2006.

[11] C. Haßelbeck, E.U. Braun, H. Mayer, A. Knoll, S. Wildhirt, R. Lange, and R. Bauernschmitt. Evaluation of force-feedback in robotic heart surgery. In *Proceedings of the 2nd Russian-Bavarian Conference on Bio-Medical Engineering [CD-ROM]*, Moscow, Russia, June 2006.

[12] F. Freyberger, M. Popp, B. Faerber, H. Mayer, and E.U. Schirmbeck. Experimentelle Evaluation haptischer Rückmeldung eines robotergestützten Systems für minimal-invasive Herzchirurgie. *ZMMS Spektrum*, 19(1):15–20, 2005.

[13] M. Krane, E.U. Braun, H. Mayer, A. Knoll, R. Bauernschmitt, and R. Lange. Mitral valve reconstruction with artificial chordae: How to secure the desired length? In *Proceedings of the 34st Annual Meeting of Computers in Cardiology*, pages 745–748, Durham, North Carolina, September 2007.

[14] H. Mayer, F. Gomez, D. Wierstra, I. Nagy, A. Knoll, and J. Schmidhuber. A system for robotic heart surgery that learns to tie knots using recurrent neural networks. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 543–548, Beijing, China, October 2006.

[15] H. Mayer, I. Nagy, and A. Knoll. Skill transfer and learning by demonstration in a realistic scenario of laparoscopic surgery. In *Proceedings of the IEEE International Conference on Humanoids [CD-ROM]*, Munich, Germany, October 2003.

[16] H. Mayer, I. Nagy, and A. Knoll. Inverse kinematics of a manipulator for minimally invasive surgery. Technical report, Technische Universität München, Germany, February 2004.

[17] H. Mayer, I. Nagy, and A. Knoll. Kinematics and modeling of a system for robotic surgery. In J. Lenarcic and C. Galletti, editors, *On Advances in Robot Kinematics (Conf. Proceedings)*, pages 181–190, Sestri Levante, Italy, June 2004. Kluwer Academic Publishers.

[18] H. Mayer, I. Nagy, A. Knoll, E.U. Braun, and R. Bauernschmitt. Human-computer interfaces of a system for robotic heart surgery. In *Proceedings of the Second IASTED International Conference on Human-Computer Interaction*, pages 31–36, Chamonix, France, March 2007.

[19] 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*, 16(5):459–470, 2007.

[20] 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. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1800–1807, Rome, Italy, April 2007.

[21] H. Mayer, I. Nagy, 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*, pages 3637–3642, Sendai, Japan, September 2004.

[22] H. Mayer, I. Nagy, A. Knoll, E.U. Schirmbeck, and R. Bauernschmitt. Integration of force feedback in an open robot platform for robotic surgery. In *Proceedings of the 10th Mediterranean Conference on Medical and Biological Engineering [CD-ROM]*, Ischia, Italy, July 2004.

[23] H. Mayer, I. Nagy, A. Knoll, E.U. Schirmbeck, and R. Bauernschmitt. Robotic system to evaluate force feedback in minimally invasive computer aided surgery. In *Proceedings of the ASME International Design Engineering Technical Conferences [CD-ROM]*, Salt Lake City, Utah, September 2004.

[24] H. Mayer, I. Nagy, A. Knoll, E.U. Schirmbeck, and R. Bauernschmitt. Upgrading instruments for robotic surgery. In *Proceedings of the Australasian Conference on Robotics and Automation [CD-ROM]*, Canberra, Australia, December 2004.

[25] H. Mayer, I. Nagy, A. Knoll, E.U. Schirmbeck, and R. Bauernschmitt. An experimental system for robotic heart surgery. In *Proceedings of the IEEE 18th International Symposium on Computer-Based Medical Systems*, pages 55–60, Dublin, Republic of Ireland, June 2005.

[26] H. Mayer, I. Nagy, A. Knoll, E.U. Schirmbeck, and R. Bauernschmitt. A robotic system providing force feedback and automation for minimally invasive heart surgery. *International Journal of Computer Assisted Radiology and Surgery*, 1(1):265–267, 2006.

[27] I. Nagy, H. Mayer, and A. Knoll. Application of skill transfer in robotic surgery. Technical report, Technische Universität München, Germany, December 2003.

[28] I. Nagy, H. Mayer, and A. Knoll. Endo[PA]R - an overview of an experimental system for robotic surgery. Technical report, Technische Universität München, Germany, December 2003.

[29] I. Nagy, H. Mayer, and A. Knoll. Application of force feedback in robot assisted minimally invasive surgery. In *Proceedings of the 4th International Conference Eurohaptics*, pages 240–245, Munich, Germany, June 2004.

[30] I. Nagy, H. Mayer, A. Knoll, E.U. Schirmbeck, and R. Bauernschmitt. Endo[PA]R: An open evaluation system for minimally invasive robotic surgery. In *Proceedings of the IEEE Conference "Mechatronics & Robotics"*, pages 1464–1467, Aachen, Germany, September 2004.

[31] I. Nagy, H. Mayer, A. Knoll, E.U. Schirmbeck, and R. Bauernschmitt. Real-time matching of angiographies with in situ heart image sequences. In *Proceedings of the IEEE 17th symposium on Computer-Based Medical Systems*, pages 516–522, Bethesda, Maryland, June 2004.

[32] E.U. Schirmbeck, C. Haßelbeck, F. Freyberger, H. Mayer, A. Knoll, S. Wildhirt, R. Lange, and R. Bauernschmitt. Tactile feedback without direct touch: An achievement for robotically working heart surgeons? *Biomedical Engineering*, 50(1):23–24, 2005.

[33] E.U. Schirmbeck, C. Haßelbeck, H. Mayer, A. Knoll, F. Freyberger, S. Wildhirt, and R. Lange. The capability of haptic feedback as additional sensory quality for robotic heart surgery. In *Proceedings of the 4th Annual Meeting of the German Society for Computer and Robot Assisted Surgery, CURAC [CD-ROM]*, Berlin, Germany, September 2005.

[34] E.U. Schirmbeck, C. Haßelbeck, H. Mayer, I. Nagy, A. Knoll, F. Freyberger, M. Popp, R. Lange, and R. Bauernschmitt. Evaluation of haptics in robotic heart surgery. *International Journal of Computer Assisted Radiology and Surgery*, 1(1):730–734, 2005.

[35] E.U. Schirmbeck, I. Nagy, H. Mayer, A. Knoll, R. Lange, and R. Bauernschmitt. Automatic coronary artery detection on in situ heart images. In *Proceedings of the 31st Annual Meeting of Computers in Cardiology*, pages 785–788, Chicago, Illinois, September 2004.

[36] E.U. Schirmbeck, I. Nagy, H. Mayer, A. Knoll, R. Lange, and R. Bauernschmitt. Evaluation of force feedback in minimally invasive robotic surgery. *Biomedical Engineering*, 49(2):108–109, 2004.

Citations of Own Work by Others

[37] F. Focacci, M. Piccigallo, O. Tonet, G. Megali, A. Pietrabissa, and P. Dario. Lightweight hand-held robot for laparoscopic surgery. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 599–604, Rome, Italy, April 2007. `Citation of [25].`

[38] K. Houston, A. Sieber, C. Eder, O. Tonet, A. Menciassi, and P. Dario. Novel haptic tool and input device for real time bilateral biomanipulation addressing endoscopic surgery. In *Proceedings of the International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 198–201, Lyon, France, August 2007. `Citation of [25].`

[39] 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*, pages 204–209, Pisa, Italy, February 2006. `Citation of [21].`

[40] A. Kapoor, M. Li, and R. Taylor. Spatial motion constraints for robot assisted suturing using virtual fixtures. In J. Duncan and G. Gerig, editors, *Proceedings of the 8th International Conference on Medical Image Computing and Computer Assisted Intervention*, volume 3750 of *Lecture Notes in Computer Science*, pages 89–96, Berlin / Heidelberg, October 2005. Springer-Verlag. `Citation of [30].`

[41] P. Knappe, S. Pieck, and J. Wahrburg. Components and architecture of a navigated robot system for surgical applications. *Automatisierungstechnik*, 53(12):615–626, 2005. `Citation of [21].`

[42] R. Locke and R. Patel. Optimal remote center-of-motion location for robotics-assisted minimally-invasive surgery. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1900–1905, Rome, Italy, April 2007. `Citation of [17].`

[43] G. Megali, S. Sinigaglia, O. Tonet, F. Cavallo, and P. Dario. Understanding expertise in surgical gesture by means of Hidden Markov Models. In *Proceedings of the IEEE International Conference on Biomedical Robotics and Biomechatronics*, pages 625–630, Pisa, Italy, February 2006. `Citation of [15].`

[44] T. Ortmaier, B. Deml, B. Kübler, G. Passig, D. Reintsema, and U. Seibold. Robot assisted force feedback surgery. *Advances in Telerobotics*, 31:361–379, 2007. `Citation of [23] and [36].`

[45] P. Puangmali, K. Althoefer, L. Seneviratne, D. Murphy, and P. Dasgupta. State-of-the-art in force and tactile sensing for minimally invasive surgery. *IEEE Sensors*, 8(4):371–381, 2008. `Citation of [21] and [14].`

[46] S. Speidel, M. Delles, C. Gutt, and R. Dillmann. Tracking of instruments in minimally invasive surgery for surgical skill analysis. In G. Yang, T. Jiang, D. Shen, L. Gu, and J. Yang, editors, *Proceedings of the International Workshop on Medical Imaging Augmented Reality*, volume 4091 of *Lecture Notes in Computer Science*, pages 148–155, Berlin / Heidelberg, August 2006. Springer-Verlag. `Citation of [15].`

[47] O. Tonet, G. Megali, and P. Dario. Influence of visual position information in a tissue stiffness discrimination task with haptic feedback. In *The 2nd International Workshop on Virtual Reality Interaction and Physical Simulation*, pages 9–14, Pisa, Italy, November 2005. `Citation of [25].`

[48] L. Verner and A. Okamura. Sensor / actuator asymmetries in telemanipulators: Implications of partial force feedback. In *Proceedings of the IEEE Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pages 309–314, Alexandria, Virginia, March 2006. `Citation of [21].`

[49] C. Wagner and R. Howe. Force feedback benefit depends on experience in multiple degree of freedom robotic surgery task. *IEEE Transactions on Robotics and Automation*, 23(6):1235–1240, 2007. `Citation of [22].`

References

[50] 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*, pages 295–302, Atlanta, USA, March 1997.

[51] L. Adhami and È. Coste-Manière. Positioning tele-operated surgical robots for collision-free optimal operation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2962–2967, Washington, D.C., May 2002.

[52] 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*, 26(23):6288–6297, 2006.

[53] B. Amirikian and A. Georgopoulos. Modular organization of directionally tuned cells in the motor cortex: Is there a short-range order? *Proceedings of the National Academy of Sciences of the United States of America*, 100(21):12474–12479, 2003.

[54] R. Amit and M. Mataric. Parametric primitives for motor representation and control. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 863–868, Washington, D.C., May 2002.

[55] W. Ang, P. Khosla, and C. Riviere. Design of all-accelerometer inertial measurement unit for tremor sensing in hand-held microsurgical Instrument. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1781–1786, Taipei, Taiwan, September 2003.

[56] J. Arata, H. Takahashi, P. Pitakwatchara, S. Warisawa, K. Tanoue, K. Konishi, S. Ieiri, S. Shimizu, N. Nakashima, K. Okamura, Y. Fujino, Y. Ueda, P. Chotiwan, M. Mitsuishi, and M. Hashizume. A remote surgery experiment between Japan and Thailand over internet using a low latency CODEC system. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 953–959, Rome, Italy, April 2007.

[57] K. Arun, T. Huang, and S. Blostein. Least squares fitting of two 3D point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(5):698–700, 1987.

[58] C. Atkeson and S. Schaal. Robot learning from demonstration. In *Proceedings of the 14th International Conference on Machine Learning*, pages 12–20, Nashville, Tennessee, Juli 1997.

[59] R. Baumann and R. Clavel. Haptic interface for virtual reality based minimally invasive surgery simulation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 381–386, Leuven, Belgium, May 1998.

[60] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[61] D. Bentivegna and C. Atkeson. Learning from observation using primitives. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1988–1993, Seoul, Korea, May 2001.

[62] R. Berguer, W. Smith, and Y. Chung. Performing laparoscopic surgery is significantly more stressful for the surgeon than open surgery. *Surgical Endoscopy*, 15(10):1204–1207, 2001.

[63] B. Bethea, A. Okamura, M. Kitagawa, T. Fitton, S. Cattaneo, V. Gott, W. Baumgartner, and D. Yuh. Application of haptic feedback to robotic surgery. *Journal of Laparoendoscopic and Advanced Surgical Techniques*, 14(3):191–195, 2004.

[64] J. Bigge, S. Best, and K. Heller. *Teaching Individuals with Physical, Health, or Multiple Disabilities (4th edition).* Prentice Hall, Boston, USA, 2000.

[65] A. Blackwell, A. Jansen, and K. Marriott. Restricted focus viewer: A tool for tracking visual attention. In M. Anderson, P. Cheng, and V. Haarslev, editors, *Theory and Application of Diagrams*, volume 1889 of *Lecture Notes in Artificial Intelligence*, pages 162–177. Springer-Verlag, 2000.

[66] C. Breazeal. *Learning by Scaffolding.* PhD thesis, Massachusetts Institute of Technology, Cambridge, Maine, 1998.

[67] R. Brooks. Elephants don't play chess. *Robotics and Autonomous Systems*, 6:3–15, 1990.

[68] C. Cao, C. MacKenzie, and S. Payandeh. Task and motion analyses in endoscopic surgery. In *Proceedings ASME Dynamic Systems and Control Division*, pages 583–590, Atlanta, USA, 1996.

[69] A. Castellani, D. Botturi, M. Bicego, and P. Fiorini. Hybrid HMM/SVM model for the analysis and segmentation of teleoperation tasks. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2918–2923, New Orleans, LA, April 2004.

[70] M. Çavaşoğlu, W. Williams, F. Tendick, and S. Sastry. Robotics for telesurgery: Second generation Berkeley/UCSF laparoscopic telesurgical workstation and looking towards the future applications. *Industrial Robot, Special Issues on Medical Robotics*, 30(1):22–29, 2003.

[71] J. Chen and B. McCarragher. Robot programming by demonstration - selecting optimal event paths. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 518–523, Leuven, Belgium, May 1998.

[72] J. Chen and A. Zelinsky. Programming by demonstration: Removing suboptimal actions in a partially known configuration space. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 4096–4103, Seoul, Korea, May 2001.

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

[74] S. Cotin and H. Delingette. Real-time surgery simulation with haptic feedback using finite elements. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3739–3744, Leuven, Belgium, May 1998.

[75] J. Craig. *Introduction to Robotics (Mechanics and Control).* Addison Wesley, Boston, Massachusetts, USA, 1986.

[76] A. Darzi and S. Mackay. Skills assessment of surgeons. *Surgery*, 131(2):121–124, 2002.

[77] J. Decuyper and D. Keymeulen. A reactive robot navigation system based on a fluid dynamics metaphor. In H. Schwefel and R. Männer, editors, *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, volume 496 of *Lecture Notes in Computer Science*, pages 356–362, Berlin / Heidelberg, October 1990. Springer-Verlag.

[78] K. Dixon and P. Khosla. Trajectory representation using sequenced linear dynamical systems. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3925–3930, Barcelona, Spain, April 2005.

[79] D. Elizondo. The linear separability problem: Some testing methods. *IEEE Transactions on Neural Networks*, 17(2):330–344, 2006.

[80] C. Erikson. *Real-Time Collision Detection*. Morgan Kaufmann Publishers, San Francisco, California, USA, 2005.

[81] P. Fager. The use of haptics in medical applications. *International Journal of Medical Robotics and Computer Assisted Surgery*, 1(1):36–42, 2004.

[82] V. Falk, S. Jacobs, J. Gummert, and T. Walther. Robotic coronary artery bypass grafting: The Leipzig experience. *The Surgical Clinics of North America*, 83(6):1381–1386, 2003.

[83] V. Falk, S. Jacobs, J. Gummert, T. Walther, and F. Mohr. Computer-enhanced endoscopic coronary artery bypass grafting: The daVinci experience. *Seminars in Thoracic and Cardiovascular Surgery*, 15(2):104–111, 2003.

[84] V. Falk, D. Mintz, J. Grunenfelder, J. Fann, and T. Burdon. Influence of three-dimensional vision on surgical telemanipulator performance. *Surgical Endoscopy*, 15(11):1282–1288, 2001.

[85] M. Ferch, J. Zhang, and A. Knoll. Robot skill transfer based on b-spline fuzzy controllers for force-control tasks. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1170–1175, Detroit, Michigan, May 1999.

[86] F.Gosselin, C. Bidard, and J. Brisset. Design of a high fidelity haptic device for telesurgery. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 206–211, Barcelona, Spain, April 2005.

[87] P. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47:381–391, 1954.

[88] R. Franzino. The Laprotek surgical system and the next generation of robotics. *The Surgical Clinics of North America*, 83(6):1317–1320, 2003.

[89] H. Friedrich, J. Holle, and R. Dillmann. Interactive generation of flexible robot programs. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 538–543, Leuven, Belgium, May 1998.

[90] O. Fuentes and R. Nelson. Learning dextrous manipulation skills using the evolution strategy. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 501–506, Albuquerque, New Mexico, April 1997.

[91] A. Gallagher and R. Satava. Virtual reality as a metric for the assessment of laparoscopic psychomotor skills. *Surgical Endoscopy*, 16(2):1746–1752, 2002.

[92] G. De Gersem, J. Van der Sloten, F. Tendick, and H. Van Brussel. Enhanced kinaesthetic sensitivity in minimally invasive telesurgery. In *Proceedings of the 10th Mediterranean Conference on Medical and Biological Engineering [CD-ROM]*, Ischia, Italy, July 2004.

[93] Z. Ghahramani. An introduction to Hidden Markov Models and Bayesian networks. *Journal of Pattern Recognition and Artificial Intelligence*, 15(1):9–42, 2001.

[94] Z. Ghahramani and D. Wolpert. Modular decomposition in visuomotor learning. *Nature*, 386:392–395, 1997.

[95] S. Giszter, F. Mussa-Ivaldi, and E. Bizzi. Convergent force fields organized in the frog's spinal cord. *Journal of Neuroscience*, 13:467–491, 1993.

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

[97] G. Guthart and J. Salisbury. The Intuitive^{TM} telesurgery system: Overview and application. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 618–621, San Francisco, USA, April 2000.

[98] K. Hagio, N. Sugano, M. Takashina, T. Nishii, H. Yoshikawa, and T. Ochi. Effectiveness of the ROBODOC system during total hip arthroplasty in preventing intraoperative pulmonary embolism. In T. Dohi and R. Kikinis, editors, *Proceedings of the 5th International Conference on Medical Image Computing and Computer Assisted Intervention*, volume 2488 of *Lecture Notes in Computer Science*, pages 339–346, Berlin / Heidelberg, September 2002. Springer-Verlag.

[99] B. Hannaford and P. Lee. Hidden Markov Model analysis of force/torque information in telemanipulation. *The International Journal of Robotics Research*, 10(5):528–539, 1991.

[100] Y. Hasegawa and T. Fukuda. Learning method for hierarchical behavior controller. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2799–2803, Detroit, Michigan, May 1999.

[101] B. Heinrich and T. Bugnyar. Testing problem solving in ravens: String-pulling to reach food. *Ethology*, 111:962–976, 2005.

[102] 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*, 5:145–196, 1975.

[103] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[104] G. Hovland, P. Sikka, and B. McCarragher. Skill acquisition from human demonstration using a Hidden Markov Model. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2706–2711, Minneapolis, Minnesota, April 1996.

[105] T. Hu, G. Tholey, J. Desai, and A. Castellanos. Evaluation of a laparoscopic grasper with force feedback. *Surgical Endoscopy*, 18(5):863–867, 2004.

[106] I. Hunter, T. Doukoglou, S. Lafontaine, P. Charette, L. Jones, M. Sagar, G. Mallinson, and P. Hunter. A teleoperated microsurgical robot and associated virtual environment for eye surgery. *Presence*, 2:265–280, 1993.

[107] C. Hwang and K. Sasaki. Control program of two-fingered dexterous manipulation with primitive motions. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2902–2907, Las Vegas, Nevada, October 2003.

[108] 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*, 3(3):171–196, 2005.

[109] K. Ikuta, S. Daifu, T. Hasegawa, and H. Higashikawa. Hyper-finger for remote minimally invasive surgery in deep area. In T. Dohi and R. Kikinis, editors, *Proceedings of the 5th International Conference on Medical Image Computing and Computer Assisted Intervention*, volume 2488 of *Lecture Notes in Computer Science*, pages 173–181, Berlin / Heidelberg, September 2002. Springer-Verlag.

[110] T. Inamura, Y. Nakamura, H. Ezaki, and I. Toshima. Imitation and primitive symbol acquisition of humanoids by the integrated mimesis loop. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 4208–4213, Seoul, Korea, May 2001.

[111] Intuitive Surgical Inc. `http://www.intuitivesurgical.com`.

[112] J. Izawa, T. Kondo, and K. Ito. Biological robot arm motion through reinforcement learning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3398–3403, Washington, D.C., May 2002.

[113] A. Johansson and A. Sandstroem. Sensitivity of the human visual system to amplitude modulated light. Technical Report 4, National Institute for Working Life, Umea, Sweden, 2003.

[114] S. Jowsey. *Can I Play Too?: Physical Education for Physically Disabled Children in Mainstream Schools*. David Fulton Publishers, Oxford, UK, 1992.

[115] J.Takamatsu, T. Morita, K. Ogawara, H. Kimura, and K. Ikeuchi. Representation for knot-tying tasks. *IEEE Transaction on Robotics*, 22(1):65–78, 2006.

[116] M. Kaiser and R. Dillmann. Building elementary skills from human demonstration. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2700–2705, Minneapolis, USA, April 1996.

[117] M. Kaiser and R. Dillmann. Hierarchical refinement of skills and skill application for autonomous robots. *Robotics and Autonomous Systems*, 19(3):259–271, 1997.

[118] H. Kang. *Robotic Assisted Suturing in Minimally Invasive Surgery*. PhD thesis, Rensselaer Polytechnic Institute, Troy, New York, 2002.

[119] O. Kathib. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1):90–98, 1986.

[120] C. Kennedy, T. Hu, and J. Desai. Combining haptic and visual servoing for cardiothoracic surgery. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2106–2111, Washington, D.C., May 2002.

[121] J. Kim and P. Khosla. Real-time obstacle avoidance using harmonic potential functions. *IEEE Transactions on Robotics and Automation*, 8(3):338–349, 1992.

[122] H. Kimura and G. Kajiura. Motion recognition based cooperation between human operating robot and autonomous assistant robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 297–303, Albuquerque, New Mexico, April 1997.

[123] M. Kitagawa, A. Okamura, B. Bethea, V. Gott, and W. Baumgartner. Analysis of suture manipulation forces for teleoperation with force feedback. In T. Dohi and R. Kikinis, editors, *Proceedings of the 5th International Conference on Medical Image Computing and Computer Assisted Intervention*, volume 2488 of *Lecture Notes in Computer Science*, pages 155–162, Berlin / Heidelberg, September 2002. Springer-Verlag.

[124] K. Kleinmann, D. Bettenhausen, and M. Seitz. A modular approach for solving the peg-in-hole problem with a multifingered gripper. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 758–763, Nagoya, Japan, May 1995.

[125] D. Kochanek and R. Bartels. Interpolating splines with local tension, continuity, and bias control. *ACM SIGGRAPH*, 18(3):33–41, 1984.

[126] Teuvo Kohonen. *Self-Organizing Maps*. Springer-Verlag, Berlin, 1995.

[127] T. Kolb, W. Ilg, and J. Wille. Using time-discrete recurrent neural networks in nonlinear control. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pages 1367–1371, Anchorage, Alaska, May 1998.

[128] J. Konczak. On the notion of motor primitives in humans and robots. In *Proceedings of the Fifth International Workshop on Epigenetic Robotics*, pages 47–53, Nara, Japan, Juli 2005.

[129] R. Konietschke, T. Ortmaier, U. Hagn, and G. Hirzinger. Kinematic design optimization of an actuated carrier for the DLR multi-arm surgical system. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4381–4387, Beijing, China, October 2006.

[130] G. Kruijff, H. Zender, P. Jensfelt, and H. Christensen. Situated dialogue and understanding spatial organization: Knowing what is where and what you can do there. In *Proceedings of the IEEE International Symposium on Robot and Human Interactive Communication*, pages 328–333, Hatfield, United Kingdom, September 2006.

[131] U. Kühnapfel, H. Çakmak, and H. Maaß. Endoscopic surgery training using virtual reality and deformable tissue simulation. *Computers and Graphics*, 24:671–682, 2000.

[132] KUKA Roboter GmbH. `http://www.kuka.com/germany/de/products/industrial_robots`.

[133] J. Kuniholm and G. Buckner. Automated knot-tying for fixation in minimally invasive, robot-assisted cardiac surgery. *Journal of Biomechanical Engineering*, 127:1001–1008, 2005.

[134] D. Kwon, K. Woo, S. Song, W. Kim, and H. Cho. Microsurgical telerobot system. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 945–950, Victoria, Canada, October 1998.

[135] F. Lai and R. Howe. Evaluating control modes for constrained robotic surgery. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 603–609, San Francisco, USA, April 2000.

[136] B. Lim, S. Ra, and F. Park. Movement primitives, principal component analysis, and the efficient generation of natural motions. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 4630–4635, Barcelona, Spain, April 2005.

[137] H. Lin, I. Shafran, T. Murphy, A. Okamura, D. Yuh, and G. Hager. Automatic detection and segmentation of robot-assisted surgical motions. In J. Duncan and G. Gerig, editors, *Proceedings of the 8th International Conference on Medical Image Computing and Computer Assisted Intervention*, volume 3749 of *Lecture Notes in Computer Science*, pages 802–810, Berlin / Heidelberg, October 2005. Springer-Verlag.

[138] M. Lopes and J. Santos-Victor. Visual learning by imitation with motor representations. *IEEE Transactions on Systems, Man and Cybernetics*, 35(3):438–449, June 2005.

[139] M. MacFarlane, J. Rosen, B. Hannaford, C. Pellegrini, and M. Sinanan. Force feedback grasper helps restore the sense of touch in minimally invasive surgery. *Journal of Gastrointestinal Surgery*, 3(3):278–285, 1999.

[140] G. Mair. Telepresence - the technology and its economic and social implications. In *Proceedings of the International Symposium on Technology and Society*, pages 118–124, Glasgow, UK, June 1997.

[141] R. Malak and P. Khosla. A framework for the adaptive transfer of robot skill knowledge using reinforcement learning agents. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1994–2001, Seoul, Korea, May 2001.

[142] R. Mann, A. Jepson, and T. El-Maraghi. Trajectory segmentation using dynamic programming. In *Proceedings of the 16th International Conference on Pattern Recognition*, pages 331–334, Quebec, Canada, August 2002.

[143] J. Marescaux, J. Leroy, M. Gagner, F. Rubino, D. Mutter, M. Vix, S. Butner, and M. Smith. Transatlantic robot-assisted telesurgery. *Nature*, 413(27):379–380, 2001.

[144] T. Massie and J. Salisbury. The PHANTOM haptic interface: A device for probing virtual objects. In *Proceedings of the ASME Winter Annual Meeting*, pages 295–300, Chicago, Illinois, November 1994.

[145] T. Matsuoka, T. Hasegawa, T. Kiriki, and K. Honda. Mechanical assembly based on motion primitives of multi-fingered hand. In *Proceedings of the IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, page 77, Tokyo, Japan, June 1997.

[146] B. McCarragher. Task primitives for the discrete event modeling and control of 6-DoF assembly tasks. *IEEE Transactions on Robotics and Automation*, 12(2):280–289, 1996.

[147] M. McDaris-Dass. Telesurgery: The medical wave of the future. Technical report, University of Southern California, 2003.

[148] P. Medricky, C. Brom, G. Zuend, J. Gruenenfelder, and N. Hitendu. *Helical Needle (Patent)*, 2006. International Application No. PCT/CH2005/000743, International Filing Date 12/12/2005, Publication No. WO/2006/063481.

[149] J. Mercer. Functions of positive and negative type and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society London*, 209:415–446, 1909.

[150] Mitsubishi Electric Inc. `http://www.mitsubishi-automation.de/roboter.html`.

[151] M. Mitsuishi, J. Arata, K. Tanaka, M. Miyamoto, T. Yoshidome, S. Iwata, S. Warisawa, and M. Hashizume. Development of a remote minimally-invasive surgical system with operational environment transmission capability. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2663–2670, Taipei, Taiwan, September 2003.

[152] M. Mitsuishi, Y. Iizuka, H. Watanabe, H. Hashizume, and K. Fujiwara. Remote operation of a micro-surgical system. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1013–1019, Leuven, Belgium, May 1998.

[153] M. Mitsuishi, S. Tomisaki, T. Yoshidome, H. Hashizume, and K. Fujiwara. Tele-micro-surgery system with intelligent user interface. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1607–1614, San Francisco, USA, April 2000.

[154] P. Morgan, T. Carter, S. Davis, A. Sepehri, J. Punt, P. Byrne, A. Moody, and P. Finlay. The application accuracy of the Pathfinder neurosurgical robot. In *Proceedings of the 17th International Congress on Computer Assisted Radiology and Surgery*, pages 561–5657, London, UK, June 2003.

[155] J. Morrow and P. Khosla. Manipulation task primitives for composing robot skills. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3354–3359, Albuquerque, New Mexico, April 1997.

[156] J. Morrow, B. Nelson, and P. Khosla. Manipulation task primitives for composing robot skills. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 234–240, Pittsburgh, Baltimore, August 1995.

[157] K. Müller, A. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik. Using support vector machines for time series prediction. In *Advances in Kernel Methods - Support Vector Learning*, pages 243–254. MIT Press, Cambridge, MA, 1999.

[158] S. Müller, F. Wallhoff, F. Hülsken, and Gerhard Rigoll. Facial expression recognition using pseudo 3D Hidden Markov Models. In *Proceedings of the International Conference on Pattern Recognition*, pages 32–35, 2002.

[159] V. Munoz-Martínez, C. Vara-Thorbeck, J. Gómez de Gabriel, J. Fernández-Lozano, E. Sanchez-Badajoz, A. García-Cerezo, R. Toscano, and A. Jimenez-Garrido. Medical robotic assistant for minimally invasive surgery. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2901–2906, San Francisco, USA, April 2000.

[160] T. Murphy. *Towards Objective Surgical Skill Evaluation with Hidden Markov Model-based Motion Recognition*. PhD thesis, The Johns Hopkins University, Baltimore, Maryland, 2004.

[161] D. Myers, M. Pritchard, and M. Brown. Automated programming of an industrial robot through teach-by showing. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 4078–4083, Seoul, Korea, May 2001.

[162] MySQL AB. `http://www.mysql.org`.

[163] I. Nagy. *3D Situs Reconstruction in Minimally Invasive Surgery*. PhD thesis, Technische Universität München, Munich, Germany, 2008.

[164] Y. Nakamura, K. Kishi, and H. Kawakami. Heartbeat synchronization for robotic cardiac surgery. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2014–2019, Seoul, Korea, May 2001.

[165] Y. Nakamura, T. Yamazaki, and N. Mizushima. Synthesis, learning and abstraction of skills through parameterized smooth map from sensors to behaviors. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2398–2405, Detroit, Michgan, May 1999.

[166] A. Nakazawa, S. Nakaoka, and K. Ikeuchi. Synthesize stylistic human motion from examples. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3899–3904, Taipei, Taiwan, September 2003.

[167] D. Necsulescu and R. Jassemi-Zargani. Extended Kalman filter based sensor fusion for operational space control of a robot arm. In *Proceedings of the IEEE Instrumentation and Measurement Technology Conference*, pages 915–918, 2001.

[168] W. Newman, C. Birkhimer, and R.Hebbar. Towards automatic transfer of human skills for robotic assembly. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2528–2533, Las Vegas, Nevada, October 2003.

[169] N. Nguyen, D. Phung, S. Venkatesh, and H. Bui. Learning and detecting activities from movement trajectories using the hierarchical Hidden Markov Model. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, pages 955–960, 2005.

[170] T. Ogasawara, H. Hirukawa, K. Kitagaki, H. Onda, A. Nakamura, and H. Tsukune. A telerobotics system for maintenance tasks integrating planning functions based on manipulation skills. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2870–2876, Leuven, Belgium, May 1998.

[171] K. Ogawara, J. Takamatsu, H. Kimura, and K. Ikeuchi. Generation of a task model by integrating multiple observations of human demonstrations. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1545–1550, Washington, D.C., May 2002.

[172] 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*, pages 3893–3898, Taipei, Taiwan, September 2003.

[173] 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*, pages 18–23, Barcelona, Spain, April 2005.

[174] A. Okamura, R. Webster, J. Nolin, K. Johnson, and H. Jafry. The haptic scissors: Cutting in virtual environments. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 828–833, Taipei, Taiwan, September 2003.

[175] T. Ortmaier, H. Weiss, C. Ott, G. Hirzinger, and U. Schreiber. A soft robotics approach for navigated pedicle screw placement - first experimental results. In *Proceedings of the 20th International Congress on Computer Assisted Radiology and Surgery*, pages 205–209, Osaka, Japan, June 2006.

[176] Y. Ou and Y. Xu. Learning human control strategy for dynamically stable robots: Support vector machine approach. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3455–3460, Taipei, Taiwan, September 2003.

[177] J. Owens and A. Hunter. Application of the self-organising map to trajectory classification. In *Proceedings of the Third IEEE International Workshop on Visual Surveillance*, pages 77–83, Dublin, Ireland, July 2000.

[178] J. Petermann, M. Schierl, P. Heeckt, and L. Gotzen. The Caspar-system in the reconstruction of the ACL - first follow-up results. In *Proceedings of the 5th International Symposium on Computer Assisted Orthopaedic Surgery*, pages 23–29, Davos, Switzerland, February 2000.

[179] P. Pitakwatchara, S. Warisawa, and M. Mitsuishi. Force feedback augmentation modes in the laparoscopic minimal invasive telesurgical system. In *Proceedings of the IEEE International Conference on Biomedical Robotics and Biomechatronics*, pages 1059–1066, Pisa, Italy, February 2006.

[180] H. Reichenspurner, R. Damiano, M. Mack, D. Boehm, H. Gulbins, C. Detter, B. Meiser, R. Ellgass, and B. Reichart. Use of the voice-controlled and computer-assisted surgical system ZEUS for endoscopic coronary artery bypass grafting. *Journal of Thoracic and Cardiovascular Surgery*, 118(1):11–16, 1999.

[181] K. Reidemeister. *Knotentheorie (Ergebnisse der Mathematik und ihrer Grenzgebiete)*. Springer Verlag, Berlin, Germany, 1932.

[182] L. Reng, T. Moeslund, and E. Granum. Finding motion primitives in human body gestures. In S. Gibet, N. Courty, and J. Kamp, editors, *6th International Gesture Workshop*, volume 3881 of *Lecture Notes in Artificial Intelligence*, pages 133–144. Springer-Verlag, 2006.

[183] S. Riesner. *Korrelations- und Prädiktionsverfahren zur Lageverfolgung in der perkutanen Radioonkologie*. PhD thesis, Technical University Munich, Garching, Germany, 2003.

[184] J. Rosen, M. Solazzo, B. Hannaford, and M. Sinanan. Task decomposition of laparoscopic surgery for objective evaluation of surgical residents' learning curve using Hidden Markov Model. *Computer Aided Surgery*, 7:49–61, 2000.

[185] S. Sabanovic, M. Michalowski, and R. Simmons. Robots in the wild: Observing human-robot social interaction outside the lab. In *Proceedings of the 9th International Workshop on Advanced Motion Control*, pages 576–581, Istanbul, Turkey, March 2006.

[186] T. Sanger. Optimal movement primitives. In *Proceedings on Advances in Neural Information Processing Systems 7*, pages 1023–1030, Denver, Colorado, November 1994.

[187] T. Sato, Y. Genda, H. Kubotera, T. Mori, and T. Harada. Robot imitation of human motion based on qualitative description from multiple measurement of human and environmental data. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2377–2384, Las Vegas, Nevada, October 2003.

[188] J. Saunders, C. Nehaniv, K. Dautenhahn, and A. Alissandrakis. Self-imitation and environmental scaffolding for robot teaching. *International Journal of Advanced Robotic Systems*, 4(1):109–124, 2007.

[189] S. Schaal, A. Ijspeert, and A. Billard. Computational approaches to motor learning by imitation. *Philosophical transactions of the Royal Society of London, series B*, 358(1431):537–547, 2003.

[190] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert. Learning movement primitives. In *Springer Tracts in Advanced Robotics: International Symposium on Robotics Research*, volume 1805. Springer-Verlag, October 2004.

[191] J. Schmidhuber, D. Wierstra, and F. Gomez. Evolino: Hybrid neuroevolution / optimal linear search for sequence learning. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 853–858, Edinburgh, August 2005.

[192] B. Schöllkopf, A. Smola, R. Williamson, and P. Bartlett. New support vector algorithms. *Neural Computation*, 12:1083–1121, 2000.

[193] S. Shew, D. Ostlie, and G. Holcomb. Robotic telescopic assistance in pediatric laparoscopic surgery. *Pediatric Endosurgery and Innovative Techniques*, 7(4):371–376, 2003.

[194] K. Shimoga. Perceptual feedback issues in dexterous telemanipulation: Part I. Finger force feedback. In *Proceedings of the IEEE International Symposium on Virtual Reality*, pages 263–270, Seattle, Washington, September 1993.

[195] K. Shoemake. Animating rotation with quaternion curves. In *Proceedings of the International Conference on Computer Graphics and Interactive Techniques*, pages 245–254, San Francisco, CA, July 1985.

[196] H. Siegelmann and E. Sontag. Turing computability with neural nets. *Applied Mathematics Letters*, 4(6):77–80, 1991.

[197] 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*, pages 38–47, Vienna, Austria, October 2005.

[198] M. Skubic and R. Volz. Learning force sensory patterns and skills from human demonstration. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 284–290, Albuquerque, New Mexico, April 1997.

[199] M. Skubic and R. Volz. Learning force-based assembly skills from human demonstration for execution in unstructured environments. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1281–1288, Leuven, Belgium, May 1998.

[200] L. Slutski, I. Gurevich, and Y. Edan. Parametric primitives for motor representation and control. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1016–1021, Albuquerque, New Mexico, April 1997.

[201] L. Suchman. *Plans and Situated Actions: The Problem of Human-Machine Communication*. Cambridge University Press, New York, USA, 1987.

[202] Systems in Motion AS. `http://www.coin3d.org`.

[203] M. Tavakoli, R. Patel, and M. Moallem. Haptic feedback and sensory substitution during telemanipulated suturing. In *Proceedings of the IEEE World Haptics Conference*, pages 543–544, Pisa, Italy, March 2005.

[204] F. Tendick, S. Bhoyrul, and L. Way. Comparison of laparoscopic imaging systems and conditions using a knot-tying task. *Computer Aided Surgery*, 2(1):24–33, 1997.

[205] J. Thompson, M. Ottensmeier, and T. Sheridan. Human factors in telesurgery: Effects of time delay and asynchrony in video and control feedback with local manipulative assistance. *Telemedicine Journal*, 5(2):129–137, 1999.

[206] H. Tominaga, J.Takamatsu, K. Ogawara, H. Kimura, and K. Ikeuchi. Symbolic representation of trajectories for skill generation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 4076–4081, San Francisco, California, April 2000.

[207] N. Turro, O. Kathib, and È. Coste-Manière. Haptically augmented teleoperation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 386–392, Seoul, Korea, May 2001.

[208] Shinji Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(4):373–380, 1991.

[209] U. Voges, E. Holler, B. Neisius, M. Schurr, and T. Vollmer. The advanced robotics and telemanipulator system for minimally invasive surgery. In *Proceedings of the IARP 2nd Workshop on Medical Robotics*, pages 137–148, Karlsruhe, Germany, 1997.

[210] R. Voyles, J. Morrow, and P. Khosla. Towards gesture-based programming: Shape from motion primordial learning of sensorimotor primitives. *Robotics and Autonomous Systems*, 22(4):361–375, 1997.

[211] L. Vygotsky. *Mind in Society: Development of Higher Psychological Processes (14th edition)*. Harvard University Press, Cambridge, MA, 1978.

[212] C. Wagner, D. Perrin, R. Howe, N. Vasilyev, and P. del Nido. Force feedback in a three-dimensional ultrasound-guided surgical task. In *Proceedings of the 14th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pages 43–48, Arlington, Virginia, March 2006.

[213] C. Wagner, N. Stylopoulos, and R. Howe. The role of force feedback in surgery: Analysis of blunt dissection. In *Proceedings of the 10th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pages 68–74, Orlando, Florida, March 2002.

[214] 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*, pages 2516–2521, April 2005.

[215] J. Walter. PSOM network: Learning with few examples. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2054–2059, Leuven, Belgium, May 1998.

[216] Q. Wang and J. De Schutter. Towards real-time robot programming by human demonstration for 6D force controlled actions. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2256–2261, Leuven, Belgium, May 1998.

[217] S. Waydo and R. Murray. Vehicle motion planning using stream functions. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2484–2491, Taipei, Taiwan, September 2003.

[218] J. Wiemeyer. Flimmerverschmelzungsfrequenz und zentralnervöse Aktivierung. *Neurologie und Rehabilitation*, 8(1):29–34, 2002.

[219] D. Wierstra, F. Gomez, and J. Schmidhuber. Modeling systems with internal state using Evolino. In *In Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, pages 1795–1802, Washington, D.C., June 2005.

[220] D. Wolpert and Z. Ghahramani. Computational principles of movement neuroscience. *Nature Neuroscience supplement*, 3:1212–1217, 2000.

[221] D. Wood, J. Bruner, and G. Ross. The role of tutoring in problem-solving. *Journal of Child Psychology and Psychiatry*, 17:89–100, 1976.

[222] L. Xu and Y. Zheng. Real-time motion planning for personal robots using primitive motions. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1414–1419, San Francisco, California, April 2000.

[223] T. Yoshikawa. *Foundations of Robotics: Analysis and Control*. MIT Press, Massachusetts, USA, 1990.

[224] P. Zegers and M. Sundareshan. Trajectory generation and modulation using dynamic neural networks. *IEEE Transactions on Neural Networks*, 14(3):520–533, 2003.

[225] J. Zhang and M. Ferch. Rapid on-line learning of compliant motion for two-arm coordination. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1853–1858, Leuven, Belgium, May 1998.

[226] J. Zhang and A. Knoll. Control architecture and experiment of a situated robot system for interactive assembly. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3906–3910, Washington, D.C., May 2002.

[227] J. Zhang, Y. von Collani, and A. Knoll. On-line learning of b-spline fuzzy controller to acquire sensor-based assembly skills. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1418–1423, Albuquerque, New Mexico, April 1997.

[228] Y. Zhang and K. Valavanis. A 3-d potential panel method for robot motion planning. *Robotica*, 15(4):421–434, 1997.

[229] G. Zhao, S. Tatsumi, and R. Sun. A heuristic Q-learning architecture for fully exploring a world and deriving an optimal policy by model-based planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2078–2083, Detroit, Michigan, May 1999.

[230] R. Zöllner, M. Pardowitz, S. Knoop, and R. Dillmann. Towards congnitive robots: Building hierarchical task representation of manipulations from human demonstrations. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1547–1552, Barcelona, Spain, April 2005.

[231] L. Zollo, E. Guglielmelli, G. Teti, C. Laschi, S. Eskiizmirliler, F. Carenczi, P. Bendaham, P. Gorce, M. Maier, Y. Burnod, and P. Dario. A bio-inspired neuro-controller for an anthropomorphic head-arm robotic system. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 12–17, Barcelona, Spain, April 2005.