Software Architecture of a System for Robotic Surgery

Hermann Mayer[1], Alois Knoll[1], Darius Burschka[1], Eva U. Braun[2], Rüdiger Lange[2] and Robert Bauernschmitt[2]

July 17, 2008

[1]{mayerh|burschka|knoll}@in.tum.de
Technische Universität München, Department of Robotics and Embedded Systems
[2]{brauneva|lange|bauernschmitt}@dhm.mhn.de
German Heart Center Munich, Department of Cardiovascular Surgery

Abstract

At the German Heart Center Munich we have installed and evaluated a novel system for robotic surgery. Its main features are the incorporation of haptics (by means of strain gauge sensors at the instruments) and partial automation of surgical tasks. However, in this paper we focus on the software engineering aspects of the system. We present a hierarchical approach, which is inspired by the modular architecture of the hardware. Each component of the system, and therefore each component of the control software can be easily interchanged by another instance (e.g. different types of robots may be employed to carry the surgical instruments). All operations are abstracted by an intuitive user interface, which provides a high level of transparency. In addition, we have included techniques known from character animation (socalled key-framing) in order to enable operation of the system by users with non-engineering backgrounds. The introduced concepts have proven effective during an extensive evaluation with 30 surgeons. Thereby, the system was used to conduct simplified operations in the field of heart surgery, including the replacement of a papillary tendon and the occlusion of an atrial septal defect.

Contents

1 Introduction

Endoscopic surgery is a challenging technique for thoracic interventions. Its application is especially expedient in the field of heart surgery, because sternotomy or large intercostal cuts can be avoided. Therefore, the collateral surgical trauma of the patients is minimized, which results in quicker recovery of patients. In addition, the time of hospitalization and the infection rate can be reduced. Therefore, patients massively profit from this endoscopic treatment option. On the other hand, surgeons have to cope with increasingly complex working conditions, but the design of intuitive user interfaces can help to overcome these barriers. Since endoscopic surgery is performed through a small port or "key-hole" in the patient's chest (cf. fig. 1), surgeons must learn to operate with unfamiliar and often awkward surgical instruments. All movements have to be performed using "P" as fulcrum and visual impressions of the field of operation can only be provided by means of an endoscopic camera. Hence, the techniques of endoscopic surgery have been

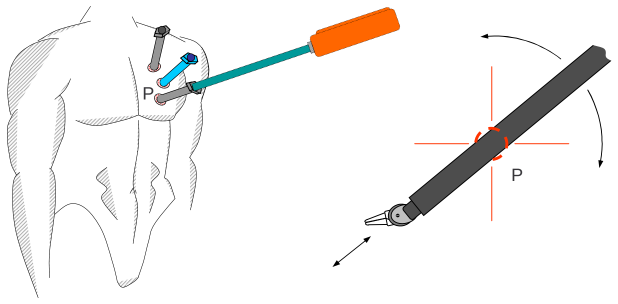


Figure 1: **Location of endoscopic ports for instruments and camera**

applied uncommonly, particularly in the field of heart surgery. An important step to push this technology was the introduction of telemanipulation, which was especially designed to overcome the fulcrum effect of endoscopic instruments. The surgeon no longer operates the instruments directly, but they are 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 for such systems are the *daVinci*^{TM} [2] and *ZEUS*^{TM} [8] systems (the latter has been discontinued). They are good examples of how the proper design of user interfaces can push forward new technologies like minimally invasive and endoscopic surgery. 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, tremor filtering and a stereo vision interface at the input console. Surgeons can now operate with a surgical mechatronic assistant in a comfortable, dextrous and intuitive manner [1]. Despite the obvious potential advantages of robot assisted surgery, most researchers and surgeons in this area agree that the lack of a haptic interface is a crucial drawback of currently available systems [7]. 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 Cartesian position control (the control loop is implicitly closed by visual servoing of the surgeon). In addition, it is not possible for users from other fields to program new trajectories for those devices. Therefore, our main research interests are the construction and evaluation of force sensory / force feedback and the development of an easy-to-use interface for trajectory planning. After a short introduction of our hardware and software we will focus on the presentation of a the planning interface and software architecture of the system.

2 Materials and Methods

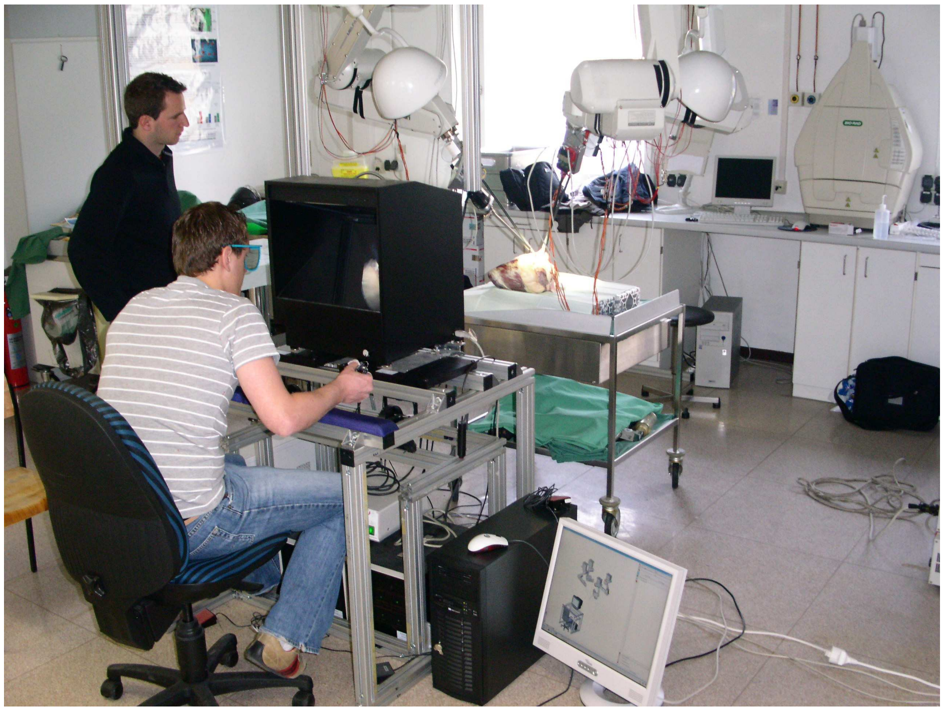


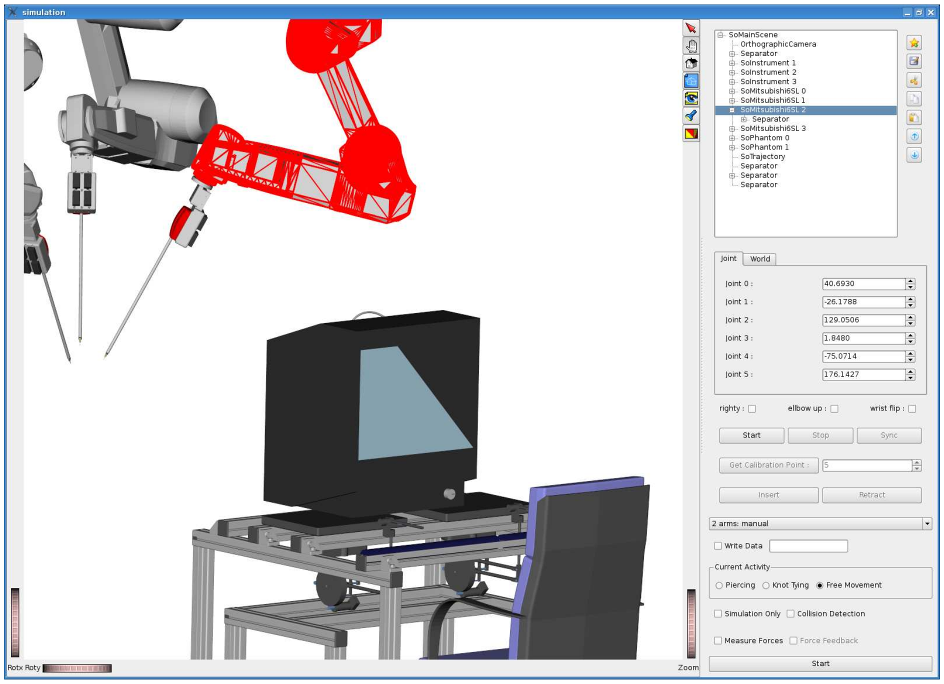
Figure 2: **System setup at the German Heart Center**

We have developed an experimental system for robotic surgery (cf. fig. 2). Four robotic manipulators are controlled by two PHANToM^{TM}devices from Sensable Inc. This device is available in different versions with different capabilities. We have chosen the version Premium 1.5, which provides a $20 \times 25 \times 40$ cm workspace that is large enough for surgical procedures. The user controls a stylus pen that is equipped with a switch that can be used to open and close the micro-grippers. The most interesting feature of the employed PHANToM^{TM}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 in a certain direction. This creates the impression of occurring forces, while the user is holding the pen at a certain posture. Our version of the PHANToM^{TM}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. Since the shaft of the surgical instrument is 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. For efficient telemanipulation, it is critical to have a 3D-interface providing a clear view of the operating area. In order to allow for such a feature, we equipped an additional robot with a 3D endoscopic camera. Like the surgical instruments, this camera can also be moved by means of trocar kinematics and can either be actively controlled by the operator or automatically tracked by the system. To enable stereo vision we have integrated an optical system with a semi transparent mirror that displays for each eye the corresponding camera view. More details about the system may be found at [6].

3 Planning Interface

Apart from the manual user interface (master console with PHANToMs), 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 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 on the upper right side of the GUI. The scene browser can be used as a basic CAD program. It is possible to insert new primitives (like cones, spheres, cubes etc.) or VRML objects, e.g. an endoscopic instrument. If these objects are selected, a context menu for the corresponding parameters is displayed. Therefore, in fig. 3, a corresponding context menu to adjust the different joints of the robot is displayed. Each context menu of an object contains functionality to translate or rotate the object. With the help of the scene browser it is also possible to aggregate objects to groups, which can be manipulated on their part. It is also possible to move objects in the hierarchy of the scene graph or to remove them completely. In addition, we have implemented copy and paste functions in order to reuse preassembled parts. The scene graph, or parts of it, can be stored to disk in order to get a permanent copy. This functions constitute an intuitive interface for users to manage different scenes and make certain modifications. All operations on the scene graph are implemented by means of the open source *Coin3D* interface from Systems in Motion AS. This is a high-level graphics language based on *OpenGL*.

The GUI provides different modes of interaction with the robots or surgical instruments. As mentioned



```
SoMainScene
  OrthographicCamera
  Separator
  SoInstrument 1
  SoInstrument 2
  SoInstrument 3
  SoMitsubishi6SL 0
  SoMitsubishi6SL 1
  SoMitsubishi6SL 2
    Separator
  SoMitsubishi6SL 3
  SoPhantom 0
  SoPhantom 1
  SoTrajectory
  Separator
  Separator
  Separator
```

Joint 0 : 40.6930
Joint 1 : -26.1788
Joint 2 : 129.0506
Joint 3 : 1.8480
Joint 4 : -75.0714
Joint 5 : 176.1427

righty : ☐ ellbow up : ☐ wrist flip : ☐

Get Calibration Point : 5

2 arms: manual

☐ Write Data

Current Activity: ○ Piercing ○ Knot Tying ● Free Movement

☐ Simulation Only ☐ Collision Detection

☐ Measure Forces ☐ Force Feedback

Figure 3: **Graphical User Interface**

above, the robots can be moved by means of sliders; one for each joint of the robot. In addition, the robots can also be moved in Cartesian space, i.e. linear translations in x,y and z direction and corresponding rotations about these axes. After the configuration of the robot has been determined, Cartesian movements will be mapped onto joint angles by a specific inverse kinematics. The same applies to the minimal invasive instruments, which require a special inverse kinematics if moved in Cartesian space (so-called port kinematics [5]). Port kinematics arranges for movements of the instrument about a small incision in the patient's

body and is indispensable for robotic applications in endoscopic surgery. Since each instrument is linked to a dedicated robot, any movement that changes the position of the instrument's base will consequently induce corresponding movements of the robot where the base is attached to. So far, we can use this interface to move the robots or instruments from one posture to another. This can either be executed in real-time or offline. In realtime mode, the robot directly follows the movements that 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, as they may occur in robotic knot-tying, this kind of interface for point-to-point movements will not be sufficient. Therefore, we have developed a planning interface based on keyframing.
Speaking of keyframing regarding trajectory planning, the robots are moved to certain consecutive positions, which are interpreted as keyframes. Afterwards, we apply a certain policy (e.g. linear or spline interpolation) to generate all other points of the trajectory lying in between those keyframes (see fig. 4). There are

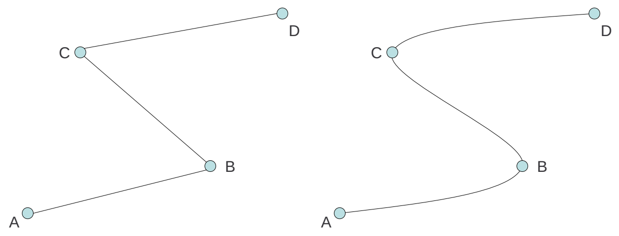


Figure 4: **Linear and spline interpolation between keyframes**

two different modes of moving on a trajectory with keyframes: one is to stop at each keyframe, the other, more complex possibility is to perform an continuous movement through all keyframes between start and end. Being the most difficult possibility, 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 stopping. Otherwise it is not possible to achieve jerkfree motions. Another prerequisite for our application is, that keyframes occur at certain points in time which have to be met exactly. I.e. if the robot starts in point A (cf. fig. 4) 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 $v_b(t)$, respectively:

$$v_a(t) = \frac{v_0}{1+e^{n(1-\frac{2t}{t_a})}} \qquad v_b(t) = \frac{v_0}{1+e^{-n(1-\frac{2t}{t_b})}} \tag{1}$$

Those are sigmoid functions shifted along the positive t-axis. The factor n changes the acclivity of the curve, which reaches its maximum at $t = \frac{t_{max}}{2}$. The time needed for acceleration and deceleration is denoted as t_a and t_b, respectively. Another nice feature of these functions is, that the area underneath the curve (i.e. the traveled path) amounts to $\frac{1}{2}t_a v_0$. Therefore, we can easily determine the residual speed v_0, given a certain path length and frame time (the same holds for deceleration). Determining the path length is easy for linear keyframe interpolations, but analytically not feasible for splines (in that case Hermite splines). This issue can be solved by using the adapted formulas of [3]. Therefore, it is guaranteed that the next keyframe will be reached at the right time and with the right velocity. The only thing left to do for the user is to set the keyframes on a timeline within the GUI. All trajectories can be displayed in the simulation environment and a preview of the corresponding movements is possible. If collisions occur, it is not possible to execute the trajectory before it is safely replanned.

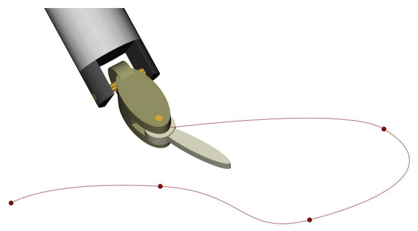


Figure 5: **Display of the keyframes and the calculated trajectory**

4 Software Architecture

After describing the planning interface, we also want to present 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. 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.
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 3*ms*. Therefore, new joint angles arrive at the robot at least once in a control cycle of 6.8*ms*. If more than one set of joints arrives during a cycle, additional values are simply omitted by the controller and will 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 induce 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.

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

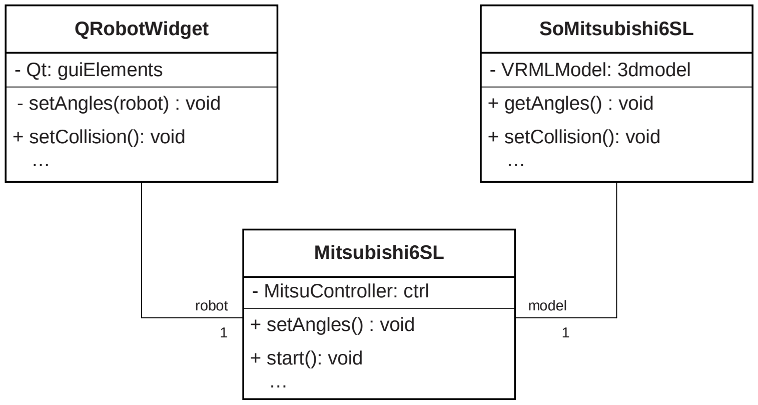


Figure 6: Model-View-Controller (MVC) architecture of the robot interface

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.
This abstract robot class can be implemented by various types of robots. While the robot-specific parts (e.g. low-level communication with the controller) are hidden within these specific classes, they have to implement 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. 7). A similar concept is used for the classes implementing the view of a robot. They are all derived from a *Coin3D* base class, `SoRobot`.

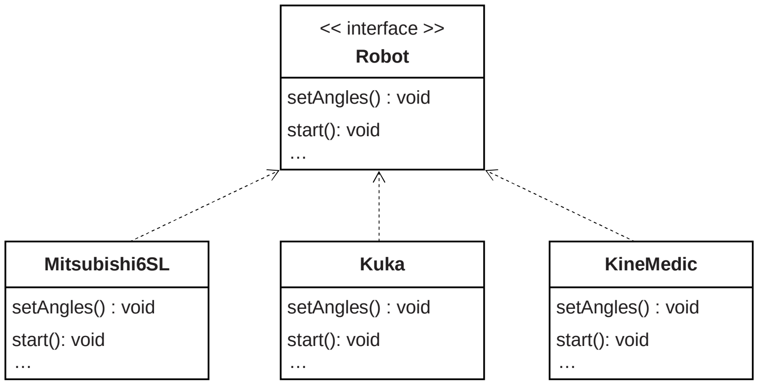


Figure 7: Different models of robots are implementations of the same abstract class

5 Conclusion

We have presented the software platform of a robotic system for minimally invasive surgery. In order to provide an intuitive user interface we have adopted a keyframing approach like it is known from computer animation. All relevant motion parameters, which are usually difficult to determine, will be calculated automatically. Therefore, the operator can concentrate on task-specific work, e.g. planning the trajectory for endoscopic knot-tying. In addition, we have based our approach on a flexible software architecture, which enables an easy adaption of the system to new hardware technology. We hope this work will simplify the handling of complex technical systems and hopefully will be adopted by programmers from other fields, e.g. industrial robotics. At least in the field of robotic surgery this technique has advanced the acceptance of robots in the operating room by surgeons. In the near future we will provide the possibility to integrate preoperative, patient centered image data like CT or MR scans. In addition, we are planning to augment the user interface by haptic concepts like virtual fixtures [4].

References

[1] 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. 1

[2] 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. 1

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

[4] P. Marayong, M. Li, A. Okamura, and G. Hager. Spatial motion constraints: Theory and demonstrations for robot guidance using virtual fixtures. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1270–1275, Taipei, Taiwan, September 2003. 5

[5] 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. 3

[6] H. Mayer, I. Nagy, A. Knoll, E.U. Braun, R. Lange, and R. Bauernschmitt. Adaptive control for human-robot skilltransfer: Trajectory planning based on fluid dynamics. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1800–1807, Rome, Italy, April 2007. 2

[7] 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. 1

[8] 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 Cardiovasc. Surgery*, 118(1):11–16, 1999. 1