

UNIVERSITY OF PATRAS - SCHOOL OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING



ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

DIVISION: SYSTEMS AND AUTOMATIC CONTROL

THESIS

of the student of the Department of Electrical and Computer Engineering of the School of
Engineering of the University of Patras

KARADIMOS ALEXIOS OF LOUKAS

STUDENT NUMBER: 1046820

Subject

**Robotic surgical tool manipulator - Recognition,
control and manipulation of laparoscopic tools**

Supervisor

Associate Professor Dr. Evangelos Dermatas

Co-Supervisor

Professor Dr. Anthony Tzes

Thesis Number:

Patras, 2020

ΠΙΣΤΟΠΟΙΗΣΗ

Πιστοποιείται ότι η διπλωματική εργασία με θέμα

Robotic surgical tool manipulator - Recognition, control and manipulation of laparoscopic tools

του φοιτητή του Τμήματος Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών

Karadimos Alexios of Loukas

(A.M.: 1046820)

παρουσιάστηκε δημόσια και εξετάστηκε στο τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών στις

— / — / —

Ο Επιβλέπων

Ο Διευθυντής του Τομέα

Evangelos Dermatas
Associate Professor Dr.

Kazakos Demosthenes
Assistant Professor Dr.

CERTIFICATION

It is certified that the Thesis with Subject

Robotic surgical tool manipulator - Recognition, control and manipulation of laparoscopic tools

of the student of the Department of Electrical and Computer Engineering

Karadimos Alexios of Loukas

(R.N.: 1046820)

Was presented publicly and defended at the Department of Electrical and Computer
Engineering at

— / — / —

The supervisor

The Director of the Division

Evangelos Dermatas
Associate Professor Dr.

Kazakos Demosthenes
Assistant Professor Dr.

Thesis details

Subject: **Robotic surgical tool manipulator - Recognition, control and manipulation of laparoscopic tools**

Student: **Karadimos Alexios of Loukas**

Supervising Team:

Associate Professor Dr. Evangelos Dermatas, University of Patras
Professor Dr. Anthony Tzes, NYU Abu Dhabi

Thesis research period:

Abstract

Sunt proident ut incidunt veniam ad excepteur aute nostrud. Anim nulla magna proident dolore non laboris deserunt irure. Aliqua velit quis labore quis nisi. Adipisicing amet dolor officia est ex nulla enim anim minim laboris nisi adipisicing cupidatat. Ut do consequat est ad sint dolore. Eiusmod ipsum nulla do aute culpa ad veniam eu in sunt qui laboris. Ea exercitation et quis fugiat.

Περίληψη

Sunt proident ut incidunt veniam ad excepteur aute nostrud. Anim nulla magna proident dolore non laboris deserunt irure. Aliqua velit quis labore quis nisi. Adipisicing amet dolor officia est ex nulla enim anim minim laboris nisi adipisicing cupidatat. Ut do consequat est ad sint dolore. Eiusmod ipsum nulla do aute culpa ad veniam eu in sunt qui laboris. Ea exercitation et quis fugiat.

Acknowledgements

Sunt proident ut incidunt veniam ad excepteur aute nostrud. Anim nulla magna proident dolore non laboris deserunt irure. Aliqua velit quis labore quis nisi. Adipisicing amet dolor officia est ex nulla enim anim minim laboris nisi adipisicing cupidatat. Ut do consequat est ad sint dolore. Eiusmod ipsum nulla do aute culpa ad veniam eu in sunt qui laboris. Ea exercitation et quis fugiat.

Contents

Contents	8
1 Introduction	10
1.1 Surgical robotics	10
1.1.1 Historical Overview of Surgical robotics	10
1.1.2 Surgical Robotics Procedure	12
1.1.3 Advantages & Disadvantages of Surgical robotics	13
1.2 Problem statement	14
1.3 Bibliography Overview	14
1.4 Methodology & Approach	14
2 Robotic arm Kinematic Analysis	16
2.1 Robotic arm, DH parameters & Forward Kinematics	16
2.1.1 End-effector to tool-tip transformations	18
2.2 Inverse Kinematics	18
2.2.1 Decoupling Technique	19
2.2.2 Workspace constraints & Singularity points	20
2.2.3 Solutions for 7DoF numerically	21
2.2.4 Comparison of Inverse Kinematics Techniques	21
3 Grasping	22
3.1 Gripper & Forward Kinematics	22
3.2 Gripper Inverse Kinematics	22
3.3 Force closure	23
4 Scene and object recognition with Computer Vision	25
4.1 Laparoscopic tool detection	25
4.2 Stereoscopic vision	26
4.3 Calculation of tool position and orientation	27
4.4 Calculation of grasping points	29
4.5 Trocar detection & Estimation of fulcrum point	30
5 Path Planning	31
5.1 Sampling methods	31
5.1.1 RRT Algorithms	31
5.1.2 PRM Algorithms	32
5.2 Pick and place algorithm	32
5.3 Task space analysis	33
6 Trajectory Planning - Laparoscopic tool manipulation	35
6.1 Tool pose & the Fulcrum Effect	35
6.2 Trajectory planning in cartesian coordinates	37
6.2.1 Circular trajectory of tool tip	38
6.2.2 Circular arc trajectory of tool tip	40
6.2.3 Line segment trajectory of tool tip	41
6.2.4 Cubic Spline trajectory of tool tip	42
6.2.5 B-Spline trajectory of tool tip	44
6.3 Trajectory planning in joint angles space	47
6.3.1 Polynomials of 5th order	48
6.3.2 Planning with velocity profiles	48

7 System Control	49
7.1 Firm grasping algorithm & Force control	49
7.2 Visual Servoing	49
7.2.1 Position based servoing	49
7.2.2 Image based servoing	50
8 ROS framework	52
8.1 Introduction to the ROS framework	52
8.2 Gazebo simulation environment	53
8.3 Visualization with RViz	54
8.4 Motion Planning with Moveit	55
8.5 Tools, Packages and Libraries	56
9 Experiments and Results	57
9.1 Robot Planner 1: Simple MoveIt planning	57
9.2 Robot Planner 2: Simulation layout and reachability experiments	57
9.3 Robot Planner 3: Trajectory planning	59
9.3.1 Circular and Circular arc trajectories in task space	59
9.3.2 Line segment trajectories in task space	60
9.3.3 Cubic Spline trajectories in task space	60
9.3.4 B-Spline trajectories in task space	60
9.3.5 Polynomial trajectories in joint space	60
9.3.6 Trajectories in joint space with trapezoidal velocity profile	60
9.3.7 Trajectories in joint space with s-curve velocity profile	60
9.4 Robot Planner 4: Simple cube pick-and-place experiment	60
9.5 Robot Planner 5: Visual servoing	62
9.6 Robot Planner 6	63
9.7 Robot Planner 7	63
9.8 Robot Planner 8: State machine & ROS Actions	63
10 Conclusions and Future Work	64
10.1 Conclusions & Comparison with similar projects	64
10.2 Future Work	64
Appendices	65
A Software and Documentation	65
B Mathematics	65
B.1 Euler angles to Quaternions	65
B.2 Cartesian to spherical coordinates	65
B.3 Spherical to cartesian coordinates	65
Nomenclature	66
List of Figures	67
List of Listings	68
List of Algorithms	68
Bibliography	69

1 Introduction

1.1 Surgical robotics

1.1.1 Historical Overview of Surgical robotics

Surgical robotics is a field of Surgery where the surgeon operates on the patient via a computer, specialised equipment and robotic arms, to which the surgical tools needed for the operation are attached. According to surgical bibliography, robotics and laparoscopic procedures are used in general surgery, cardiothoracic surgeries, colon surgeries, gynecology, neurosurgery and orthopedics.

Robotic mechanisms were first introduced in Medicine, in 1987 with the first laparoscopic surgery of a cholecystectomy. Since then numerous laparoscopic operations have been performed and there has been a lot of improvements and innovations in this field. Such surgical operations are characterised as **minimally invasive**, because the surgical incisions made at the patient are very small and thus the probability of infection of the patient during or after the operation are very small, the hospitalization time is reduced (which means mean better and more efficient use of hospital resources) and the overall recovery of the patient is significantly faster and less painful.

However, traditional laparoscopic mechanisms have some downsides as well. First of all, the surgeon should operate in a mirrored-way, meaning that they should move at the opposite direction from what they saw at the screen (this effect is also known as the **fulcrum effect**), in order to reach the desired point of operation. Earlier laparoscopic tools had less degrees of freedom, which means less flexibility in motion control. Moreover these systems provided limited touch sensibility and feedback to the doctor they were very susceptible to the surgeon's micro movements and tremble.

The first application of robotics in Surgery appears in 1985, when Kwoh et al. [57] used a **PUMA 560**, a standard industrial robotic arm, to perform a neurosurgical biopsy, where the biopsy needle was inserted in the brain and guided with the help of Computed Tomography. This successful application was followed by the **PROBOT** surgical robot [44], which was developed at the Imperial College and used in a prostatectomy operation. Another example of an early surgery robot was the **ROBODOC** system [11] developed by Integrated Surgical Supplies in Sacramento California, which was the first to be used in orthopedics for a hip replacement surgery and was also the first to be approved by the FDA (Food & Drug Administration, organization responsible for medical devices, drugs etc.).



Figure 1: The PUMA 560 robotic arm, which was the first to be used in surgery robotics in 1985

Some other important surgery robots are listed below:

- **AESOP®Endoscope Positioner:** A voice controlled endoscopic system
- **HERMES®Control Center**

- **daVinci Surgical System®:** One of the most popular surgery robots and most used in hospitals. It is a master-slave system, which means that the operation commands are sent unidirectionally from the master console, which is controlled by the surgeon, and are executed by the robot. It also comes with a high definition 3D video feed and advanced manipulator system, one for each hand, called EndoWrist®. It is officially approved by the FDA for laparoscopic surgeries.

- **SOCRATES Robotic Telecollaboration System**

- **Raven-II [27]:** An open platform for collaborative research on surgical robotics.

- **Monarch™ Platform** by Auris Health Inc., an endoscopic system for robotic-assisted bronchoscopy



Figure 2: DaVinci Xi, ©2020 Intuitive Surgical, Inc. Patient Cart with the robotic arms that control the surgical tools



Figure 3: The MonarchTM Platform endoscopic system ¹

1.1.2 Surgical Robotics Procedure

The robotic surgery procedure starts with total anesthesia of the patient. Then the surgeon makes small incisions at the anatomical region of interest, where the procedure will take place. Through these small incisions special tubes, called trocars, are mounted, through which the laparoscopic tools are inserted. After the patient is prepared and after the patient cart, which carries the robotic arms, is successfully positioned and calibrated, the surgeon sits on a console, from where they control the robot via special sensitive joysticks. The surgeon has vision access (often in 3D) to the surgical site via a small endoscopic camera and the video is displayed on the console. In some cases, the surgeon gets force feedback from the joysticks via haptic mechanisms. Haptic force feedback is very important for the doctor in order to have a better sense of the anatomy and the surgical site, and it has gained a lot of interest in the research community.

¹<https://www.aurishealth.com/patients/robotic-bronchoscopy-patient-about-monarch-platform>



Figure 4: DaVinci Xi ©2020 Intuitive Surgical, Inc. Surgeon Console ²

1.1.3 Advantages & Disadvantages of Surgical robotics

Surgical robotics have a huge impact in Medicine and Healthcare in general. Some of the advantages are the following:

- **Minimally Invasive Procedures** which means
 - Smaller incisions
 - Less blood loss
 - Reduced risk of inpatient infection
 - Less pain
 - Faster patient recovery
- Increased **precision** and reduced human errors
 - Smooth and precise movements
 - Detection and correction of errors caused by hand tremble
- **No fulcrum effect** and intuitive manipulation of surgical tools
- **Haptic feedback.** This technology uses small mechanical forces and vibrations to give the user the sense of touch or force. Force feedback gives the ability to the surgeon to understand the mechanical properties of the tissue they operate on, such as resistance and elasticity, and thus distinguish between healthy from unhealthy tissue
- **Teleoperation** (currently in the same room only): the surgeon operates while they sit on a special **ergonomic** console, which makes the long procedures more comfortable and efficient.

²<https://www.intuitive.com/en-us/about-us/press/press-resources>

1.2 Problem statement

The goal of this thesis is to design the kinematic models and algorithms necessary for a robotic arm to detect, pick and manipulate a laparoscopic surgical tool. To achieve these goals the robotic arm must successfully do the following:

- Use computer vision to detect the scene and laparoscopic tools and with the help of stereoscopic vision, calculate the position and orientation of the center of mass of each tool, with respect to the universal reference frame
- Calculate the contact points on the tool, on which the fingers of the gripper will be placed, so that there is a firm grasp (force closure)
- Calculate the path from the tools' table to the surgical site table
- Calculate the trajectory that needs to be executed when the tool is inserted in the trocar. This is a special type of trajectory, because the motion is constrained and is known in the bibliography as a **Remote Center of Motion** (RCM) control. The constraint is that, at each time one point of the inserted tool must coincide with the RCM point (this point will also be referenced in this thesis as the center of the trocar, or the fulcrum reference frame point).

1.3 Bibliography Overview

1.4 Methodology & Approach

Robotics and especially surgery robotics is a multi-disciplinary field and a robot has many subsystems. A very important approach to successfully design a robotic arm to execute a surgery task is to subdivide the task in multiple smaller submodules and design, implement and test each submodule separately and then combine them together for end-to-end testing.

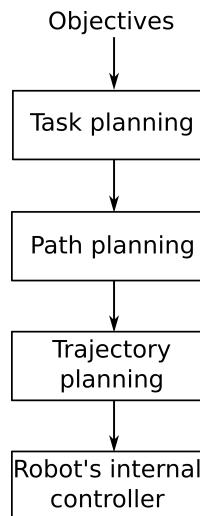


Figure 5: Motion planning pipeline

Tools & Software used in this thesis:

- Matlab & Matlab toolboxes: ROS Toolbox, Robotics Toolbox
- ROS Framework
- MoveIt
- Gazebo simulator

- VREP simulator
- OpenCV

Methodology of conducting this thesis:

- Mathematical calculations for Kinematics and Motion planning
- Mathematics validation with Matlab
- Quick prototyping and testing with Matlab and/or VREP
- Implementation in ROS frameworks, by splitting the end-to-end robotic operation in smaller experiments and tasks

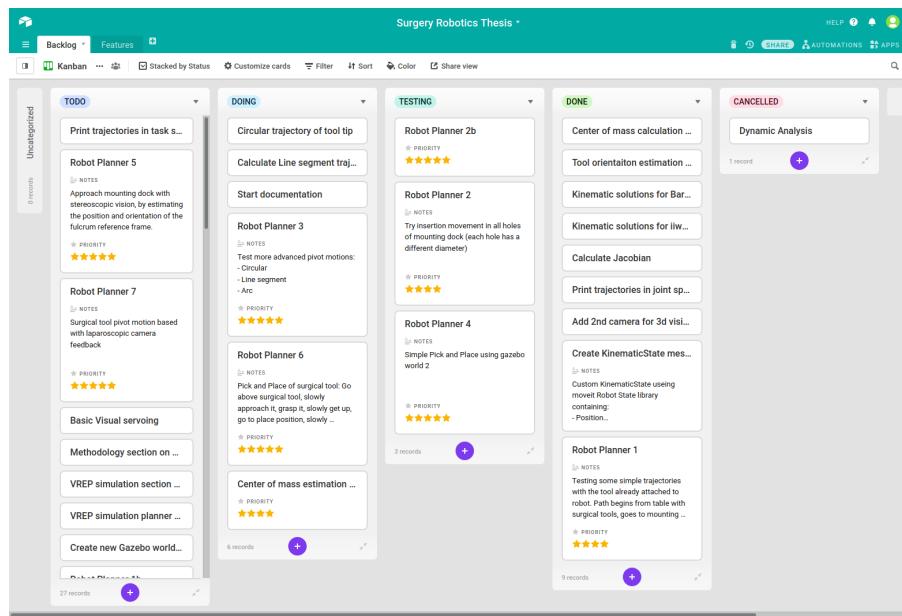


Figure 6: Kanban view of backlog tasks to organize all features, requirements and tasks needed to complete this thesis. The tool used to keep track of all tasks is Airtable

2 Robotic arm Kinematic Analysis

2.1 Robotic arm, DH parameters & Forward Kinematics

The Forward Kinematics problem seeks to specify the way in which the robot's links are connected together. In other words, we want to specify the transformations between every pair of consecutive robot links. It is known that the forward kinematics of a robot can be calculated given only four parameters for each link. These parameters are known in robotics as the **Denavit-Hartenberg** (DH) parameters. Two of these parameters describe the link itself and the other two describe the link's relation to the neighboring link.

- The **length** L_i of the i -th link is equal to the distance between the axes z_i and z_{i+1}
- The **twist angle** α_i of the i -th link, is the angle between the axes z_i and z_{i+1}
- The **rotation angle** θ_i of the link $\{i\}$ with respect to the $\{i - 1\}$ link, is the angle between the axes x_{i-1} and x_i
- The **distance** d_i of the link $\{i\}$ with respect to the $\{i - 1\}$ link, is the distance between the axes x_{i-1} and x_i

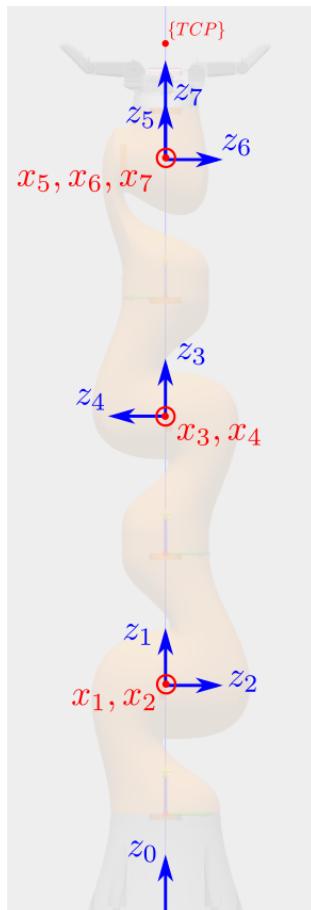


Figure 7: Joint reference frames of the KUKA iiwa14 robot

i	θ_i (rad)	L_{i-1} (m)	d_i (m)	α_{i-1} (rad)
1	θ_1	0	0.36	0
2	θ_2	0	0	$-\pi/2$
3	θ_3	0	0.36	$\pi/2$
4	θ_4	0	0	$\pi/2$
5	θ_5	0	0.4	$-\pi/2$
6	θ_6	0	0	$-\pi/2$
7	θ_7	0	0	$\pi/2$

Using the DH parameters of the above table, one can calculate the transformation matrix between two consecutive links, and is calculated as the following

$${}^{i-1}T_i = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & L_{i-1} \\ s\theta_i c a_{i-1} & c\theta_i c a_{i-1} & -s a_{i-1} & -s a_{i-1} d_i \\ s\theta_i s a_{i-1} & c\theta_i s a_{i-1} & c a_{i-1} & c a_{i-1} d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.1.1)$$

When all the neighboring transformations are calculated, then one can calculate the total transformation 0T_N , which represents the position and the orientation of the local coordinate system of the end-effector with respect to the global coordinate system of the robot's base. The orientation is the upper-left 3x3 matrix and the position is given by the fourth column of the matrix 0T_N .

$${}^0T_N = {}^0T_1 \cdot {}^1T_0 \cdots {}^{N-1}T_N \quad (2.1.2)$$

All transformations ${}^{i-1}T_i$ are members of a special set of matrices (Lie Group), called **Special Euclidean Group**

$${}^{i-1}T_i \in SE(3) = \left\{ \begin{bmatrix} R & \mathbf{p} \\ \mathbf{0} & 1 \end{bmatrix} : R \in SO(3), \mathbf{p} \in \mathbb{R}^3 \right\}$$

where $SO(3)$ is another Lie Group called **Special Orthogonal Group**

$$SO(3) = \left\{ R \in \mathbb{R}^{3 \times 3} : R^{-1} = R^\top, \det(R) = 1 \right\}$$

The properties of $SE(3)$ and $SO(3)$ are very useful in all the calculations of various transformations because it can reduce the amount of matrix operations and also speeds up the calculation of inverse matrices.

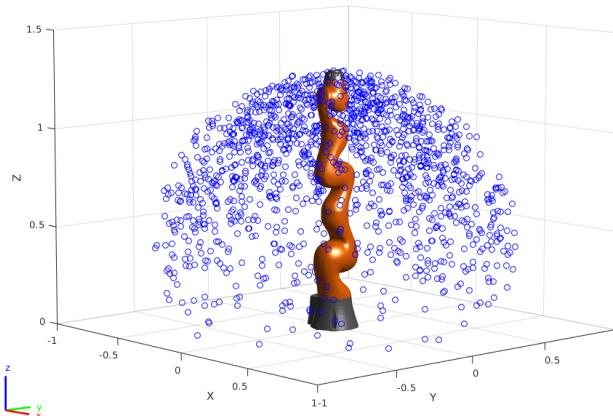


Figure 8: Approximation of the KUKA iiwa14 workspace, calculated with Forward Kinematics by randomly sampling the value ranges of the joints.

2.1.1 End-effector to tool-tip transformations

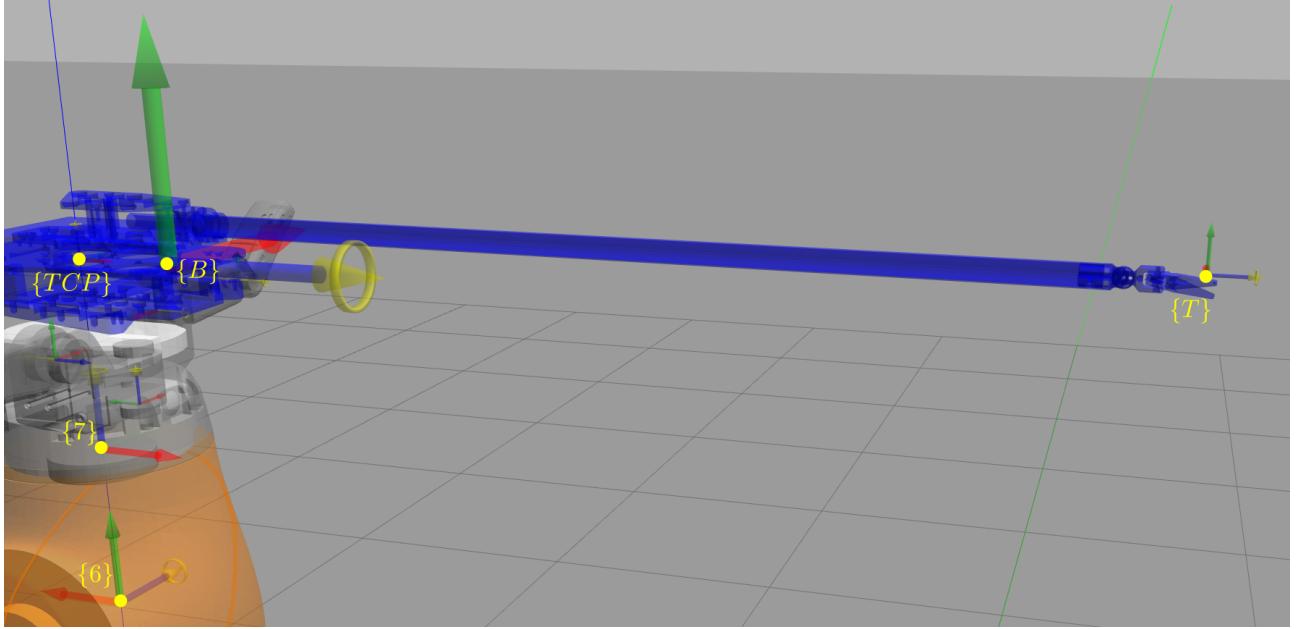


Figure 9: Reference frames of last link $\{7\}$, end-effector $\{TCP\}$, surgical tool base (center of mass) $\{B\}$ and tool-tip frame $\{T\}$

$$\begin{aligned} {}^7T_{TCP} &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.094 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ {}^{TCP}T_B &= \begin{bmatrix} 1 & 0 & 0 & 0.05 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ {}^BT_T &= \begin{bmatrix} 1 & 0 & 0 & -0.01 \\ 0 & 1 & 0 & 0.022 \\ 0 & 0 & 1 & 0.455 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

2.2 Inverse Kinematics

The Inverse Kinematics problem is one of the most important problems a roboticist must solve to design trajectories and control the robot's motion to do useful tasks. The IK problem seeks to find those joint values that make the robot's end effector to be at a specific desired position and orientation or equivalently the input to this problem is the desired pose (position and orientation of the end-effector) and the output are the solutions for the joints. For a robot of six degrees of freedom, i.e. six actuators that move independently and are positioned in such a way so that their axes are not aligned, the solutions to the I.K. problem are typically 8. For robots with more actuators than the 6 degrees of freedom of motion in 3D space (3 for position and 3 for orientation), like the robot of this thesis which has 7 degrees of freedom, the I.K. problem has infinite solutions for a given pose, which means that an additional constraint is required to find a specific solution. This extra degree of freedom is very useful in finding kinematic solutions that are optimal under some circumstances and are also useful in avoiding **singularity points** where the robot loses some degrees of freedom.

2.2.1 Decoupling Technique

In this section the inverse kinematics problem is solved for only the 6 out of the 7 total degrees of freedom. The third joint is not used in this analysis and its angle is set to zero $\theta_3 = 0$. The rest of the joints form a special kind of kinematic chain that can be solved using the decoupling technique. In this technique the Inverse kinematics problem is split to 2 separate subproblems, one for the position and one for the orientation of the end-effector. This technique can be applied in this case because the axes of the 3 last joints intersect at the same point and they form an Euler wrist.

To solve for the joints' angles, the transformation matrix 0T_7 of the end-effector with respect to the robot's base is required. Usually the transformation ${}^U T_{tcp}$ is known, which is the pose of Tool's center point (TCP) with respect to the Universal Coordinate Frame $\{U\}$ from which the required 0T_7 can be calculated

$${}^U T_{tcp} = {}^U T_0 \ {}^0 T_7 \ {}^7 T_{tcp} \quad (2.2.1)$$

$${}^0 T_7 = {}^U T_0^{-1} \ {}^U T_{tcp} \ {}^7 T_{tcp}^{-1} \quad (2.2.2)$$

$${}^0 T_7 = \begin{bmatrix} R_t & \mathbf{p}_t \\ 0 & 1 \end{bmatrix} \quad (2.2.3)$$

where ${}^U T_0$, ${}^7 T_{tcp}$ are translation transformations by a constant distance and R_t , \mathbf{p}_t are the target's orientation and position respectively.

$${}^0 \mathbf{p}_5 = {}^0 T_4 {}^4 \mathbf{p}_5 = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \quad (2.2.4)$$

$$\theta_1 = \begin{cases} \text{atan2}(p_y, p_x) \\ \pi - \text{atan2}(p_y, p_x) \end{cases} \quad (2.2.5)$$

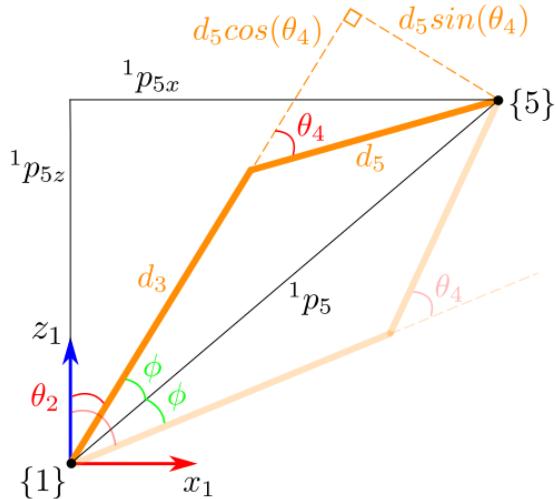


Figure 10: Calculation of angles θ_2, θ_4

$$\varphi = \text{acos} \left(\frac{d_3^2 + \|{}^1 p_5\|^2 - d_5^2}{2d_3 \|{}^1 p_5\|} \right) \quad (2.2.6)$$

$$\theta_2 = \text{atan}2\left(\sqrt{p_x^2 + p_y^2}, {}^1p_{5z}\right) \pm \varphi \quad (2.2.7)$$

$$c_4 = \frac{\|{}^1p_5\|^2 - d_3^2 - d_5^2}{2d_3d_5}$$

$$\theta_4 = \text{atan}2\left(\pm\sqrt{1 - c_4^2}, c_4\right) \quad (2.2.8)$$

Once $\theta_1, \theta_2, \theta_3, \theta_4$ are known, the orientation matrix of the wrist can be calculated as following

$$R_{target} = \begin{bmatrix} i_x & j_x & k_x \\ i_y & j_y & k_y \\ i_z & j_z & k_z \end{bmatrix}$$

$$\theta_6 = \text{atan}2\left(\pm\sqrt{1 - k_y^2}, k_y\right) \quad (2.2.9)$$

$$\theta_7 = \text{atan}2(-j_y, i_y)$$

$$\theta_5 = \text{atan}2(-k_z, k_x)$$

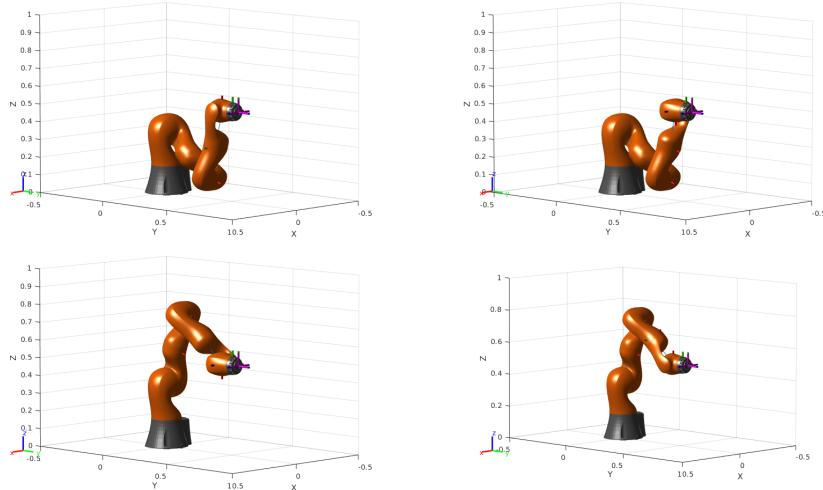


Figure 11: The first 4 out of 8 solutions of the Inverse Kinematics problem, using the decoupling technique

2.2.2 Workspace constraints & Singularity points

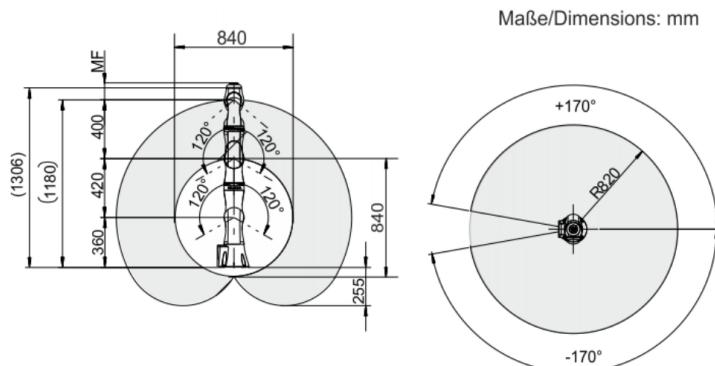


Figure 12: KUKA iiwa LBR14 workspace dimensions

Singularity points:

- When $p_x^2 + p_y^2 = 0$ then the end-effector lies on the z-axis and θ_1 is not defined
- When $\sin(\theta_6) = 0$ then the angles θ_5, θ_7 are not defined

2.2.3 Solutions for 7DoF numerically

To solve numerically for the joint angles, we use a relationship between the derivative of the spatial coordinates and the joint derivatives. These two vectors are related by a matrix known as the **Jacobian** and is given by the following equation

$$\dot{\mathbf{x}} = J(\mathbf{q})\dot{\mathbf{q}} \quad (2.2.10)$$

where

$$J(\mathbf{q}) = \begin{bmatrix} \frac{\partial \mathbf{f}_1(\mathbf{q})}{\partial q_1} & \dots & \frac{\partial \mathbf{f}_1(\mathbf{q})}{\partial q_7} \\ \vdots & \ddots & \vdots \\ \frac{\partial \mathbf{f}_6(\mathbf{q})}{\partial q_1} & \dots & \frac{\partial \mathbf{f}_6(\mathbf{q})}{\partial q_7} \end{bmatrix} \quad (2.2.11)$$

and the functions $\mathbf{f}_i(\mathbf{q})$ are calculated using the robot's forward kinematics.

$J(\mathbf{q})$ is non rectangular and thus non-invertible, due to the extra degree of freedom. Instead of the inverse of the Jacobian the pseudoinverse is calculated which by the equation

$$J^\dagger = J^\top (JJ^\top)^{-1} \quad (2.2.12)$$

Algorithm 1: Newton-Raphson numerical method

```

initialize  $\mathbf{q}^0 \in \mathbb{R}^7$  with an initial guess, given a desired position and orientation  $\mathbf{x}_d \in \mathbb{R}^6$ ;
set error  $e = \mathbf{x}_d - f(\mathbf{q}^i)$ ;
set error tolerance  $\varepsilon$  with some small value;
while  $e \geq \varepsilon$  do
    set  $\mathbf{q}^{i+1} = \mathbf{q}^i + J^\dagger(\mathbf{q})e$ ;
     $i \leftarrow i + 1$ ;
end
```

If the numerical method has more than one solutions, then the method will converge to the one which is closest to the initial guess or it will not converge at all, if the initial error is too big. One way to get some approximate initial guesses is if we use a solution from the previous section (which solves for 6 dof) and use these solutions as input to this numerical method.

The numerical calculation of the inverse kinematics can also be calculated using the geometric Jacobian, given by the following equation

$$\begin{aligned}
 J_v &= J_v(\mathbf{q}) = [J_{v1}, J_{v2}, \dots, J_{v7}] \in \mathbb{R}^{6 \times 7} \\
 J_i &= \begin{bmatrix} {}^0\mathbf{z}_i \times ({}^0\mathbf{p}_8 - {}^0\mathbf{p}_i) \\ {}^0\mathbf{z}_i \end{bmatrix}
 \end{aligned} \quad (2.2.13)$$

2.2.4 Comparison of Inverse Kinematics Techniques

3 Grasping

3.1 Gripper & Forward Kinematics

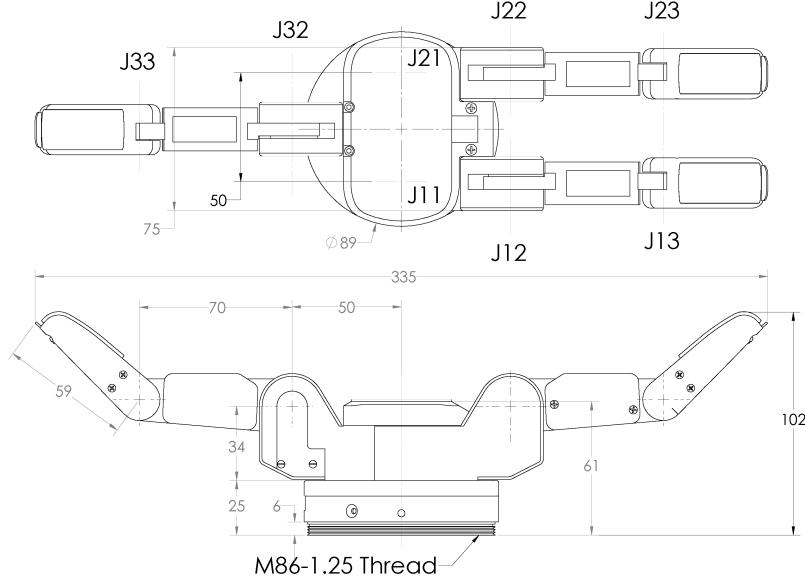


Figure 13: Barrett Hand gripper (model BH8-282) dimensions

i	θ_i (rad)	L_{i-1} (m)	d_i (m)	α_{i-1} (rad)
1 (J11)	θ_{11}	0	0	0
2 (J12)	θ_{12}	0	0	0
3 (J13)	θ_{13}	0	0	0

i	θ_i (rad)	L_{i-1} (m)	d_i (m)	α_{i-1} (rad)
1 (J21)	θ_{21}	0	0	0
2 (J22)	θ_{22}	0	0	0
3 (J23)	θ_{23}	0	0	0

i	θ_i (rad)	L_{i-1} (m)	d_i (m)	α_{i-1} (rad)
1 (J32)	θ_{32}	0	0	0
2 (J33)	θ_{33}	0	0	0

3.2 Gripper Inverse Kinematics

The following Inverse Kinematics analysis refers to one finger of the Barrett Hand gripper, which has 3 revolute joints. Finger 3 has only 2 revolute joints for which the angle solutions are the same with the solutions of the last 2 joints of the other fingers. Let

$$\mathbf{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$$

be the position of the grasp point for one finger. The first angle can easily be calculated as

$$\varphi_1 = \text{atan2}(p_y, p_x) \quad (3.2.1)$$

Next, we calculate the third angle based on the law of cosines (see fig.)

$$\cos\left(\pi - \varphi_3 - \frac{\pi}{4}\right) = \frac{L_2^2 + L_3^2 - p^2}{2L_2L_3}$$

$$\cos\left(\varphi_3 + \frac{\pi}{4}\right) = \frac{p^2 - L_2^2 - L_3^2}{2L_2L_3}$$

$$\varphi_3 = \text{atan}2\left[\pm\sqrt{1 - \left(\frac{p^2 - L_2^2 - L_3^2}{2L_2L_3}\right)^2}, \frac{p^2 - L_2^2 - L_3^2}{2L_2L_3}\right] - \frac{\pi}{4} \quad (3.2.2)$$

In a more general case, the first argument of the *atan2* function in the expression of φ_3 could also be negative, but in this case this second solution is rejected, because due to mechanical constraints, this angle can't be negative. After having calculated φ_3 we can calculate φ_2

$$\tan(\psi + \varphi_2) = \frac{p_z}{\sqrt{p_x^2 + p_y^2}}$$

$$\tan(\psi) = \frac{L_3 \sin\left(\varphi_3 + \frac{\pi}{4}\right)}{L_2 + L_3 \cos\left(\varphi_3 + \frac{\pi}{4}\right)}$$

$$\varphi_2 = \text{atan}2\left(p_z, \sqrt{p_x^2 + p_y^2}\right) - \text{atan}2\left[L_3 \sin\left(\varphi_3 + \frac{\pi}{4}\right), L_2 + L_3 \cos\left(\varphi_3 + \frac{\pi}{4}\right)\right] \quad (3.2.3)$$

3.3 Force closure

In order to achieve a firm grasp of the surgical tool, the gripper fingers must be positioned in such a way around the object so that there is **force closure**. According to the **Nguyen** theorem, a planar object that is constrained by 2 points, is in force closure if the line which connects the two constraint points lies inside both friction cones of these points. A simplistic explanation of a friction cone is that it is a set of forces that result the contact point to be in friction and all other forces outside of the cone will result in sliding.

For the three dimensional case, a spatial body which is constrained by 3 contact points. These 3 contact points have friction and they define a unique plane S and also the 3D friction cone of each point intersects the plane S in a 2D cone. Then the spatial body is in force closure if and only if the plane S is in a planar force closure group.

To check for force closure, one must know the contact points and the direction of the forces that are applied there. In the Gazebo simulator program that was used in this thesis, the information of the contact between a gripper's finger and an object is easily available as shown in listing 1. However this kind of information is much harder to acquire in real scenarios. The Barrett gripper that is used in this thesis comes with arrays of tactile sensors in each surface of the gripper and there are methodologies to use these spatially distributed sensors to approximate the intensity as well as and most importantly the direction of the forces [40].

Listing 1: Example ROS message with collision/contact information between one finger of the gripper and one surgical tool

```

header:
  seq: 327978
  stamp:
    secs: 564
    nsecs: 468000000
  frame_id: "world"
states:
-
  info: "Debug: geom:surgical_tool4::surgical_tool_link::collision ..."
  collision1_name: "surgical_tool4::surgical_tool_link::collision"
  collision2_name: "robot::bh_finger_33_link::bh_finger_33_link_collision"
  wrenches:
-
  force:

```

```
x: -1.42463913929e-91
y: 1.47711790023e-91
z: -1.58225043603e-91
torque:
x: 5.44545932609e-93
y: -3.73283810062e-93
z: -8.38783555513e-93
total_wrench:
force:
x: -1.42463913929e-91
y: 1.47711790023e-91
z: -1.58225043603e-91
torque:
x: 5.44545932609e-93
y: -3.73283810062e-93
z: -8.38783555513e-93
contact_positions:
-
x: -0.00130886196029
y: -0.587570558845
z: 1.0584908858
contact_normals:
-
x: 0.041868681791
y: -0.00620219851863
z: -0.999103871586
depths: [4.625789111756262e-07]
```

4 Scene and object recognition with Computer Vision

At this section we explore ways to detect and recognize the surgical tools as well as other objects of the simulation scene. To reduce the complexity of this thesis and focus on the more important features of this thesis, we assume in the simulation that the surgical tools are blue and the mounting dock, where the tools will be placed, is green. These assumptions make the scene and object recognition much easier without the need of more advanced image processing and/or machine learning recognition algorithms.

Camera setup used in this thesis:

- 2 HD RGB cameras with resolution 1280×720
- near clipping plane: 0.02
- far clipping plane: 300
- horizontal FoV (field of view): 1.396
- update rate: 30fps

4.1 Laparoscopic tool detection

In order to detect the shape of the tool there are some standard steps that need to be executed. After having loaded the input image we convert it to grayscale, so that we can work on only one channel instead of 3 color channels and thus reduce the amount of calculations. Also for the purposes of extracting the shape of an object, the color doesn't have a very significant role in the algorithm. Next step is to remove the unwanted noise. In this thesis we only assume that the video frames have only Additive White Gaussian Noise (AWGN). To remove some of the noise we use a moving average filter (the filter is also known as a kernel), which is convoluted around the whole image. The filter that was used is the following 3-by-3 matrix

$$h = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

the output, filtered image is the result of the convolution of the image with the filter and is calculated as following

$$g(i, j) = \sum_{k,l} f(i + k, j + l)h(k, l) \quad (4.1.1)$$

where $g(\cdot, \cdot)$ is the output image and $f(\cdot, \cdot)$ is the input image.

After the noise is removed the image is getting binarized. To do that, we set a threshold, below which the pixels will be black and the rest will be white. This conversion to binary format, makes it easier to extract the boundaries of the black shapes, which will correspond to the boundaries of the objects of the initial image.

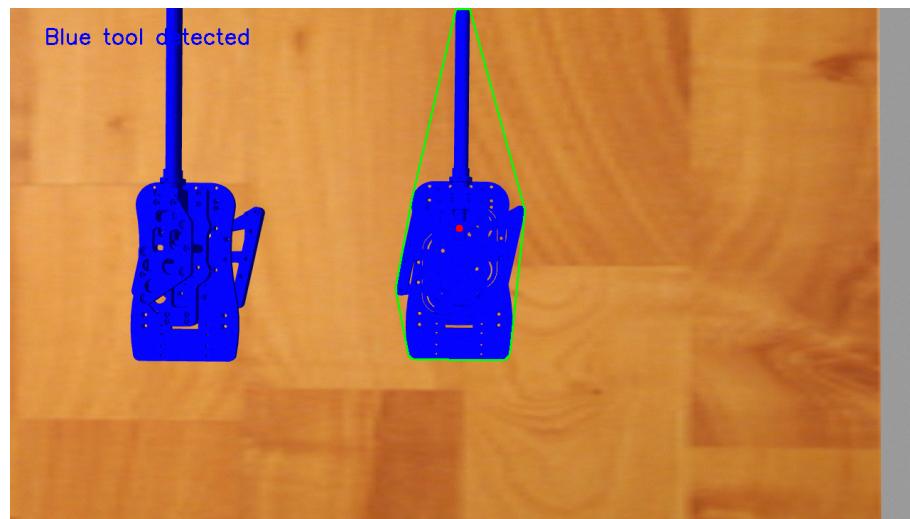


Figure 14: Simple tool detection in simulation based on color, using OpenCV. The green polygon is the convex hull, and the red point is the estimated center of mass

4.2 Stereoscopic vision

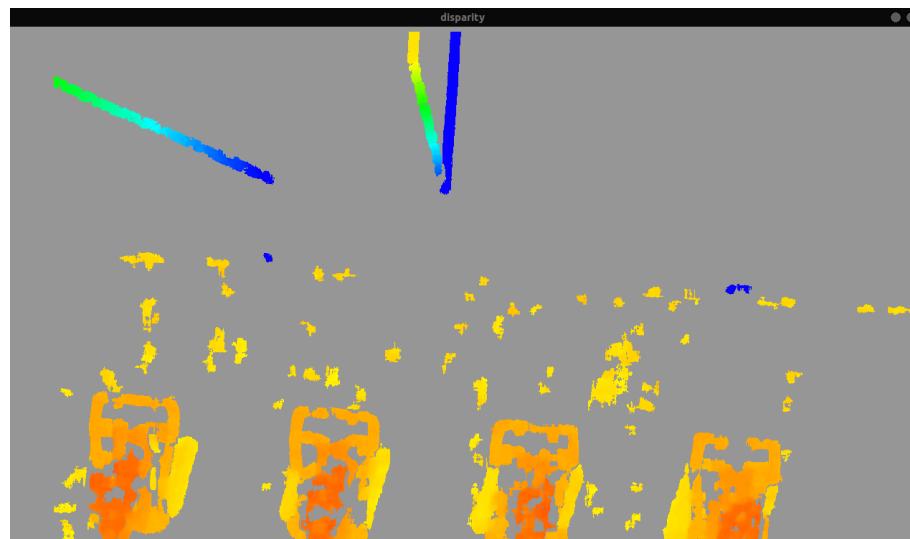


Figure 15: Disparity image calculated from the 2 cameras

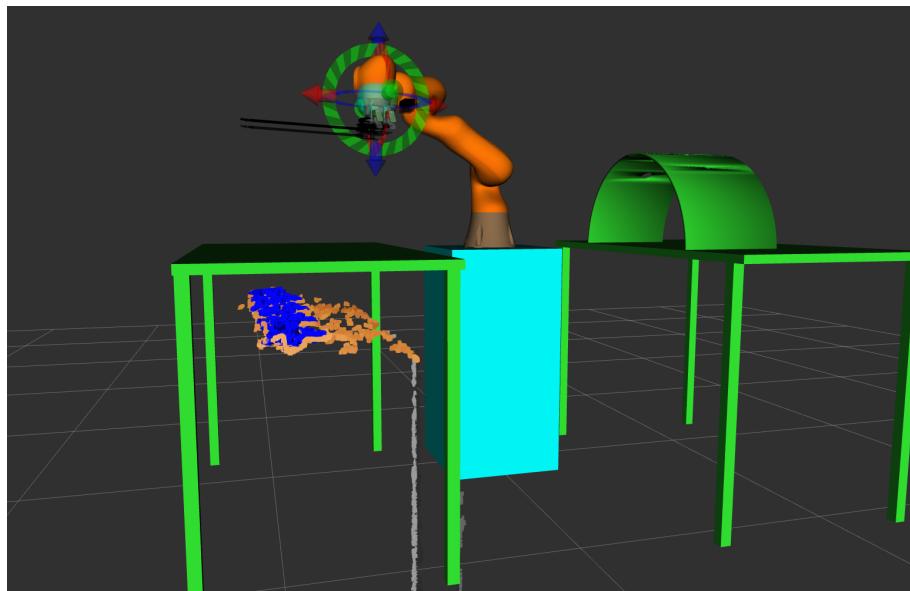


Figure 16: Point cloud of surgical tools, generated from the 2 cameras and visualized in RViz

4.3 Calculation of tool position and orientation

In order for the gripper to grasp correctly the laparoscopic tool, it is required to calculate the tool's position and orientation in the pixel space which must then be converted with respect to the robot's workspace. From all the pixels that have been classified as part of the laparoscopic tool, one can estimate the center of mass and two perpendicular vectors attached to that point that define the orientation. The center of mass is simply the average of the (x, y) coordinates of all the tool's pixels

$$(\bar{x}, \bar{y}) = \left(\frac{1}{N} \sum_{i=1}^N x_i, \frac{1}{N} \sum_{i=1}^N y_i \right) \quad (4.3.1)$$

The easiest way to calculate the center of mass is by calculating the average using the first moments of the contour points. However, since the contour is only using the boundary of the object and not its area, this method is not very accurate. To get a more accurate value for the center of mass, one must use the pixels that are inside the detected object's contour. The simplest way (but most expensive) to get the inside pixels of the tool is to loop over all the pixels and for each pixel check if it is inside the polygon (point inside polygon test). To make this method even faster, one can take a sample of the total pixels, for example check one in every 10 pixels in the x and y coordinates, which means reducing the time complexity to one tenth.

Taking this approach a step further in optimization, one can iterate not in all the video frame pixels but only those pixels that are inside the **Region of Interest** of the tool. The Region of Interest, also known as ROI, is a widely used structure in computer vision and is simply a bounding box (rectangle) that fits exactly (or is a bit bigger than) an object or part of the image frame that we want to study. Having already calculated the contour of the surgical tool and its convex hull we can easily calculate this bounding box. For this calculation we prefer to use the convex hull, because it often contains much less pixels than the contour. We iterate over all the pixels of the convex hull and we get the minimum and maximum x and y coordinates. The combination of these four values is the desired ROI. Since we now have access to the tool's ROI, we can iterate and sample the pixels inside it (and not all pixels as we did before) to get some of the pixels of the tool so that we can more accurately calculate its center of mass and orientation vectors.

The two orientation vectors are the eigenvectors of the covariance matrix of the above pixels. Let \mathbf{a}, \mathbf{b} be the orientation vectors, then \mathbf{a}, \mathbf{b} are solutions of the equation

$$C\mathbf{v} = \lambda\mathbf{v} \quad (4.3.2)$$

where C is the covariance matrix given by

$$C = \begin{bmatrix} \sigma(x, x) & \sigma(x, y) \\ \sigma(y, x) & \sigma(y, y) \end{bmatrix} \quad (4.3.3)$$

$$\sigma(x, y) = \frac{1}{n-1} \sum_{i=1}^N (x_i - \bar{x})(y_i - \bar{y}) \quad (4.3.4)$$

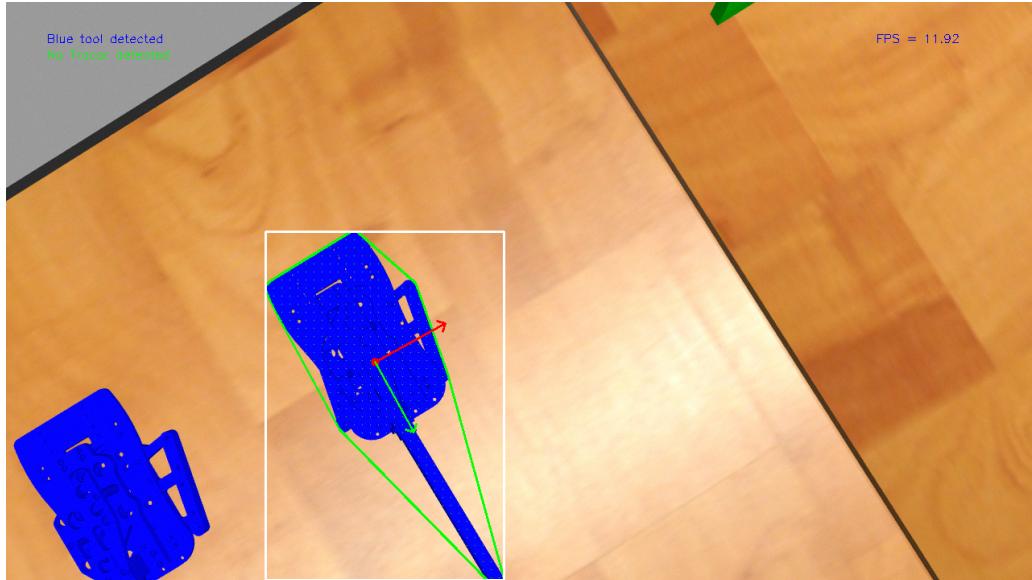


Figure 17: Estimation of tool's pose (position and orientation). The red dot is the center of mass and attached to that are the two orientation vectors of the tool. The green polygon is the convex hull of the tool and the white rectangle is it's ROI as calculated from the convex hull

4.4 Calculation of grasping points

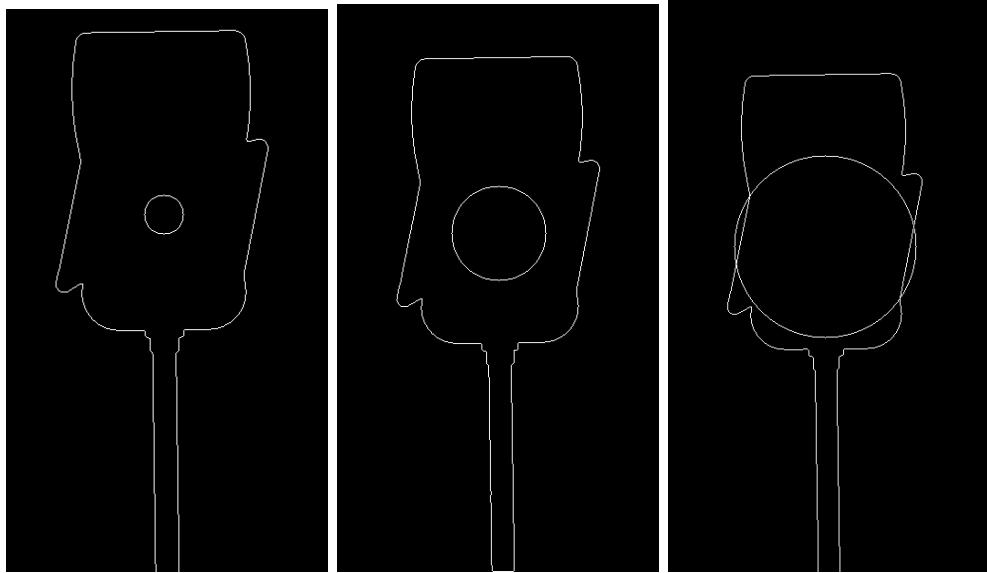


Figure 18: Finding candidate grasping points from the intersections of a growing circle and the contour of the detected surgical tool

The calculation of the candidate grasping points is a problem where we seek to find the points that lie on the contour of the detected object such that the gripper can make a good grasp of the object and preferably with force closure. The method that was implemented in this thesis for a simplistic calculation of these points is by using a **growing circle**. This circle is initiated with a small radius and has at all times its center at the center of mass of the detected tool. At each step of the method the radius is incremented by a fixed amount and then we check if this circle has any intersection with the contour of the detected tool. If no intersection is found the method proceeds with a new radius. The method is terminated when at least 3 intersections are found, and that's because the gripper that is used in this thesis has three fingers.

$$\mathbb{G} = \arg \max_{(x,y)} I_1(x,y) \odot I_2(x,y) \quad (4.4.1)$$

At the equation 4.4.1, the set of intersection points \mathbb{G} is calculated. Given one binary image $I_1(x,y)$ which contains the contour of the detected tool (all white pixels are the contour) and one second binary image $I_2(x,y)$ which contains the growing circle, then the set of intersection points are the (x,y) coordinates where the Hadamard product $I_1(x,y) \odot I_2(x,y)$ (element-wise product) of the two binary images is maximum (or equivalently has a value of 1).

It is often the case that for one intersection multiple pixels are returned from this method due to the "pixelated nature" of all curves. To filter all these pixels and get only one pixel for each intersection point we initialize a list of the final points with the first pixel found and then iterate over all of the other pixels. For each pixel we check if it has a Manhattan distance bigger than a threshold (e.g. 10) from the pixels that are already added in the list of the final pixels. If two pixels have a Manhattan distance bigger than a selected threshold then we consider them to be two different intersection points. The Manhattan distance of two points $\mathbf{p}_1 = (x_1, y_1)$, $\mathbf{p}_2 = (x_2, y_2)$ is given by

$$d_1(\mathbf{p}_1, \mathbf{p}_2) = |x_1 - x_2| + |y_1 - y_2| \quad (4.4.2)$$

4.5 Trocar detection & Estimation of fulcrum point

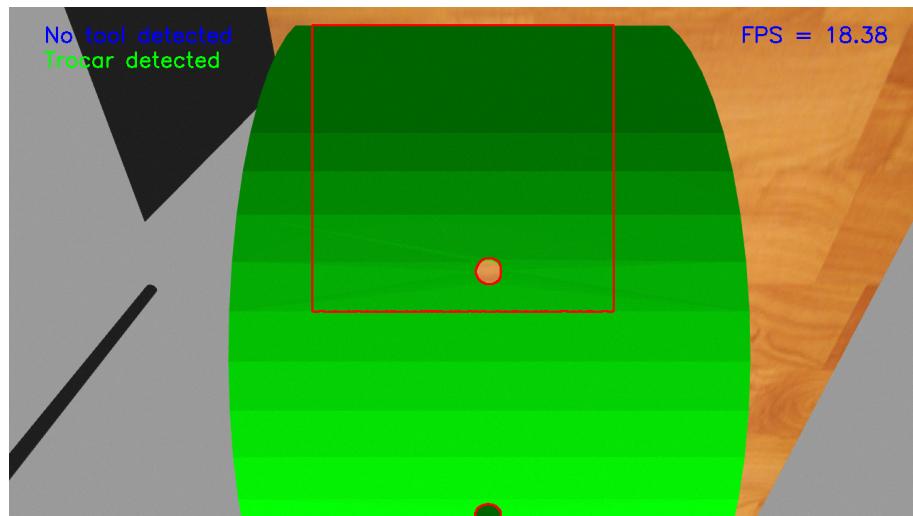


Figure 19: Simple trocar detection in simulation based on color, using OpenCV. In simulation, the trocar is simply considered to be a small cylindrical hole and it's center is the fulcrum point

5 Path Planning

Path Planning is a geometric problem, where it is desired to find a path from a starting point to a goal point and also satisfying a set of constraints, such as: restricting the solutions inside the robot's configuration space, avoiding obstacles in the task space, avoiding singularity points and respecting the robot's joint limits.

5.1 Sampling methods

The path planning algorithms that were mostly used in this thesis belong to the category of sampling methods. These methods use random functions to choose a sample from the configuration space or the state space. Sampling methods differ from the deterministic grid methods which, which discretize the whole space. Sampling methods are less computationally expensive than the grid methods, but they do not deliver optimal solutions like the latter.

5.1.1 RRT Algorithms

The **Rapidly-exploring Random Trees** algorithm is a sampling planning method that searches for an obstacle-free motion plan from an initial state x_{init} to a set of goal states \mathcal{X}_{goal} . We refer to a set of goal states, because apart from the one desired goal state there can be other neighbor states that are within the allowed position and orientation tolerances.

Algorithm 2: RRT Algorithm

```

foreach replanning attempt do
    initialize vertices  $V \leftarrow \{x_{init}\}$ ;
    initialize edges  $E \leftarrow \emptyset$ ;
    initialize search tree  $T \leftarrow (V, E)$ ;
    while time  $\leq maxPlanningTime$  do
         $x_{rand} \leftarrow \text{getSampleStateFrom}(\mathcal{X})$ ;
         $x_{nearest} \leftarrow \text{getNearestNodeInTreeToState}(T, x_{rand})$ ;
         $x_{new} \leftarrow \text{findLocalPlanFromTo}(x_{nearest}, x_{rand})$ ;
        if  $\text{isPathCollisionFree}(x_{nearest}, x_{rand})$  then
             $V \leftarrow V \cup \{x_{new}\}$ ;
             $E \leftarrow E \cup \{(x_{nearest}, x_{rand})\}$ ;
            if  $x_{new} \in \mathcal{X}_{goal}$  then
                return SUCCESS and path plan  $T = (V, E)$  ;
            end
        end
    end
end
return FAILURE and  $T = (V, E)$  ;

```

Other variations of the RRT Algorithm, which are also available in the OMPL library, included in the MoveIt library of ROS framework are:

- **TRRT** Transition-based RRT
- **BiTRRT** Bidirectional Transition-based RRT
- **RRT***
- **RRTConnect** with is the default OMPL path planner in ROS
- **LBTRRT** Lower Bound Tree RRT

5.1.2 PRM Algorithms

The **Probabilistic Roadmap** (PRM) algorithm is a sampling planning method that constructs a roadmap representation of \mathcal{C}_{free} before searching for a solution. After the roadmap is successfully built, then the algorithm searches for a solution using a traditional graph-based search algorithm. A very important aspect of this algorithm is how the sampling of the free configuration space will be done. The sampling is usually performed using a uniform distribution except from the regions close to objects where the sampling is more dense.

Algorithm 3: PRM roadmap construction (preprocessing phase)

```

initialize vertices  $V \leftarrow \{x_{init}\}$ ;
initialize edges  $E \leftarrow \emptyset$ ;
initialize roadmap graph  $G \leftarrow (V, E)$ ;
for  $i = 1, \dots, n$  do
     $x_{rand,i} \leftarrow \text{getSampleStateFrom}(\mathcal{X})$ ;
     $\mathcal{N}(x_{rand,i}) \leftarrow \text{getKNearestNeighbors}(G = (V, E), x_{rand,i})$ ;
     $V \leftarrow V \cup \{x_{rand,i}\}$ ;
    foreach  $x \in \mathcal{N}(x_{rand,i})$  do
        if there is no edge between  $x$  and  $x_{rand,i}$  then
            if  $\text{isPathCollisionFree}(x_{nearest}, x_{rand,i})$  then
                |  $E \leftarrow E \cup \{(x_{rand,i}, x), (x, x_{rand,i})\}$ 
            end
        end
    end
end
return  $G = (V, E)$ 
```

Other variations of the PRM Algorithm, which are also available in the OMPL library, included in the MoveIt library of ROS framework are:

- PRM*
- LazyPRM
- LazyPRM*

5.2 Pick and place algorithm

Algorithm 4: Pick and Place algorithm

```

foreach surgical tool do
    /* Plan the Pick pipeline */  

    set grasp pose;  

    set pre-grasp approach;  

    set post-grasp retreat;  

    set posture of eef before grasp (open gripper);  

    set posture of eef during grasp (closed gripper);  

    /* Plan the Place pipeline */  

    set place location pose;  

    set pre-place approach;  

    set post-grasp retreat;  

    set posture of eef after placing object;  

    Plan pick and place paths;  

end
```

If the pick and place algorithm targets small objects, such as cubes or spheres or other small convex objects then the path planning is straightforward. In the case where, the object to pick and place has at least one dimension that is bigger than the others like a rod or other long objects, such as the

surgical tools, used in this thesis, then the path planning becomes more complicated, because of the almost certain collisions of the tool with the links of the rest of the robot (the link of the end-effector will probably not collide with the tool).

5.3 Task space analysis

Dexterity analysis for tool's task space

$$\mathcal{D} = \mathcal{L}_q \mathcal{M} \quad (5.3.1)$$

where

$$\mathcal{M} = \sqrt{\det(JJ^\top)} \quad (5.3.2)$$

$$\mathcal{L}_q = 1 - \exp \left\{ -\kappa \prod_{i=1}^{n_k} \frac{(q_i - q_{i,\min})(q_{i,\max} - q_i)}{(q_{i,\max} - q_{i,\min})^2} \right\} \quad (5.3.3)$$

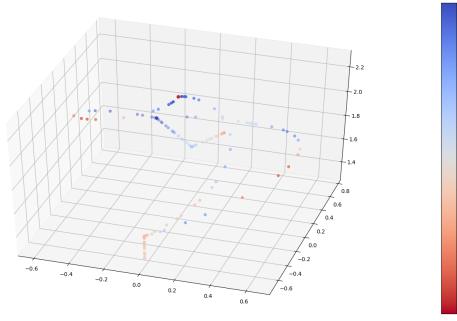


Figure 20: Plot the manipulability of the robot arm at sample points of the executed trajectory

The equation 5.3.3 calculates a joint limit measure which is multiplied with the manipulability measure and gives the dexterity measure. From that equation we can conclude the following:

- If $q_i = q_{\min}$ or $q_i = q_{\max}$ then the exponential is equal to 1 which means that \mathcal{L}_q and \mathcal{D} are both 0, which means that the robot has **no dexterity at the boundary of the task space**.
- If the value of q_i is close to it's boundary value then the dexterity approaches 0. The how much close or far it is from the boundary (or in other words how fast the exponential term converges) depends on the parameter κ
- The q_{\min}, q_{\max} are calculated from the geometry of the task-space

For maximum dexterity at most points of a trajectory in a pivoting motion, the pivot sub- taskspace (i.e. the space of all configurations of feasible pivot motions) must be fully within the robot's whole reachable taskspace, otherwise only a small range of pivot movements will be feasible.

Finding $q_{i,\min}, q_{i,\max}$ at 5.3.3 is very difficult and time-consuming especially at task spaces with more intricate geometries. A similar and more practical equation to 5.3.3 can be written for calculating the dexterity of the robot in task space:

$$\mathcal{L}_p = 1 - e^{-\kappa(r_{\max} - r)} \quad (5.3.4)$$

where r_{\max} is the maximum radius of a circle that the tool tip can follow at a given insertion depth z . Moreover, at every point of the taskspace, it is $L^2 = r^2 + z^2$. The equation 5.3.4 however only shows

dexterity in terms of approaching the boundary of the taskspace and it does not take into consideration internal points of low dexterity and singularities like equation 5.3.1.

```

1 x = [] ; y = [] ; z = [] ;
2 r = [] ;
3 s = 0:0.005:0.5;
4 L = 0.48;
5 k = 4.5;
6
7 for z0=-L:0.01:0
8 if abs(z0)*sqrt(2) <= L
9     rmax = abs(z0);
10 else
11     rmax = sqrt(L^2-z0^2);
12 end
13 for r0=0:0.02:rmax
14     x = [x, r0*cos(2*pi*s)];
15     y = [y, r0*sin(2*pi*s)];
16     z = [z, z0*ones(size(s))];
17     lp = 1-exp(-k*(rmax-r0));
18     r = [r, lp*ones(size(s))];
19 end
20 end

```

Listing 1: RCM Taskspace calculation using MATLAB

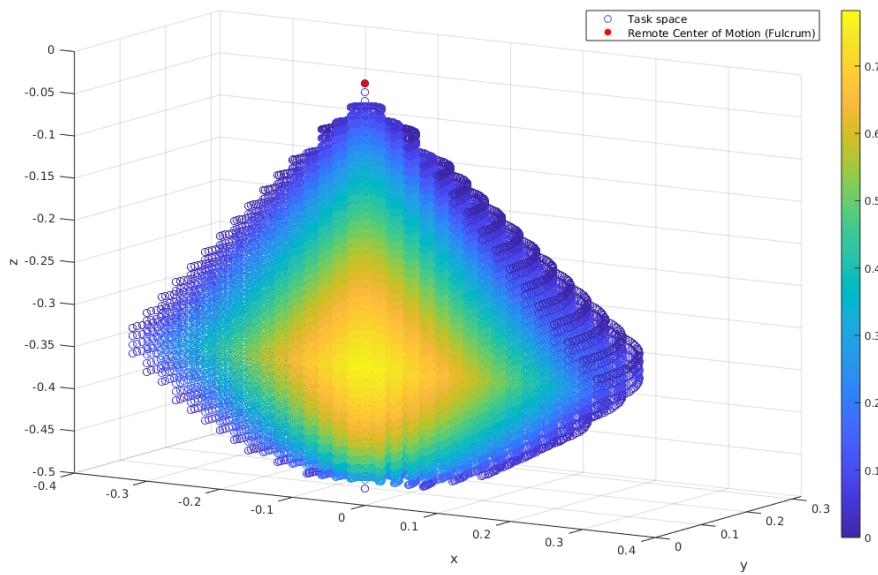


Figure 21: Task space inside patients body. Colors with 0 or low value correspond to points with low dexterity

6 Trajectory Planning - Laparoscopic tool manipulation

At this step, given the points of the desired path, a more detailed trajectory is calculated, which will contain all the waypoints that the robot will have to visit. Trajectory planning is executed after a desired path is generated, and consists in mapping the geometric points to specific **time points**, as well as assigning specific **velocities**, **accelerations** and **jerks**, in order to generate the commands needed for the robot controller to execute a smooth motion.

The paths that are calculated are parameterized by the path parameter s . As s increases from 0, the robot moves from the start configuration $q(0)$ to the goal configuration $q(1)$. Path planning outputs geometric information $q(s), s \in [0, 1]$, whereas the **trajectories** that are the subject of this chapter, also include **time information** $q(t), t \in [0, T]$. A path can be converted to a trajectory by defining a function $s(t) : [0, T] \rightarrow [0, 1]$ which maps the time parameter's range to the path parameter's range. This function is also known as **time scaling** or **time parameterization**. The most common methodology of trajectory planning, which is also used in this thesis, is the one that is studied in the **joint angles space** also known as **configuration space**.

The biggest challenge in manipulating a laparoscopic tool with a robot is overcoming the **fulcrum effect** problem. This is also one of the reasons that robotic assisted surgery replaced the traditional laparoscopic procedures. The fulcrum effect means that the surgeon's hand motions are inverted and scaled with respect to the Remote Center of Motion point, which lies approximately on the center of the incision. Apart from the scaling and inversion, laparoscopic procedures add an additional motion constraint that demands at each time one point of the laparoscopic tool to coincide with the RCM point.

6.1 Tool pose & the Fulcrum Effect

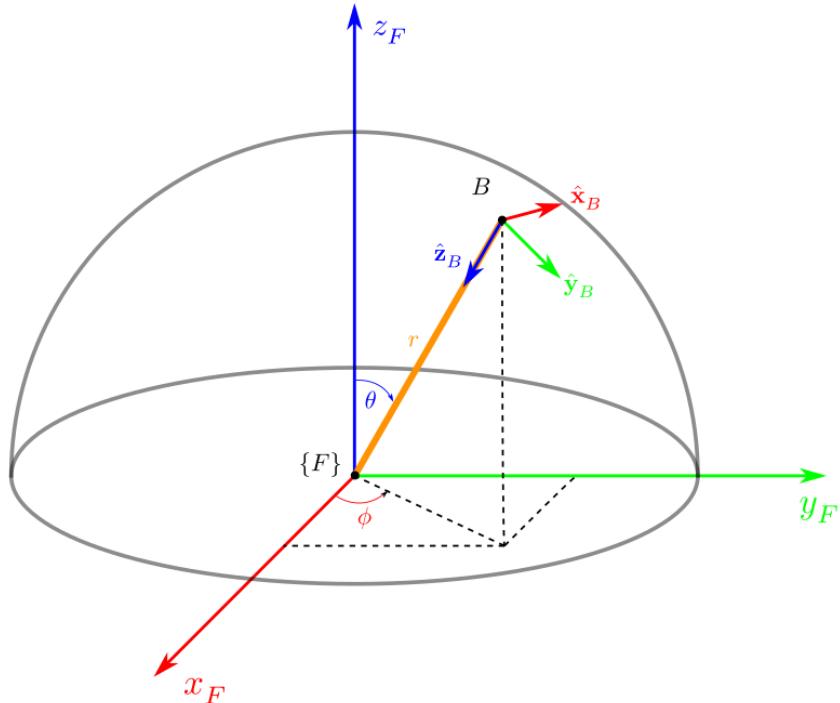


Figure 22: Tool pose at target point B calculated with respect to Fulcrum's reference frame $\{F\}$

The laparoscopic tool pose is given by the position and orientation vectors at target point B with

respect to the coordinate frame $\{F\}$. The pose is given by the following transformation matrix

$${}^F T_B = \begin{bmatrix} {}^F R_B & {}^F \mathbf{p}_B \\ \mathbf{0} & 1 \end{bmatrix} \quad \text{where } {}^F R_B = [\hat{\mathbf{x}}_B \quad \hat{\mathbf{y}}_B \quad \hat{\mathbf{z}}_B]$$

$$\hat{\mathbf{x}}_B = \hat{\theta} = \cos(\theta)\cos(\varphi)\hat{\mathbf{x}}_F + \cos(\theta)\sin(\varphi)\hat{\mathbf{y}}_F - \sin(\theta)\hat{\mathbf{z}}_F = \begin{bmatrix} \cos(\theta)\cos(\varphi) \\ \cos(\theta)\sin(\varphi) \\ -\sin(\theta) \end{bmatrix} \quad (6.1.1)$$

$$\hat{\mathbf{y}}_B = \hat{\varphi} = -\sin(\varphi)\hat{\mathbf{x}}_F + \cos(\varphi)\hat{\mathbf{y}}_F = \begin{bmatrix} -\sin(\varphi) \\ \cos(\varphi) \\ 0 \end{bmatrix} \quad (6.1.2)$$

$$\hat{\mathbf{z}}_B = -\hat{\mathbf{r}} = -(\sin(\theta)\cos(\varphi)\hat{\mathbf{x}}_F + \sin(\theta)\sin(\varphi)\hat{\mathbf{y}}_F + \cos(\theta)\hat{\mathbf{z}}_F) = \begin{bmatrix} -\sin(\theta)\cos(\varphi) \\ -\sin(\theta)\sin(\varphi) \\ -\cos(\theta) \end{bmatrix} \quad (6.1.3)$$

The position of the point B is given in spherical coordinates by:

- $r = \rho$: outside penetration of laparoscopic tool
- $\theta = \beta$: altitude angle
- $\varphi = \alpha$: orientation angle

thus the position with respect to the coordinate frame $\{F\}$ is given by

$${}^F \mathbf{p}_B = \begin{bmatrix} \rho \sin(\beta) \cos(\alpha) \\ \rho \sin(\beta) \sin(\alpha) \\ \rho \cos(\beta) \end{bmatrix} = \rho \hat{\mathbf{r}} \quad (6.1.4)$$

The above goal point must be the same as the TCP point of the robot's end-effector. This means, that this pose must be converted with respect to the robot's reference frames.

$${}^U T_{TCP} = {}^U T_B$$

$${}^U T_0 {}^0 T_7 {}^7 T_{TCP} = {}^U T_F {}^F T_B$$

$${}^0 T_7 = {}^U T_0^{-1} {}^U T_F {}^F T_B {}^7 T_{TCP}^{-1} \quad (6.1.5)$$

```

1 function tcp = fulcrumEffectTrajectory(P, L)
2     tcp = zeros(size(P));
3     for i=1:size(P)
4         px = P(i,1); py = P(i,2); pz = P(i,3);
5         r = sqrt(px^2+py^2+pz^2);
6         th = atan2(sqrt(px^2+py^2), pz);
7         phi = atan2(py, px);
8         vx = [cos(th)*cos(phi); cos(th)*sin(phi); -sin(th)];
9         vy = [-sin(phi); cos(phi); 0];
10        vz = [-sin(th)*cos(phi); -sin(th)*sin(phi); cos(th)];
11        vp = r*[sin(th)*cos(phi); sin(th)*sin(phi); cos(th)];
12        T = zeros(4,4);
13        R = [vx, vy, vz];
14        T(1:3,1:3) = R;
15        T(1:3,4) = vp;
16        T(4,4) = 1;
17        Td = eye(4);
18        Td(1:3,4) = (r-L)/r*vp;
19        tcp_point = Td*inv(T)*[P(i,:).'; 1];
20        tcp(i,:) = tcp_point(1:3).';
21    end
22 end

```

Listing 2: Fulcrum Effect transformation of a trajectory, in MATLAB

6.2 Trajectory planning in cartesian coordinates

On this section, some basic pivoting trajectories around the fulcrum point, are presented. In all of the following three example pivoting motions, we have made the assumption that the position and orientation of the F reference frame is precisely known, which is however not applicable in real-life scenarios. The trajectories presented in this section are planned in cartesian coordinates or also known as **Task space**.

Advantages of planning in Task Space

- Since the interpolation of the path points is in task space, then the planned motion is **more predictable**.
- It is easier to design trajectories which are **better in avoiding obstacles and handling collisions**.

Disadvantages of planning in Task Space

- **Planning and execution are significantly slower**, because for each interpolation point in task space the Inverse Kinematics problem must be solved. This issue is even more problematic when the IK problem is not solved with analytical equations but numerically using optimization techniques.
- A smooth trajectory in task space is not necessarily smooth in Joint Space.

6.2.1 Circular trajectory of tool tip

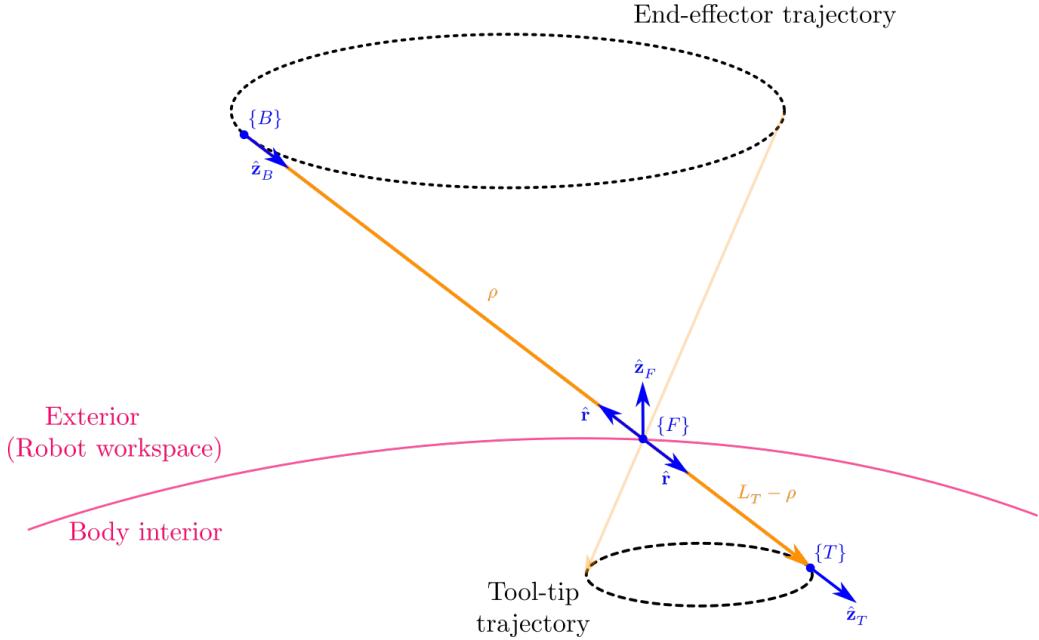


Figure 23: Circular trajectory of tool tip with respect to Fulcrum reference frame

To generate a circular trajectory for the pivot movement we must specify the center of the circle and a vector whose magnitude is the radius of the circle and it's direction gives the orientation of the plane that the circle lies at. The simplest case of a circular trajectory is the one, whose circle lies in a plane parallel to the xy plane.

We first consider the motion of the laparoscopic tool tip on a circle parallel to a z-plane, with respect to the $\{F\}$ coordinate frame.

$$(x_F - x_{F0})^2 + (y_F - y_{F0})^2 = r_0^2, \quad z_F = z_{F0} \quad (6.2.1)$$

It's often more convenient to express trajectories in a parametric form, which makes it easier to calculate all the waypoints of the trajectory

$$\begin{cases} x_F = r_0 \cos(2\pi s) + x_{F0} \\ y_F = r_0 \sin(2\pi s) + y_{F0} \\ z_F = z_{F0} \end{cases}, \quad s \in [0, 1] \quad (6.2.2)$$

After having calculated the cartesian coordinates we can calculate the spherical coordinates as follows

$$\begin{cases} r = \sqrt{x_F^2 + y_F^2 + z_F^2} \\ \theta = \text{atan2}\left(\sqrt{x_F^2 + y_F^2}, z_F\right) \\ \varphi = \text{atan2}(y_F, x_F) \end{cases} \quad (6.2.3)$$

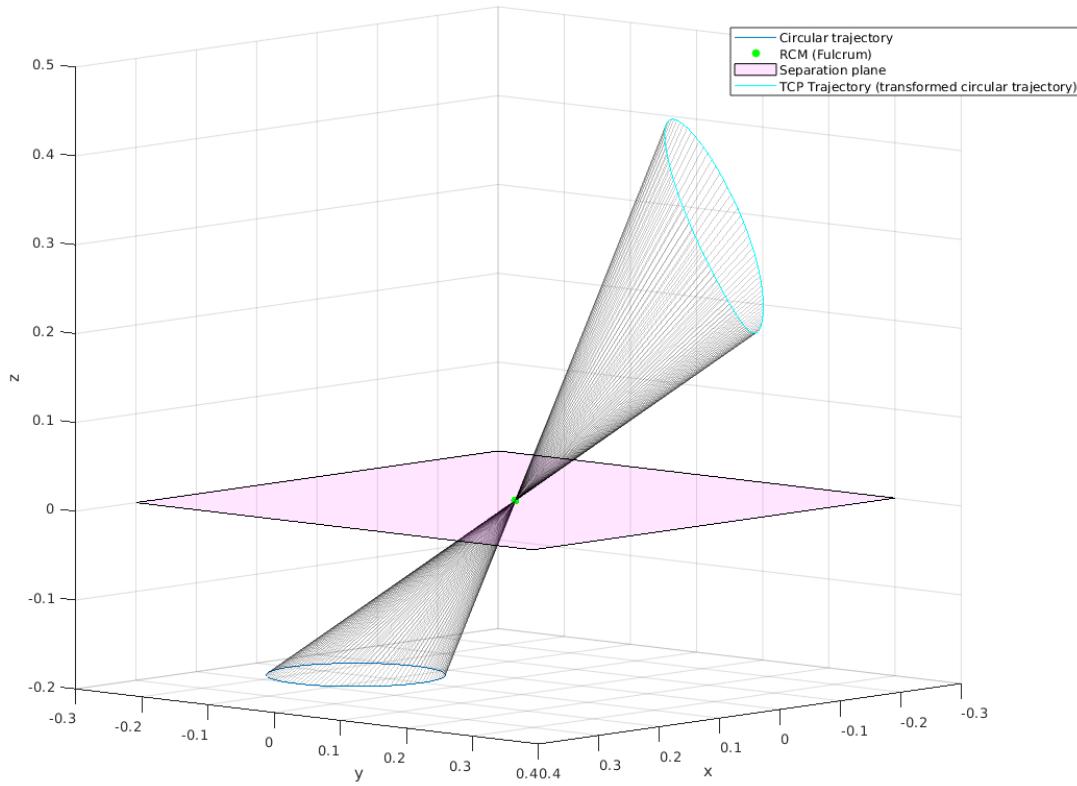


Figure 24: Circular trajectory of tool tip with respect to Fulcrum reference frame and it's transformation via the Fulcrum Effect

Although the equations 6.2.2 are very simple to calculate, it is more often the case that the circular trajectory will lie on an arbitrary plane that will not be parallel to the z-plane. This means that the equations 6.2.2 must first be transformed inside the task space (with the desired rotation and translation) and then these transformed equations should be used as an input to the Fulcrum Effect transformation.

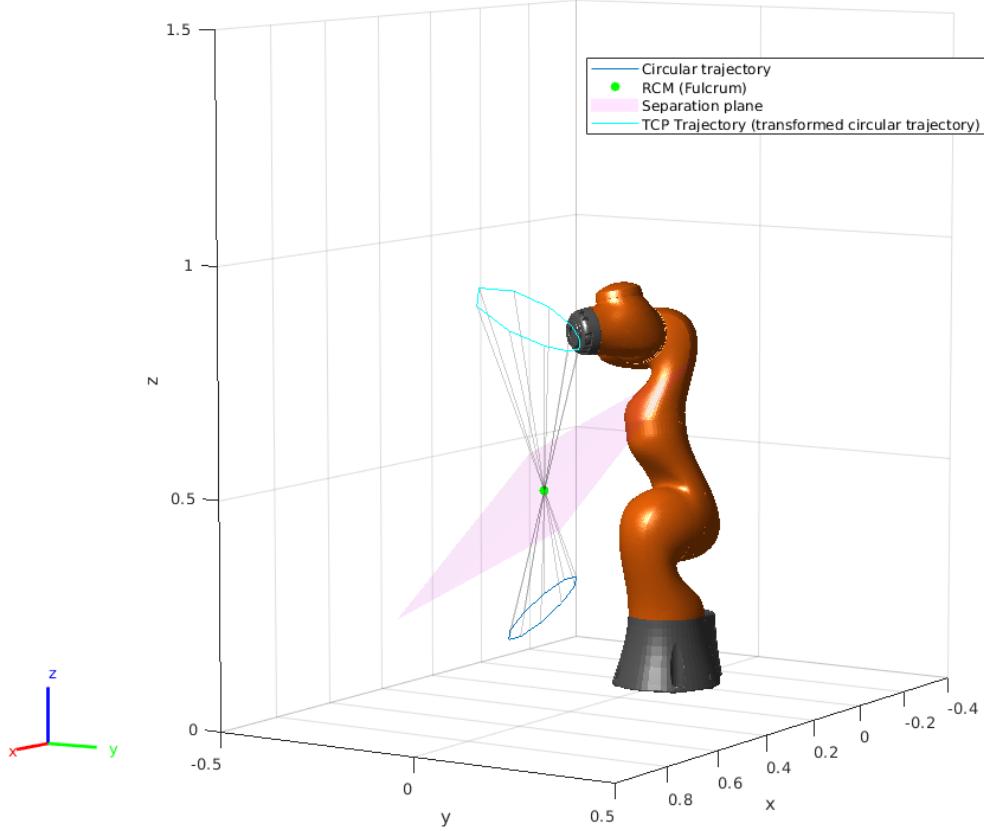


Figure 25: Circular trajectory that lies on an a plane of arbitrary orientation with respect to the fulcrum point

6.2.2 Circular arc trajectory of tool tip

To generate a circular arc trajectory for a pivot motion we must specify the same parameters as in the circular trajectory as well as the length of the arc or the total angle φ of the arc section. Another parameter has more significance in circular arc trajectories than plain circles is the phase of the arc. This initial phase can be expressed as either an angle φ_0 added to the arguments of sine and cosine functions or it can be expressed as s values that have as initial value different from 0. The parametric equations for a circular arc are:

$$\begin{cases} x_F = r_0 \cos(2\pi s + \varphi_0) + x_{F0} \\ y_F = r_0 \sin(2\pi s + \varphi_0) + y_{F0} \\ z_F = z_{F0} \end{cases}, \quad s \in \left[0, \frac{\varphi}{2\pi}\right] \quad (6.2.4)$$

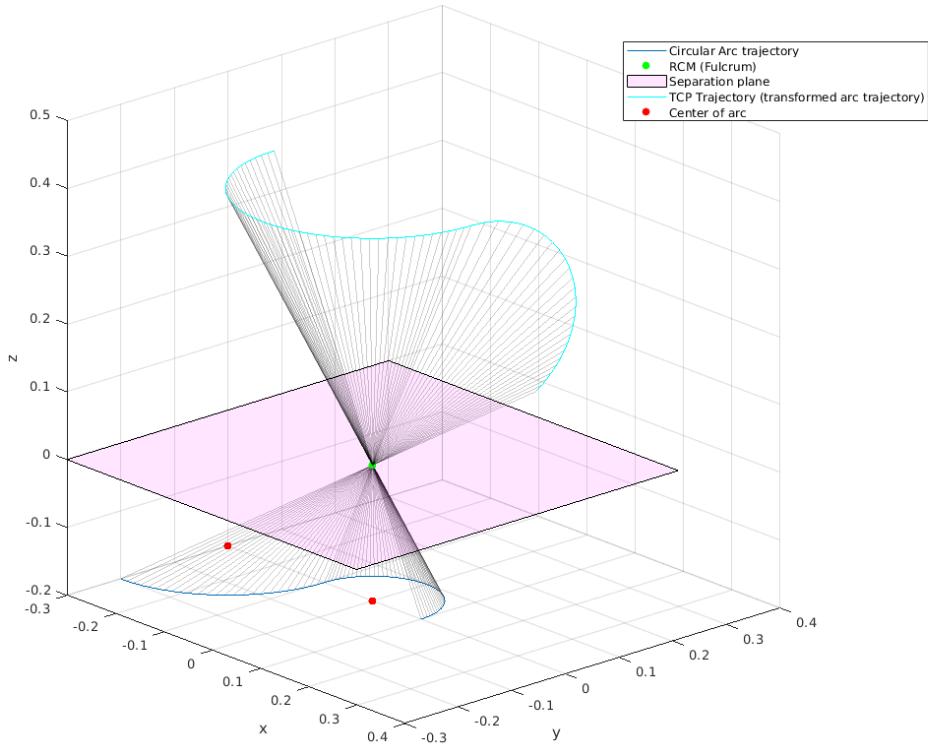


Figure 26: Circular arc trajectory of tool tip with respect to Fulcrum reference frame and it's transformation via the Fulcrum Effect. In this trajectory 2 circular arcs are used

6.2.3 Line segment trajectory of tool tip

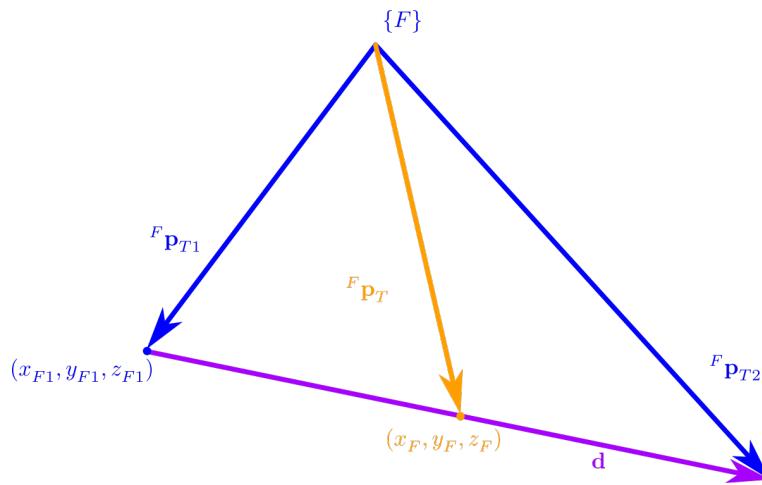


Figure 27: Line segment trajectory of tool tip with respect to Fulcrum reference frame

$$\begin{aligned}\mathbf{d} &= {}^F \mathbf{p}_{T2} - {}^F \mathbf{p}_{T1} = [l, m, n]^\top \\ {}^F \mathbf{p}_T &= [x_F, y_F, z_F]^\top \\ {}^F \mathbf{p}_T &= {}^F \mathbf{p}_{T1} + s\mathbf{d}\end{aligned}$$

$$s = \frac{x_F - x_{F1}}{l} = \frac{y_F - y_{F1}}{m} = \frac{z_F - z_{F1}}{n} \quad s \in [0, 1]$$

$$\begin{cases} x_F = sl + x_{F1} = (1-s)x_{F1} + sx_{F2} \\ y_F = sm + y_{F1} = (1-s)y_{F1} + sy_{F2} \\ z_F = sn + z_{F1} = (1-s)z_{F1} + sz_{F2} \end{cases} \quad (6.2.5)$$

After having calculated the cartesian coordinates we can calculate the spherical coordinates using the 6.2.3 equations.

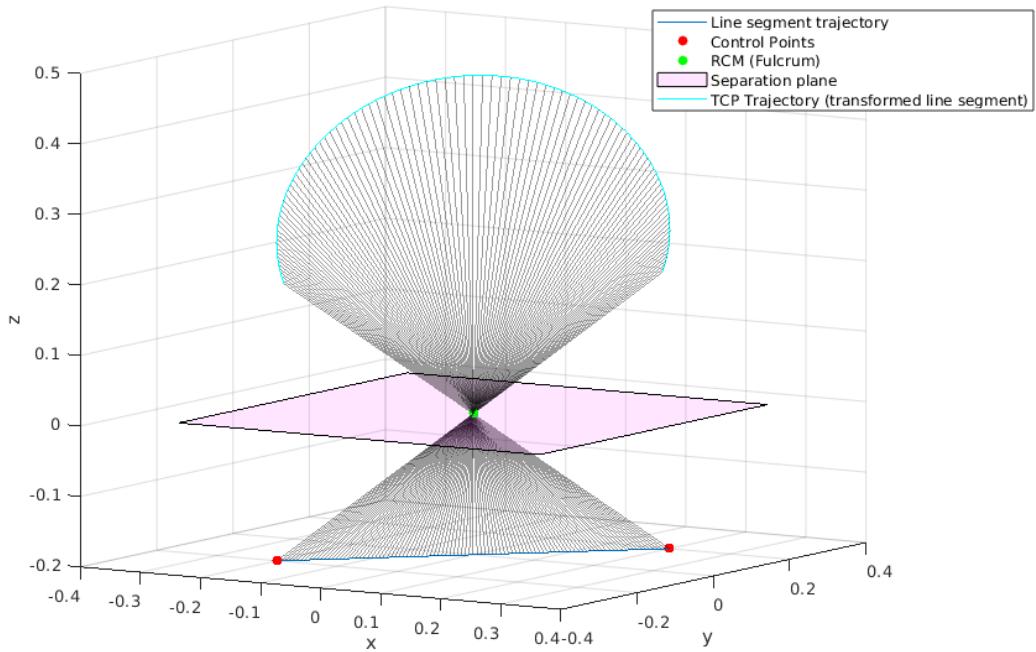


Figure 28: A Line segment trajectory and it's transformation via the Fulcrum Effect

The line segment trajectory of tool tip, as analysed above, **should not be confused with the** computeCartesianPath method provided by ROS MoveIt library that can be used to create line segment trajectories. This method produces line segment trajectories for the end effector of the robot which are not transformed as line segments via the Fulcrum Effect.

6.2.4 Cubic Spline trajectory of tool tip

A useful mathematical tool to construct a smooth curve that visits every point from a given set of waypoints are **cubic splines**. A cubic spline is constructed using smaller curves that are described by a polynomial of 3rd order. Let $\{\mathbf{P}_0, \mathbf{P}_1, \dots, \mathbf{P}_n\}$ be a set of waypoints, where each point has coordinates $\mathbf{P}_i = [x_i, y_i, z_i]^\top$. Then between each 2 points a cubic polynomial can be constructed (one for each coordinate, 3 in total). The following equations are for the x -coordinate and in the exact same way one can calculate the cubic polynomials for the y, z coordinates as well. For each pair of waypoints we want to calculate the following cubic polynomial

$$x_i(s) = a_i(s - s_i)^3 + b_i(s - s_i)^2 + c_i(s - s_i) + d_i, \quad s_i \leq s \leq s_{i+1} \quad (6.2.6)$$

The polynomial in equation 6.2.6 has four unknowns which means that four additional equations are needed to get a unique solution and fully define the polynomial. These equations can be formed using

the boundary conditions for the first and last point of each curve.

$$x_i(s_i) = x_i \quad (6.2.7)$$

$$x_i(s_{i+1}) = x_{i+1} \quad (6.2.8)$$

$$\dot{x}_i(s_i) = \dot{x}_i \quad (6.2.9)$$

$$\dot{x}_i(s_{i+1}) = \dot{x}_{i+1} \quad (6.2.10)$$

First we solve for c_i and d_i , which can easily be calculated as follows

$$d_i = x_i(s_i) = x_i \quad (6.2.11)$$

and by taking the derivative of 6.2.6, we can calculate c_i

$$\dot{x}_i(s_i) = 3a_i(s - s_i)^2 + 2b_i(s - s_i) + c_i \quad (6.2.12)$$

$$c_i = \dot{x}_i(s_i) = \dot{x}_i \quad (6.2.13)$$

By substituting $s = s_{i+1}$ in 6.2.6 and 6.2.12, by using equations 6.2.13, 6.2.11 and if we set $\sigma = s_{i+1} - s_i$ for brevity, we get the following two equations

$$x_{i+1} = x_i(s_{i+1}) = a_i\sigma^3 + b_i\sigma^2 + c_i\sigma + x_i \quad (6.2.14)$$

and

$$\dot{x}_{i+1} = \dot{x}_i(s_{i+1}) = 3a_i\sigma^2 + 2b_i\sigma + \dot{x}_i \quad (6.2.15)$$

By multiplying 6.2.15 by σ and 6.2.14 by -3 and add them together we get

$$\begin{aligned} \dot{x}_{i+1}\sigma - 3x_{i+1} &= -b_i\sigma^2 - 2\dot{x}_i\sigma - 3x_i \\ b_i &= \frac{1}{\sigma^2}(3x_{i+1} - 3x_i - \dot{x}_{i+1}\sigma - 2\dot{x}_i\sigma) \end{aligned} \quad (6.2.16)$$

Similarly, by multiplying 6.2.15 by σ and 6.2.14 by -2 and add them together we get

$$\begin{aligned} \dot{x}_{i+1}\sigma - 2x_{i+1} &= a_i\sigma^3 - \dot{x}_i\sigma - 2x_i \\ a_i &= \frac{1}{\sigma^3}(\dot{x}_{i+1}\sigma - 2x_{i+1} + \dot{x}_i\sigma + 2x_i) \end{aligned} \quad (6.2.17)$$

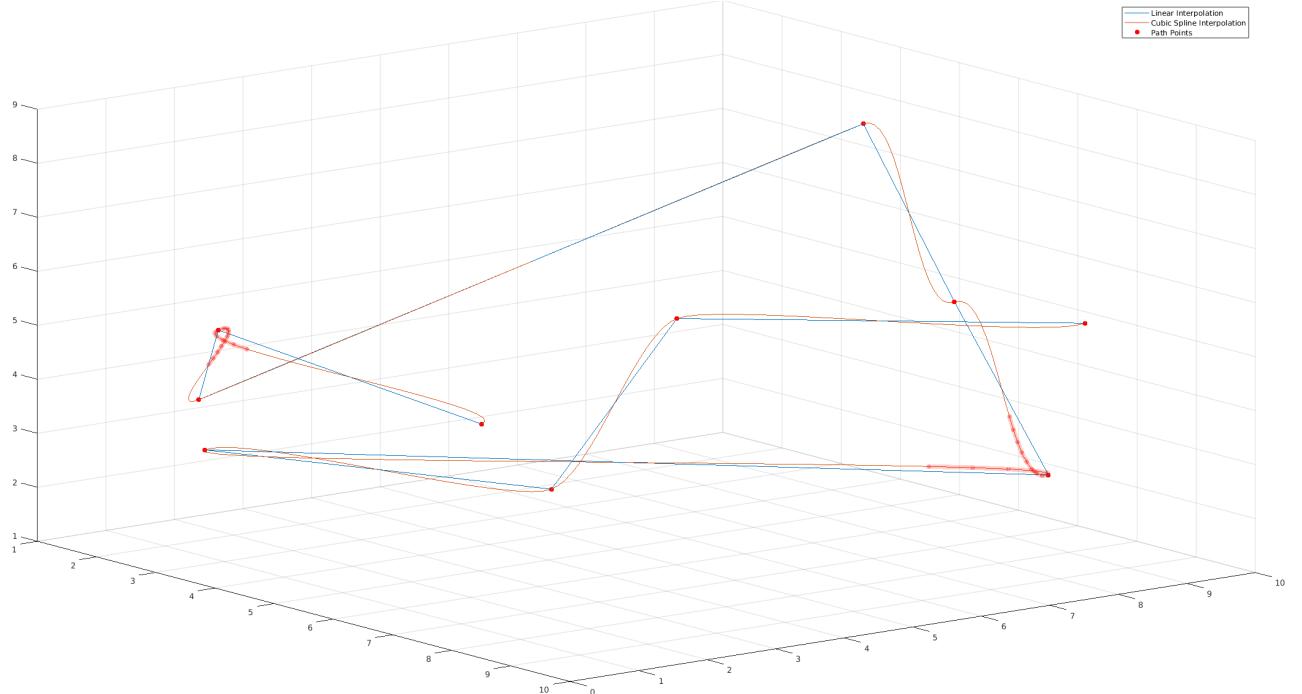


Figure 29: Cubic Spline curve with 10 waypoints

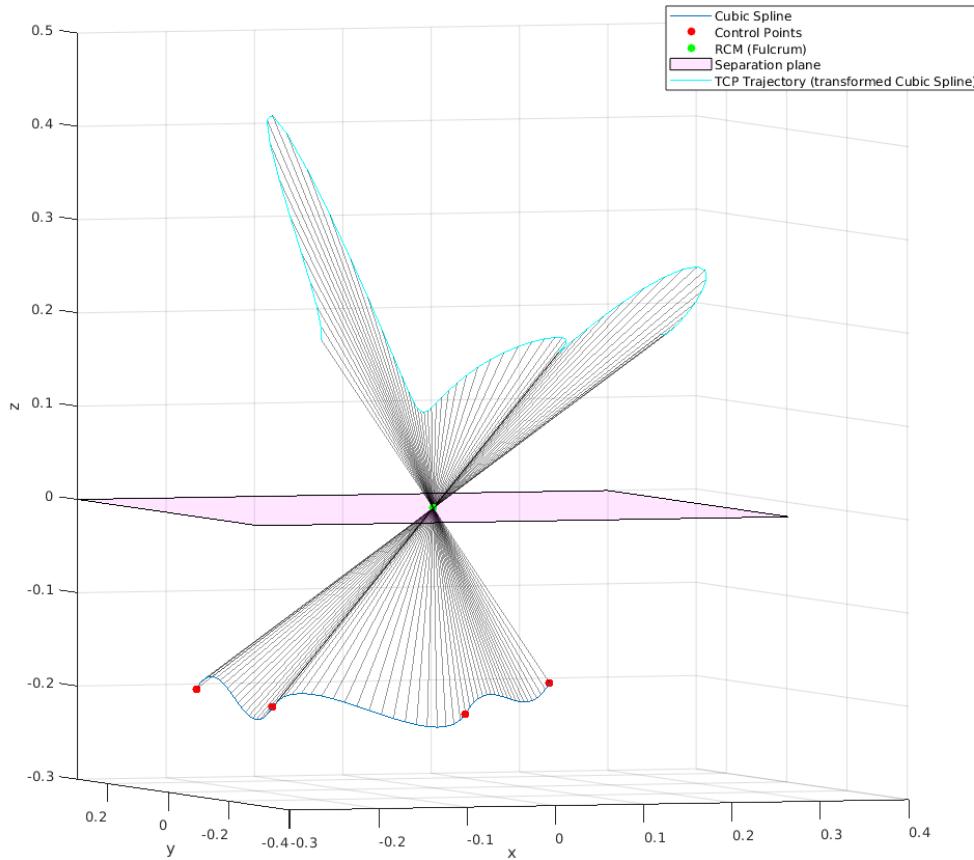


Figure 30: A Cubic Spline trajectory and it's transformation via the Fulcrum Effect

6.2.5 B-Spline trajectory of tool tip

The **B-Splines** are smooth curves which are constructed from **Bézier** curves. A Bézier curve is a parametric smooth curve and is a k -th order interpolation of $k + 1$ control points.

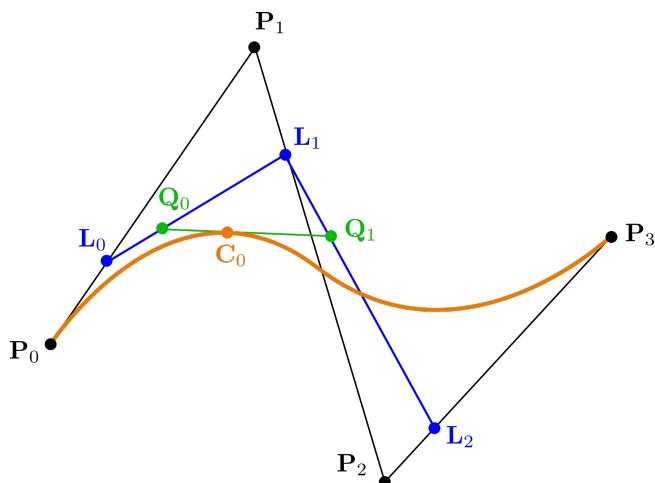


Figure 31: Cubic Bézier curve calculated using cubic interpolation of 4 control points

We first calculate the linear interpolation of the control points

$$\mathbf{L}_0(s) = (1 - s)\mathbf{P}_0 + s\mathbf{P}_1 \quad (6.2.18)$$

$$\mathbf{L}_1(s) = (1 - s)\mathbf{P}_1 + s\mathbf{P}_2$$

$$\mathbf{L}_2(s) = (1 - s)\mathbf{P}_2 + s\mathbf{P}_3$$

The next step is to calculate the quadratic interpolation of the control points or equivalently, the linear interpolation of the previously calculated points $\mathbf{L}_0, \mathbf{L}_1, \mathbf{L}_2$

$$\mathbf{Q}_0(s) = (1 - s)\mathbf{L}_0(s) + s\mathbf{L}_1(s)$$

$$\mathbf{Q}_0(s) = (1 - s)^2\mathbf{P}_0 + 2(1 - s)s\mathbf{P}_1 + s^2\mathbf{P}_2 \quad (6.2.19)$$

$$\mathbf{Q}_1(s) = (1 - s)^2\mathbf{P}_1 + 2(1 - s)s\mathbf{P}_2 + s^2\mathbf{P}_3$$

Similarly for the last step, we calculate the cubic interpolation of the control points or equivalently, the linear interpolation of the previously calculated points $\mathbf{Q}_0, \mathbf{Q}_1$

$$\mathbf{C}_0(s) = (1 - s)\mathbf{Q}_0(s) + s\mathbf{Q}_1(s)$$

$$\mathbf{C}_0(s) = (1 - s)^3\mathbf{P}_0 + 3(1 - s)^2s\mathbf{P}_1 + 3(1 - s)s^2\mathbf{P}_2 + s^3\mathbf{P}_3 \quad (6.2.20)$$

The cubic Bézier curve can also be calculated using the following more compact equation, in matrix form

$$\mathbf{C}_0(s) = [\mathbf{P}_0 \quad \mathbf{P}_1 \quad \mathbf{P}_2 \quad \mathbf{P}_3] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} s^3 \\ s^2 \\ s \\ 1 \end{bmatrix} \quad (6.2.21)$$

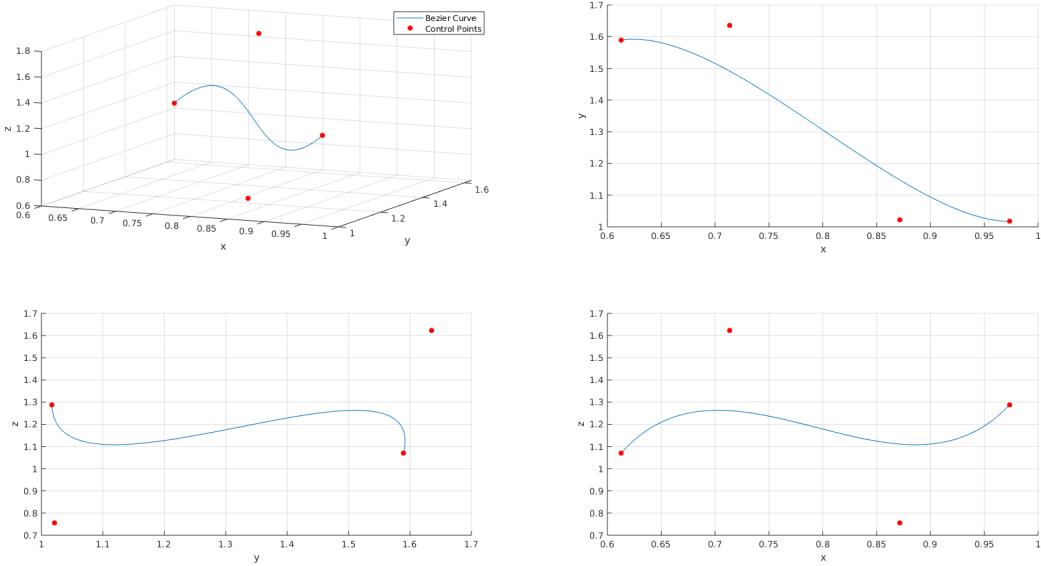


Figure 32: A cubic Bézier curve calculated and plotted in MATLAB

A k -degree **B-Spline** curve defined by $n + 1$ control points will consist of $n - k + 1$ Bézier curves. For example if we want to construct a cubic B-Spline using 6 control points, then we will need to construct and connect together 3 Bézier curves.

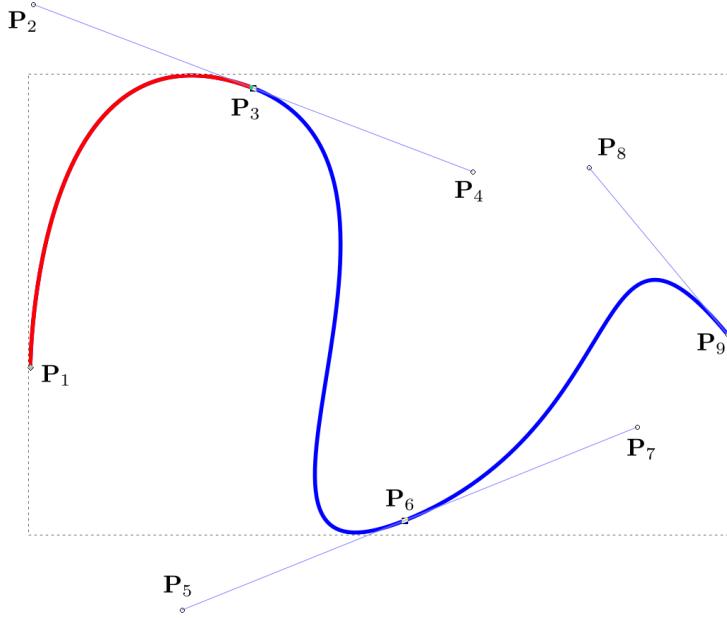


Figure 33: B-Spline curve constructed from 3 Bézier curves. The first Bézier curve colored in red is a quadratic one and the following two are both cubic.

In most cases the B-Spline curves are constructed by starting from a quadratic Bézier curve, which is constructed from 3 control points and then all other parts of the curve are constructed from cubic Bézier curves, each constructed from 4 control points. As shown in figure 33, the B-Spline curves do not pass from all control points. This means that if we have a path formed by a set of waypoints and we want the robot to pass from all of them, then in order to construct a B-Spline trajectory we will need additional intermediary points. The first and last points are part of the curve and the other control points are not. If no additional points can be calculated then the robot will not pass from all the points, which in some cases is also acceptable and useful.

One must notice also, in figure 33 for example, that some points must be colinear in order for the transition from one Bézier curve to another to be smooth, or equivalently the point where the two curves are connected to have a continuous derivative. For example in figure 33, $\mathbf{P}_2, \mathbf{P}_3, \mathbf{P}_4$ must be colinear so that the derivative of the curve at \mathbf{P}_3 is continuous. In the same example one can also notice that if the distance of $\overrightarrow{\mathbf{P}_2\mathbf{P}_3}$ and $\overrightarrow{\mathbf{P}_3\mathbf{P}_4}$ is getting smaller then the curve at \mathbf{P}_3 is getting sharper and if these distances are 0 then instead of a smooth curve we have a sharp corner and the curve has no longer a continuous derivative at \mathbf{P}_3 . These two parameters; **3 colinear points** and the **distances of the two vectors** of these 3 colinear points can be very useful in designing B-Spline trajectories when we only have the waypoints available and not the extra control points required to construct the Bézier curves.

It is very important that the designed trajectory respects the joints angles' range. For example depending on the starting position of the circular trajectory depicted at figure ??, the robot arm may reach its joint bounds and in order to continue executing the trajectory it will have to make a sudden jump to reset the angles. This could have serious side-effects for both the surgical task and thus the patient, as well as for the operating staff, who control the robot.

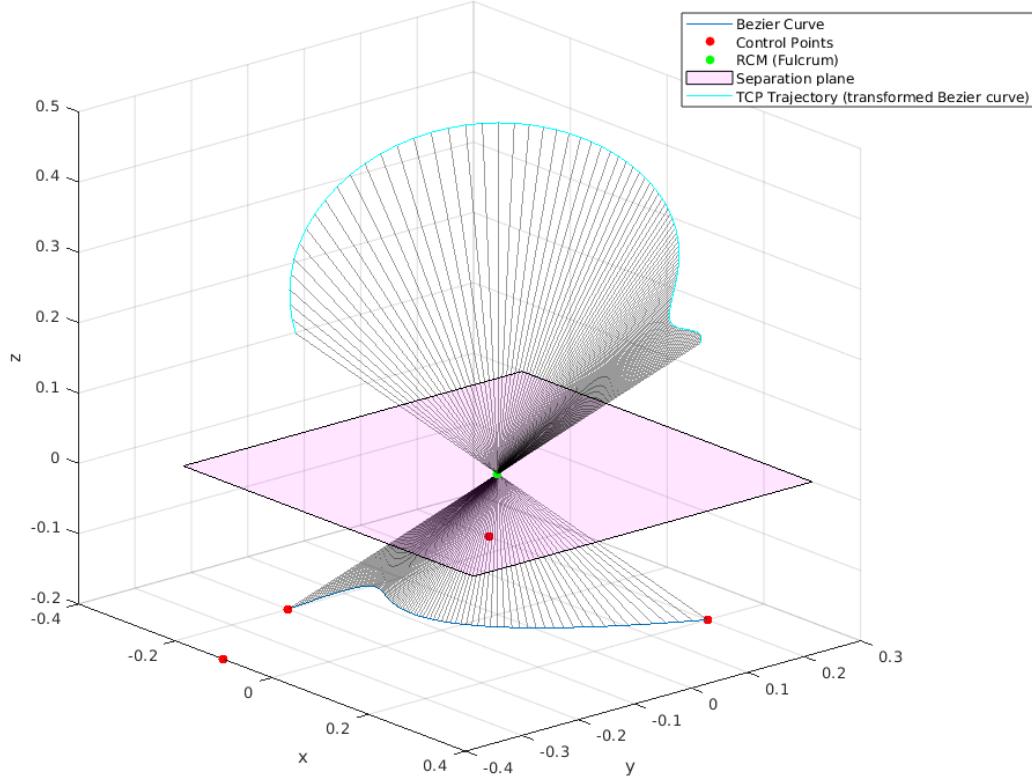


Figure 34: A Bézier curve trajectory and it's transformation via the Fulcrum Effect

6.3 Trajectory planning in joint angles space

Advantages of planning in Joint Space

- Planning trajectories in Joint Space is **much faster in execution** than Task Space planning, because the Inverse Kinematics problem is solved for fewer points, only at the waypoints of the trajectory and not at the intermediate points (the points between the waypoints).
- The actuators' motion is smoother

Disadvantages of planning in Joint Space

- The interpolated intermediate points are not guaranteed to be collision-free.

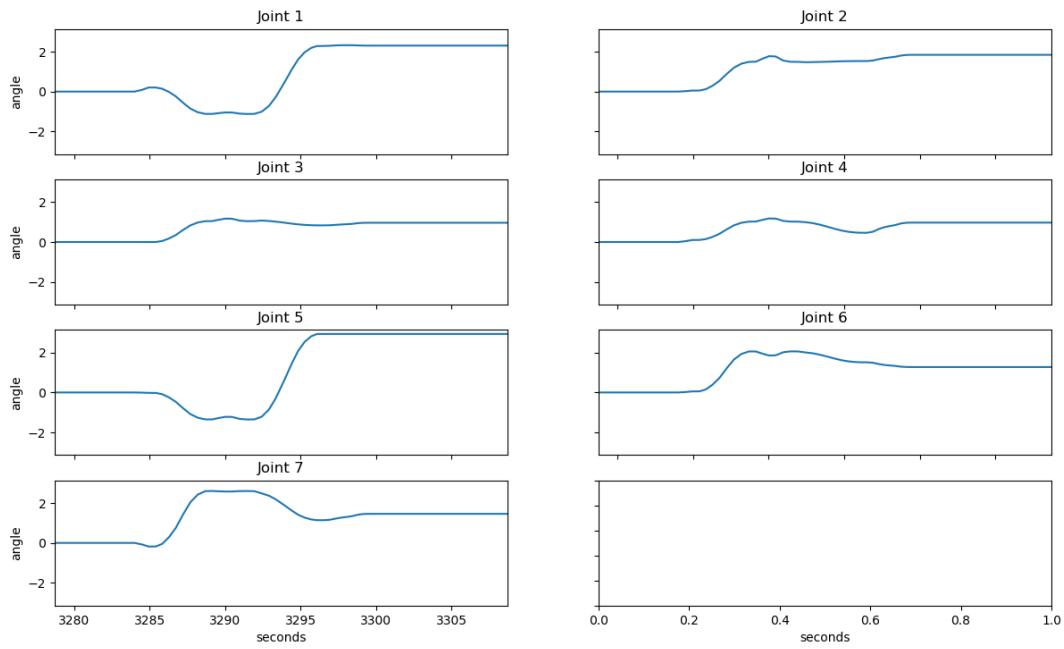


Figure 35: Trajectory diagrams in joints space.

6.3.1 Polynomials of 5th order

6.3.2 Planning with velocity profiles

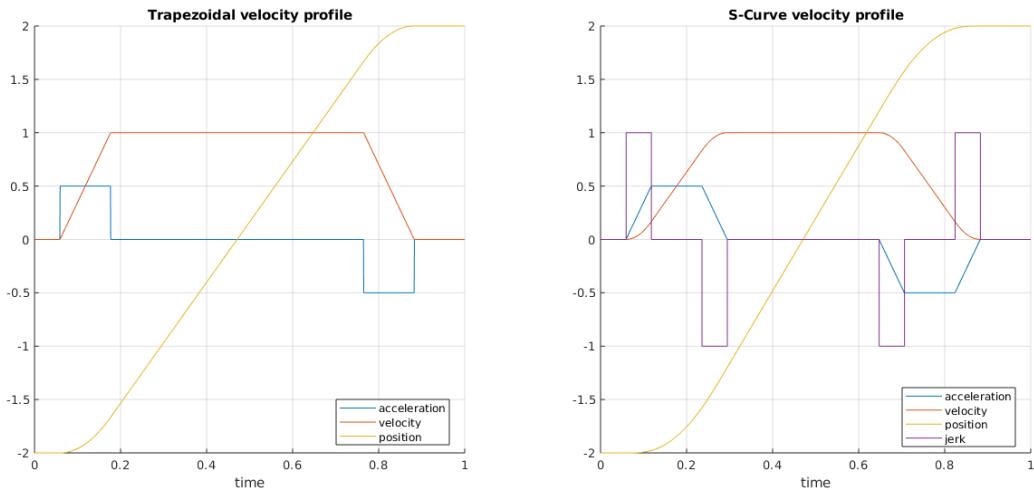


Figure 36: Joint trajectory generated using velocity profiles. Calculated and plotted in MATLAB

7 System Control

7.1 Firm grasping algorithm & Force control

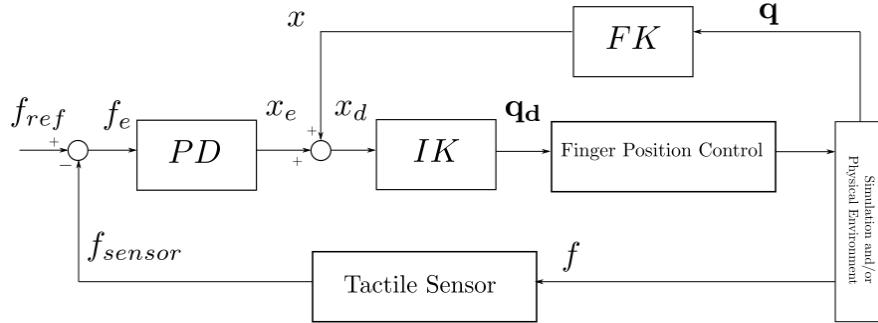


Figure 37: Force control on a Barrett Hand gripper finger

7.2 Visual Servoing

At this chapter we briefly investigate how visual servoing can be applied in surgery robotics. **Visual Servoing** is the use of visual information to guide and control a robot. The main task of visual servoing is to control the end-effector's pose using features extracted from visual information. The features that are usually extracted from cameras are the position and orientation of the detected object, the distance of the object from the camera (using stereoscopic vision, photogrammetry or other techniques), the size and the shape of the object. The visual servoing can be executed either in the robot's space using position-based servoing or in the camera's space (also known as "pixel space") by using the image-based technique.

7.2.1 Position based servoing

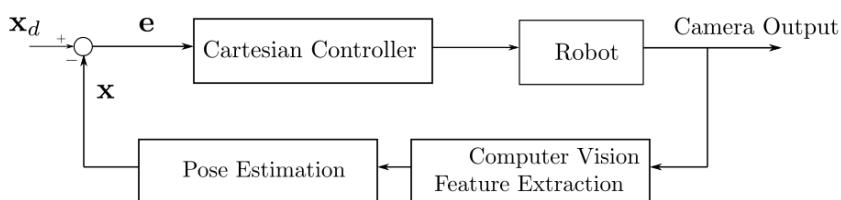


Figure 38: Position based visual servoing closed loop control

- **Photogrammetric technique**
- **Stereoscopic vision:** This methodology uses two separate views of the scene as taken from two viewports (from two cameras) and calculates the depth of various objects and areas using information from both views. For more details about stereoscopic vision see chapter 4.2
- **Extracting depth from motion** This methodology is very similar to stereoscopic vision and is also known as *monocular* or *motion stereo*. The difference is that instead of using two views from two distinct cameras this methodology uses two views from the same camera but from different points in time. A very important assumption for this methodology is to assume that two consecutive views from the video frame do not change significantly, so that some feature points can be matched in both views in order to calculate the depth information. This methodology is cheaper in terms of hardware but it fails to extract depth informations in cases where the robot is completely still.
- **Servoing using 3D sensors**

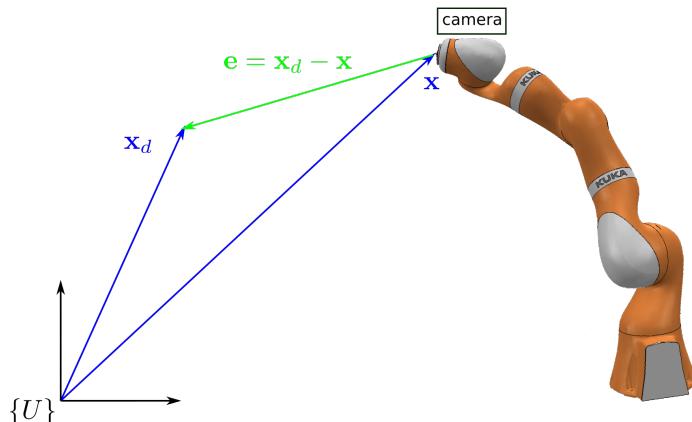


Figure 39: Position based visual servoing using depth from motion, stereo vision or 3D sensors, from which the desired position \mathbf{x}_d is calculated and used to drive the robot.

7.2.2 Image based servoing

Image based servoing is a methodology for controlling a robot by directly using features extracted from the image as well as positions on the image plane. The goal of this methodology is to drive the robot in such a way so that the video frame is changed from an initial view to a final, desired view (see figure 40 left and right frames). The commands that are sent to the robot from this methodology are defined in image space and not in the robot's task space. Moreover the distances calculated in image space are not directly related to distances in task space.



Figure 40: Image based visual servoing. The robot arm is controlled using the information gained from the video frames. The frames are 2Dimensional and thus the detected objects can have only 3 degrees of freedom which means we can mainly control 3 independent variables, here the x, y, θ variables. The left image is the initial frame and the right image is the frame where the object is at the target pose.

The image based visual servoing control system as depicted in figure 41 consists of the **Image Controller**, the **Plant** (robot) and the **feedback term**. The Image controller is a simple **PD Controller** (PID can also be used, but in servo systems PD is more common) which outputs commands to be executed in the plant. These commands are not to be confused with the robot's internal controller commands. These commands are to be used to control the robot in task space, whereas the internal controller drives each joint to the desired angle. The feedback used to calculate the error for the controller, uses the camera's output and based on that it calculates the vector from the detected tool's center of mass to the center of the image frame (feature extraction).

$$\mathbf{x}[(k+1)T] = \mathbf{x}[kT] + \mathbf{u}[kT] \quad (7.2.1)$$

where $\mathbf{x}[kT] = [x, y, z, \theta, \varphi, \psi]^T$ and the discrete PID control law is given by equation 7.2.2

$$\mathbf{u}[kT] = K_p \left[\mathbf{e}[kT] + \frac{T}{T_i} \sum_{i=0}^{k-1} \mathbf{e}[iT] + \frac{T_d}{T} (\mathbf{e}[kT] - \mathbf{e}[(k-1)T]) \right] \quad (7.2.2)$$

where $\mathbf{e}[kT] = [e_x, e_y, e_z, e_\theta, e_\varphi, e_\psi]^T$.

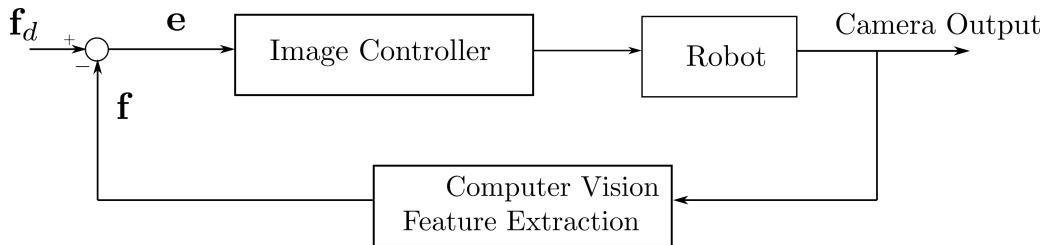


Figure 41: Image based visual servoing closed loop control

8 ROS framework

8.1 Introduction to the ROS framework

ROS is an open-source robotics software framework. It is a meta-operating system, which means that it provides its own abstractions on top of the host's operating system including filesystem, hardware abstractions, low-level device control, package management and networking. It also provides tools and services to develop large, scalable robotics software, it supports a wide variety of libraries and programming languages and it has a huge community, support and documentation resources.

1. ROS Filesystem

- Packages
- Metapackages
- Package Manifests
- Messages
- Services
- Launch files

2. ROS Computation Graph

- Nodes
- Master
- Parameter Server
- Topics
- Bags

3. ROS Community

- Distributions
- Repositories
- ROS Wiki
- ROS Answers

4. ROS Tools

- Gazebo
- RViz
- Moveit
- Filesystem tools

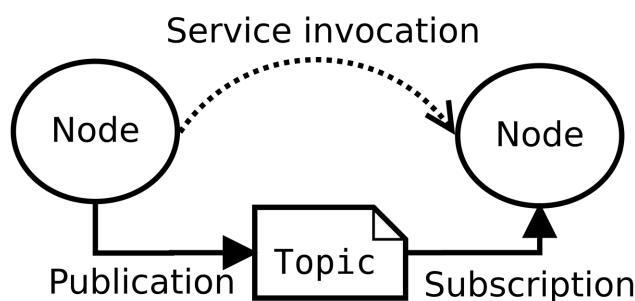


Figure 42: Communication diagram of 2 ROS nodes with a topic and a service

8.2 Gazebo simulation environment

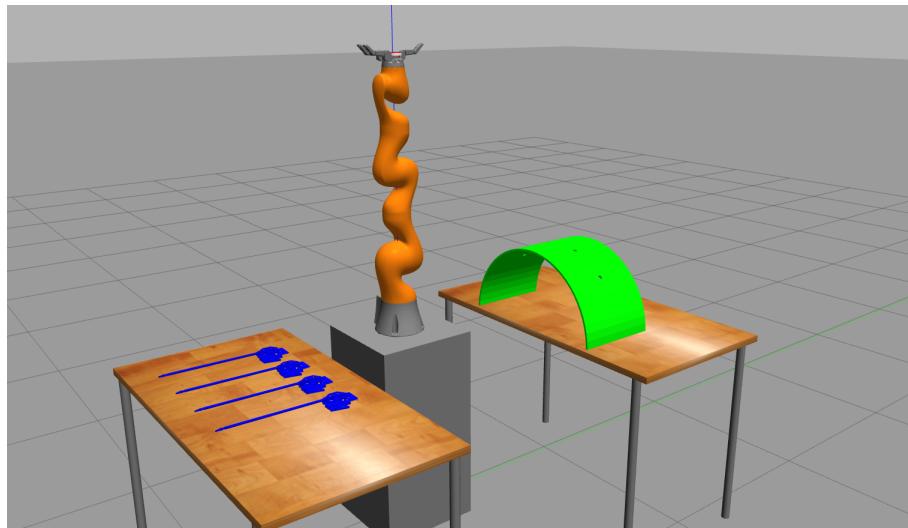


Figure 43: Simulation environment in Gazebo

The main environment setup of this thesis was designed using the Gazebo simulation environment and it consists of the following objects:

- the robot arm, KUKA®iiwa14 lbr, being at the center of the setup
- the robot base, so that the robot arm can better reach the tools and the surgical site and have more flexibility in movement
- 2 tables, one for the tools and one for the surgical site
- 4 surgical tools, using a modified version of the surgical tools used in the Raven II surgical platform
- a mounting dock, which has holes that have the same role as the trocars (small tubes from which the surgical tool is inserted). Initially a mounting dock with 4 same holes of 4mm diameter was used, but it was later replaced with a new one with holes of variable diameters to test feasibility of pivot motions. Larger diameters means more space for motion planner to search for solution and thus more probable to find a solution.

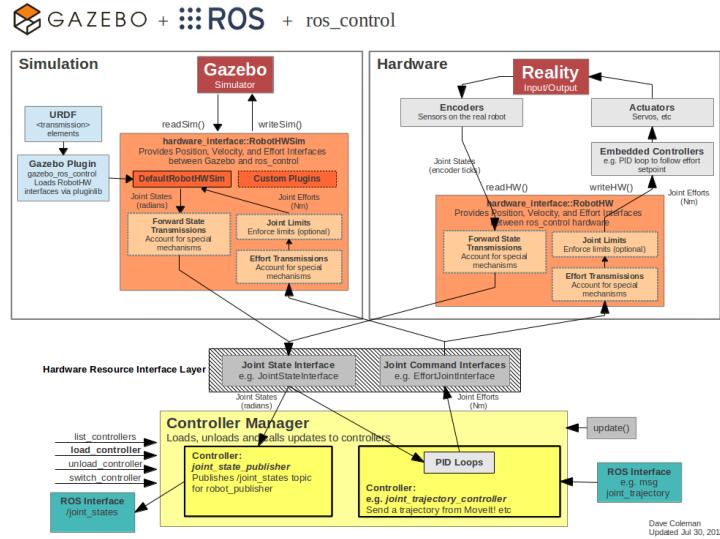


Figure 44: Control & Hardware Interfaces in Gazebo and ROS

8.3 Visualization with RViz

RViz is one of the most important and most used tools in robotic applications development and is a 3D visualizer for the Robot Operating System framework. RViz functionality should not be confused with that of Gazebo, because the first one visualizes the robot state and the **perceived** world (perceived objects or other calculations related to the world) whereas the second one simulates the real world.

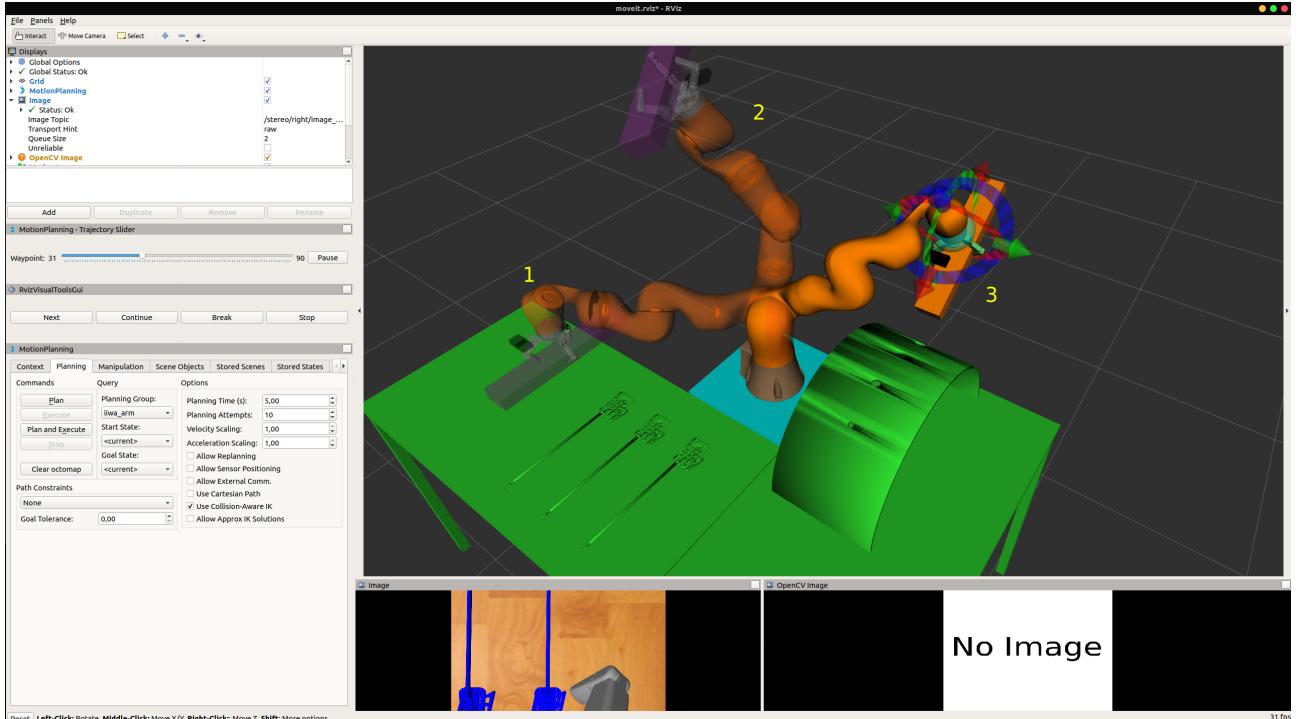


Figure 45: RViz: Visualizing the robot state as well as the state of the perceived world
In this screenshot, various poses of the robot are shown: 1) the current actual real pose of the robot, 2) the planned pose and 3) the goal pose, which can freely be moved within the RViz environment

The objects that appear in RViz can either be visualized from approximations calculated from actual measurements from the robot (for example a point cloud) or can be manually loaded, in which case we make an assumption that the robot already "knows" the exact position, orientation, size and shape of the object, which is rarely the case in real life scenarios. It is important to mention, that every such object is taken into consideration in collision checks and in path planning algorithms.

8.4 Motion Planning with Moveit

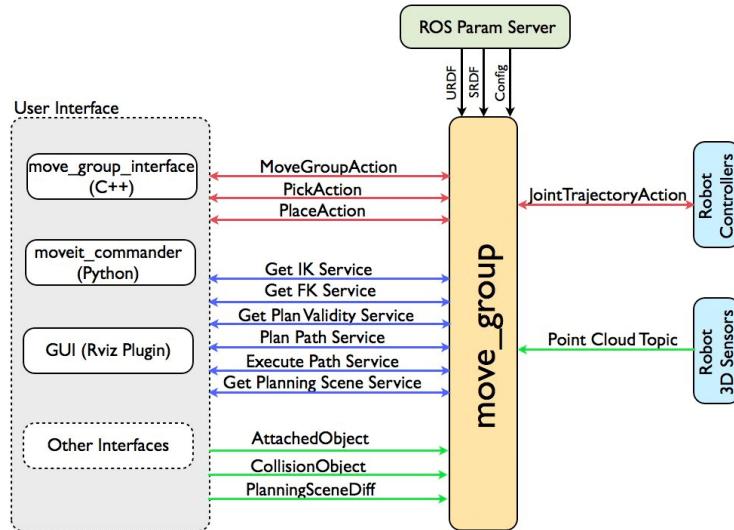


Figure 46: MoveIt ROS Architecture

When using the Moveit ROS library a set of parameters need to be configured in order to generate the desired commands for the controller:

- **Position tolerance:** The radius of the sphere that the end-effector must reach. This is the maximum allowed error in the target's position. This tolerance is typically much smaller inside the surgical site, to avoid fatalities.
- **Orientation tolerance:** The tolerance or maximum allowed error for roll, pitch and yaw, in radians
- **Maximum planning time:** Maximum amount of time to be used when planning a trajectory
- **Replanning:** Specify whether the robot is allowed to replan if it detects changes in the environment
- **Maximum planning attempts:** Number of times the motion plan is to be computed from scratch before the shortest solution is returned
- **Base frame:** The frame in respect to which the motion plan is calculated.
- **Jump threshold:** This parameter sets an upper bound to the amount of "jump" (change in distance) that can occur between two consecutive trajectory points in joint positions. Disabling the jump threshold while operating real hardware can cause large unpredictable motions of redundant joints and could be a safety issue
- **Velocity scaling factor:** The approaching motion needs to be slower. We reduce the speed of the robot arm via a scaling factor of the maximum speed of each joint. Note this is not the speed of the end effector point

- **End-effector step:** The resolution at which the Cartesian path is interpolated or the max step in Cartesian translation
- **Planner algorithm:**
- **Fraction:** The fraction of the path achieved as described by the waypoints

Typical motion planning parameter values outside of surgical site:

- Position tolerance: 50µm
- Orientation tolerance: 0.00005 deg
- Planning time: 10s

Typical motion planning parameter values inside surgical site:

- Position tolerance:
- Orientation tolerance:
- Planning time
- End-effector interpolation step: 1mm
- Maximum velocity scaling factor

Sometimes the motion planner finds a solution but the execution from the controller is aborted. After many iterations of the same experiment this does not happen always, which means that the feasibility of the execution of the movement by the controller depends on the initial state of the robot, i.e. if initially some joints of the robot are at their boundaries, then the next commanded trajectory maybe unfeasible. Another reason that the path planner may fail is the probabilistic nature of the path planning algorithms (see also RRT and PRM algorithms in chapter 6).

At each time step it is important to publish a custom message containing all the information about the kinematic state of the robot. In this thesis a custom **ROS** message was created containing a tf transform with a 3D vector for the position and a quaternion for the rotation and a custom 6-by-7 matrix containing the values of the Jacobian. The MoveIt library, from which the kinematic state of the robot is obtained, returns the orientation of the end effector as a 3-by-3 rotation matrix, but in the ROS tf message it must be expressed as a quaternion. To convert the matrix to a quaternion we first calculate the euler angles and then use these values to construct the quaternion “vector”. The quaternion representation of rotation is often preferred in robotic applications due to its efficiency in calculations and memory. To convert the transformation matrix to euler angles and then to quaternions the following formulas were used:

$$T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & x \\ r_{21} & r_{22} & r_{23} & y \\ r_{31} & r_{32} & r_{33} & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\varphi = \text{atan2}(r_{21}, r_{11})$$

$$\theta = \text{atan2}(-r_{31}, \sqrt{r_{11}^2 + r_{21}^2})$$

$$\psi = \text{atan2}(r_{32}, r_{33})$$

where T is the transformation matrix and φ, θ, ψ are the roll, pitch and yaw (Euler) angles.

8.5 Tools, Packages and Libraries

Stereo Vision with stereo image proc

State machines with Smach

9 Experiments and Results

9.1 Robot Planner 1: Simple MoveIt planning

In this first experiment we are testing some simple trajectories with the surgical tool already attached to the robot arm's end effector. The path is designed using the appropriate coordinates and orientations so that the robot begins from the home position, then visits the table with the surgical tools and then visits the other table on top of which the mounting dock is placed. Upon arrival at the mounting dock, the robot inserts the tool inside a hole (we consider these holes to be a simplistic alternative to the trocars used in real operations), then executes a simple pivot motion, while the tool is still inserted and then the tool gets ejected from the mounting dock's hole.

The aim of this experiment is to test the overall behaviour of the robot inside the work space, before implementing more complex path planning algorithms.

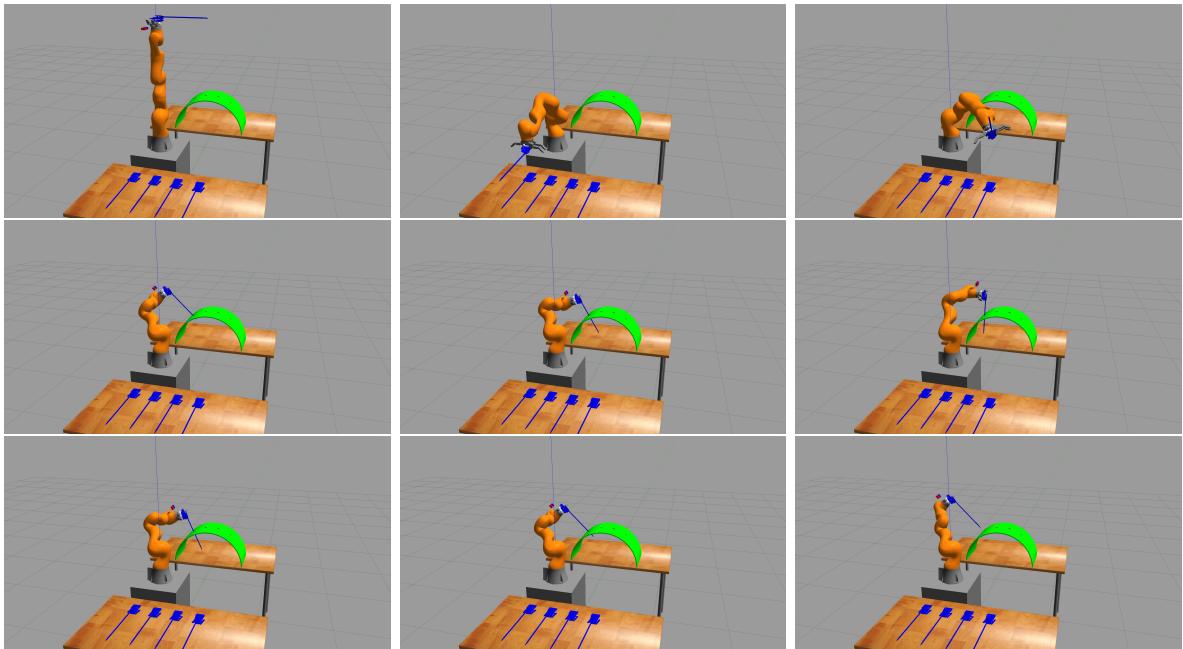


Figure 47: Experiment 1:

	RRTConnect							
Robot Planner 1	Fake Pick and Place				Pivot trajectory			
Experiment	Status	Solution Time	Path Simplification Time	Planning Attempts	Status	Solution Time	Path Simplification Time	Planning Attempts
1								
2								
3								
4								
5								
Average								

9.2 Robot Planner 2: Simulation layout and reachability experiments

In this experiment, we plan a path such that the robot arm will visit all holes of the mounting dock and will try the insertion movement of the surgical tool. This experiment is very useful, because it shows whether all holes of the mounting dock are **reachable** (inside the robot's work space) and if

so, how **dexterous** the robot will be in pivoting around each hole, i.e. how free the robot arm is to execute pivot motions.

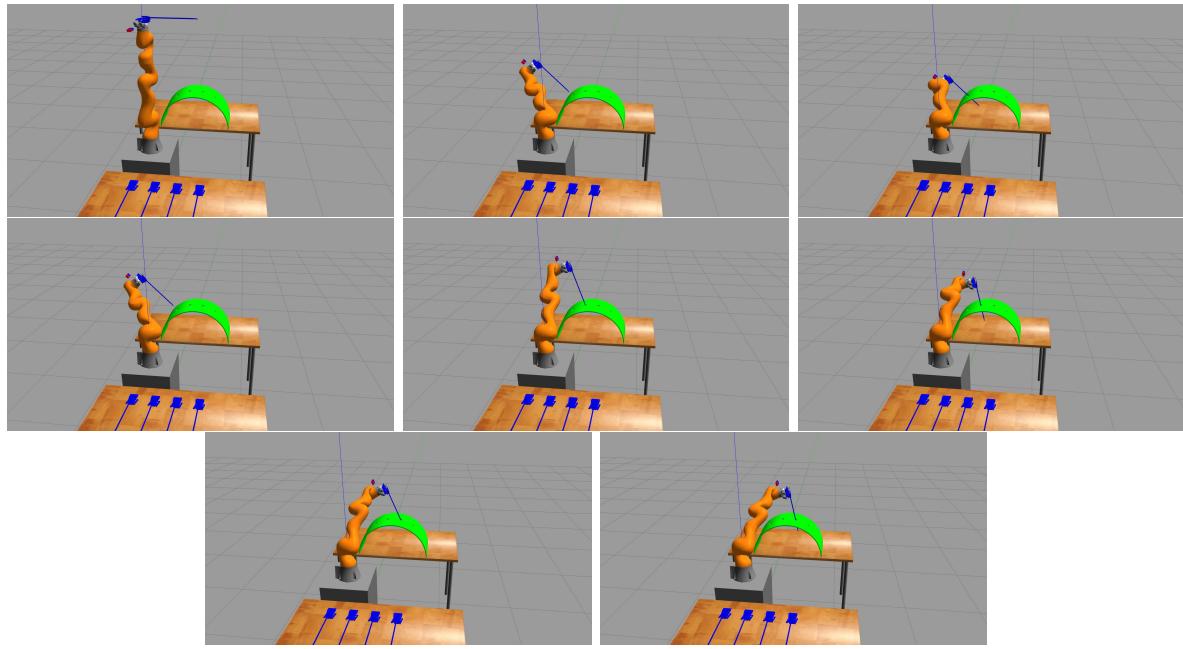


Figure 48: Experiment 2a:

To overcome the reachability issue shown in Figure 48, the algorithm was repeated, but this time using a different simulation layout in Gazebo, in which the mounting dock is closer to the robot and in front of it. This new layout enables the robot to reach all mounting holes with ease and with sufficient dexterity, the robot is free to pivot around.

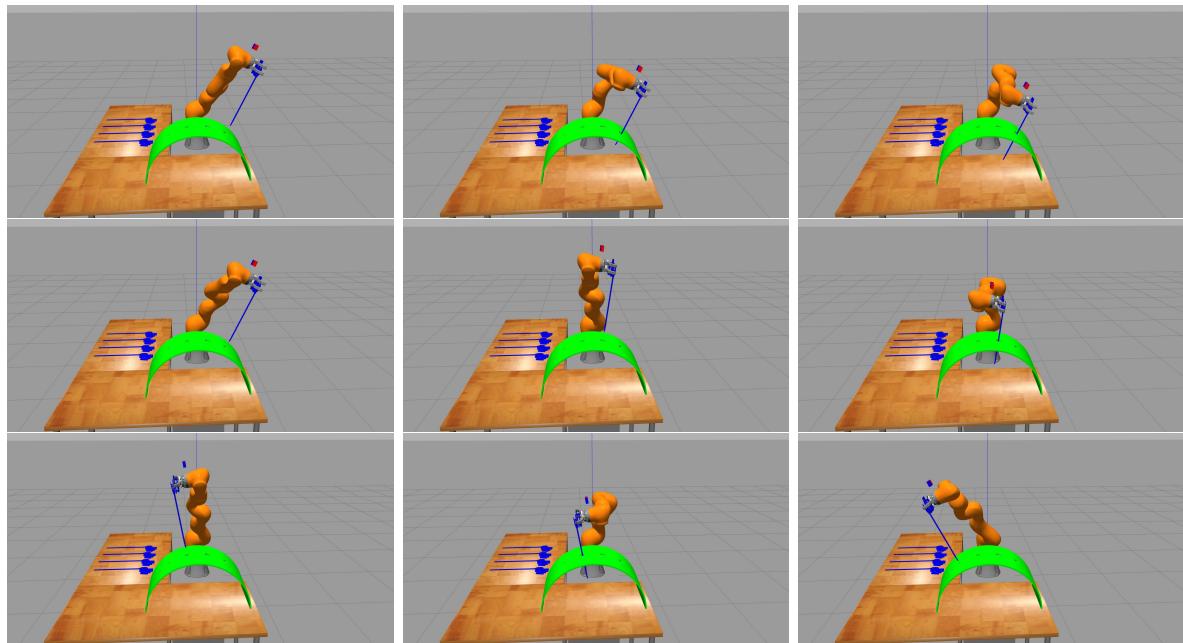


Figure 49: Experiment 2b:

Due to the probabilistic nature of the motion planner (in these experiments the OMPL library is used with the RRTConnect path planning algorithm), the solutions to the path planning problem are not always the same and thus it is possible that the robot arm reaches a pose which is close to a singularity

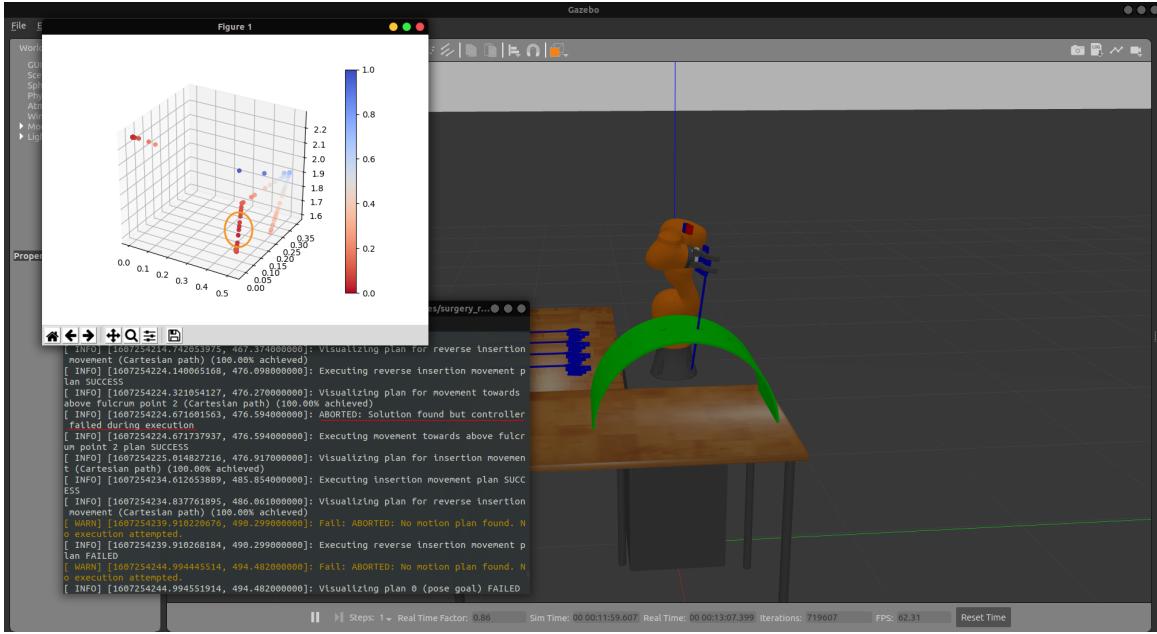


Figure 50: Experiment 2b: Singularity failure

9.3 Robot Planner 3: Trajectory planning

The goal of this third experiment is to design and test only some pivot trajectories. The pivot motions follow the equations described in 6.2. The trajectories that were designed and tested in this experiments are the following:

- Circular
 - Arc
 - Line segment

9.3.1 Circular and Circular arc trajectories in task space

9.3.2 Line segment trajectories in task space

		RRTConnect							
Robot Planner 3b	Place & Insert tool				Pivot trajectory				
Experiment	Status	Solution Time	Path Simplification Time	Planning Attempts	Status	Solution Time	Path Simplification Time	Planning Attempts	
1									
2									
3									
4									
5									
Average									

9.3.3 Cubic Spline trajectories in task space

		RRTConnect							
Robot Planner 3c	Place & Insert tool				Pivot trajectory				
Experiment	Status	Solution Time	Path Simplification Time	Planning Attempts	Status	Solution Time	Path Simplification Time	Planning Attempts	
1									
2									
3									
4									
5									
Average									

9.3.4 B-Spline trajectories in task space

		RRTConnect							
Robot Planner 3d	Place & Insert tool				Pivot trajectory				
Experiment	Status	Solution Time	Path Simplification Time	Planning Attempts	Status	Solution Time	Path Simplification Time	Planning Attempts	
1									
2									
3									
4									
5									
Average									

9.3.5 Polynomial trajectories in joint space

9.3.6 Trajectories in joint space with trapezoidal velocity profile

9.3.7 Trajectories in joint space with s-curve velocity profile

9.4 Robot Planner 4: Simple cube pick-and-place experiment

In this experiment we plan a simple pick-and-place path for a cube. The robotic arm first visits the left table and starts from the pre-grasp posture and then slowly approaches the cube until the grasp posture. When the gripper has reached the grasp posture, it closes the fingers to grasp the object and then retreats to the post-grasp posture. After that the robotic arm visits the right table to execute the place steps which are similar to the pick steps. The images below show some frames from the experiment with the first three to show the pick steps in the simulation environment and then other three images show the place steps in the visualization program.

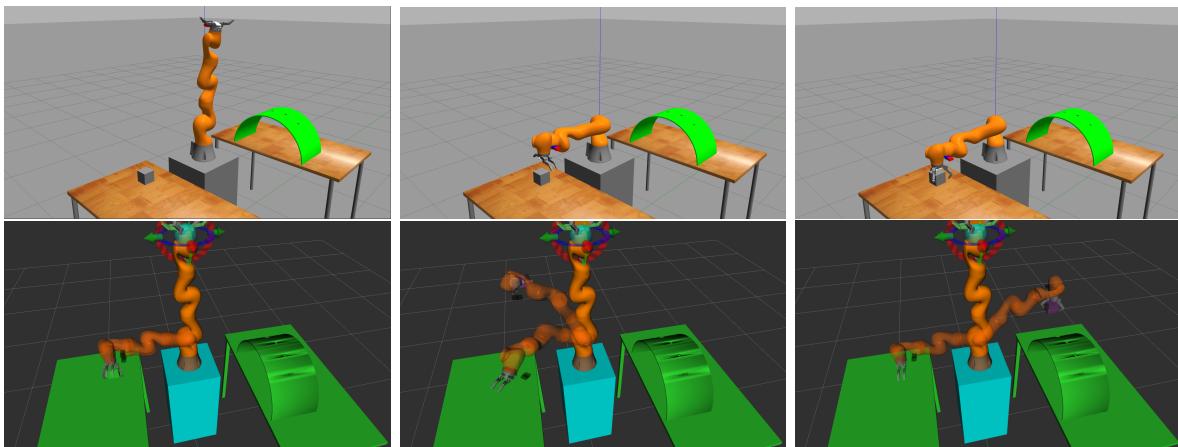


Figure 51: Experiment 4: Simple pick-and-place experiment of a cube

9.5 Robot Planner 5: Visual servoing

The goal of this fifth experiment is to control the KUKA robot using the camera via the visual servoing technique. In the first part of the experiment the robotic arm goes to an initial known position (e.g. the corner of the table) and then moves around until it detects a surgical tool. When that happens, the image-based visual servoing will send commands to the robot so that the detected tool is at the center of the video frame and with the same orientation as the video frame. At the second part of the experiment, the robot follows a similar algorithm. It starts with a known position, like the corner of the second table, then moves around until the mounting dock starts to appear inside the frame. When that happens, the image-based visual servoing will send commands to the robot so that the center of the trocar or the fulcrum reference frame is at the center and with the same orientation as the video frame. Similar results can be achieved by using position-based visual servoing, but that was not chosen at this implementation for simplicity and less operations. Position-based servoing requires more calculations because it is required to get the camera's transformation, express it with respect to the end-effector and then calculate the end-effector pose which is then used as an input to a Cartesian Controller.

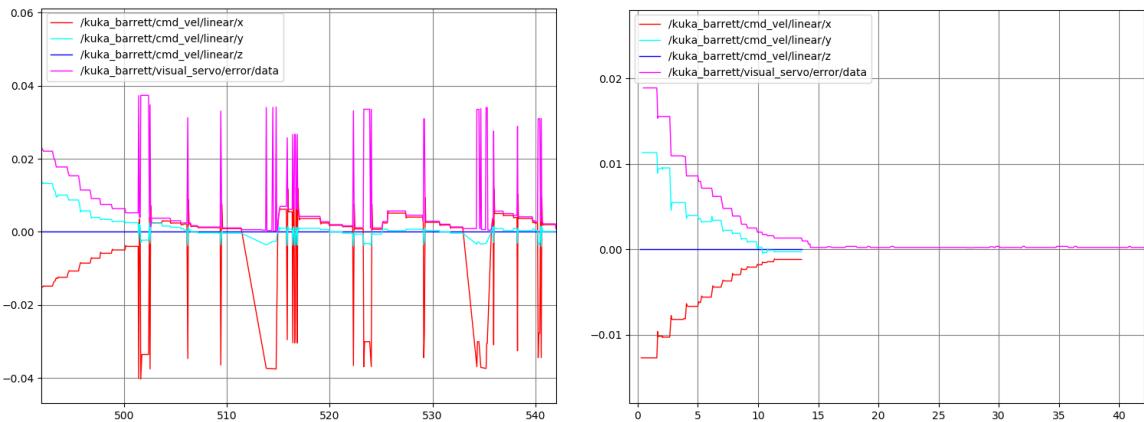


Figure 52: Visual servo controller error diagrams. On the left image in the error graphs appear some spikes. These spikes occur from the sudden temporary detection of a nearby surgical tool. On the right image, these spikes are filtered out, and only the error graphs of the visual servoing of one tool are shown. The controller parameters are $K_p = 0.9, K_d = 0.2$

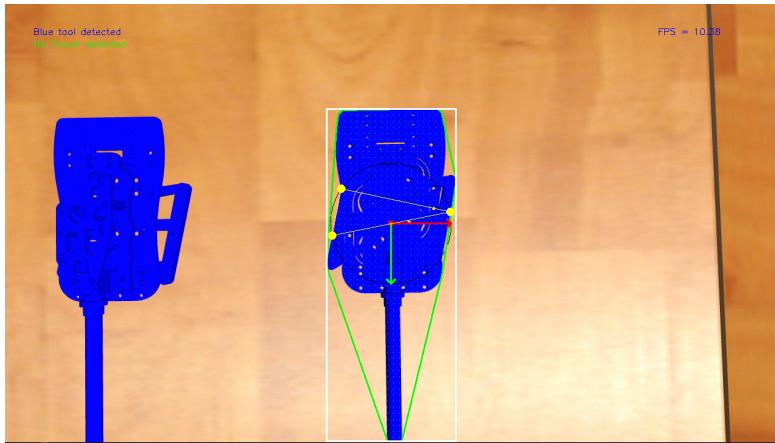


Figure 53: Image based visual servoing and calculation of grasp points. The yellow points are the grasp points and the thin black circumscribed circle is the growing circle that was used to calculate them.

9.6 Robot Planner 6**9.7 Robot Planner 7****9.8 Robot Planner 8: State machine & ROS Actions**

10 Conclusions and Future Work

10.1 Conclusions & Comparison with similar projects

10.2 Future Work

Simulation and interaction with deformable bodies

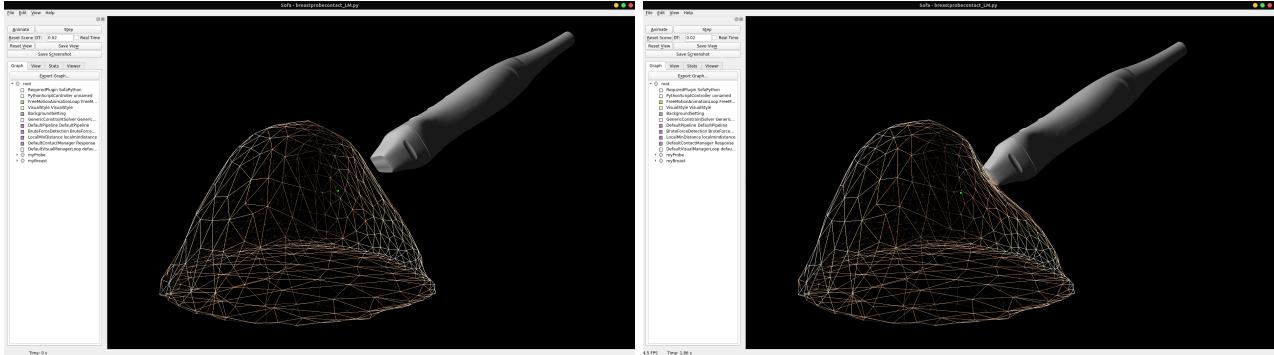


Figure 54: Deformable tissue/organ medical simulation - Simulation of breast probing using the SOFA Framework. Screenshots from running the repository at <https://gitlab.com/altairLab/probe-tissue-simulation>

Advanced visualization and Haptics feedback

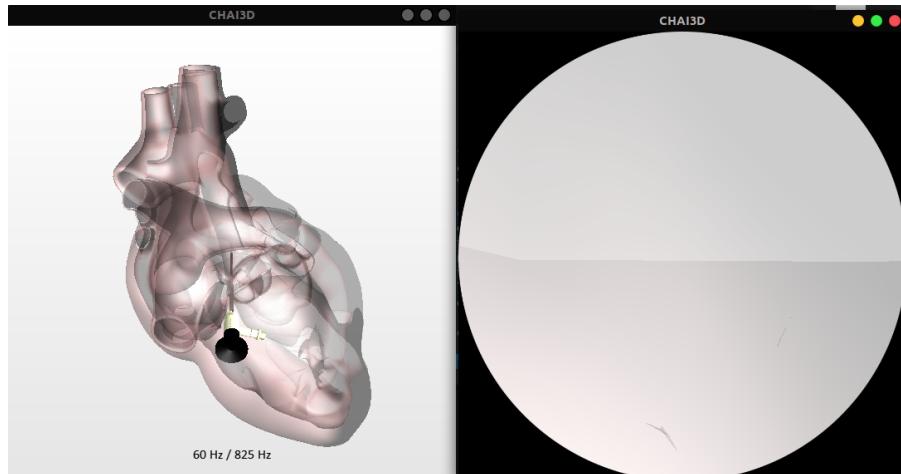


Figure 55: Heart endoscopy medical simulation using the CHAI3D framework. Screenshots from running the repository at <https://github.com/chai3d/chai3d>

Applications of Machine Learning in Computer Vision and Path Planning

Appendices

A Software and Documentation

- All software that developed for this thesis is available and will be maintained at https://github.com/karadalex/surgery_robotics_kuka_barrett
- Instruction on how to run the software of this thesis, as well as documentation of the various software components is available and will be maintained at https://karadalex.github.io/surgery_robotics_kuka_barrett/

B Mathematics

B.1 Euler angles to Quaternions

Let θ, ϕ, ψ be the Euler angles (roll, pitch, yaw) respectively, then using the following equations

$$c\theta = \cos\left(\frac{\theta}{2}\right), s\theta = \sin\left(\frac{\theta}{2}\right)$$

$$c\varphi = \cos\left(\frac{\varphi}{2}\right), s\varphi = \sin\left(\frac{\varphi}{2}\right)$$

$$c\psi = \cos\left(\frac{\psi}{2}\right), s\psi = \sin\left(\frac{\psi}{2}\right)$$

we can calculate the associated quaternion, in vector notation, as follows

$$\mathbf{q} = \begin{bmatrix} q_x \\ q_y \\ q_z \\ q_w \end{bmatrix} = \begin{bmatrix} s\theta c\varphi c\psi - c\theta s\varphi s\psi \\ c\theta s\varphi c\psi + s\theta c\varphi s\psi \\ c\theta c\varphi s\psi - s\theta s\varphi c\psi \\ c\theta c\varphi c\psi + s\theta s\varphi s\psi \end{bmatrix}$$

B.2 Cartesian to spherical coordinates

$$r = \sqrt{x^2 + y^2 + z^2}$$

$$\theta = \text{atan2}(\sqrt{x^2 + y^2}, z)$$

$$\varphi = \text{atan2}(y, x)$$

B.3 Spherical to cartesian coordinates

$$x = r \sin(\theta) \cos(\varphi)$$

$$y = r \sin(\theta) \sin(\varphi)$$

$$z = r \cos(\theta)$$

Nomenclature

$\hat{\mathbf{r}}, \hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\phi}}$ Unit vectors of r, θ, φ axes respectively, in spherical coordinates

$\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}}$ Unit vectors of x, y, z axes respectively

\mathcal{C} Configuration Space

\mathcal{C}_{free} Free Configuration Space

\mathcal{C}_{obs} Obstacle Space

\mathcal{L}_q Dexterity measure of the robotic arm

\mathcal{M} Manipulability measure of the robotic arm

θ, ϕ, ψ roll, pitch, yaw angles, also known as the Euler angles

${}^{i-1}\mathbf{p}_{iO}$ Position vector from the origin of the coordinate frame $\{i\}$ to the origin of the coordinate frame $\{i - 1\}$

${}^{i-1}R_i$ Rotation matrix from coordinate frame $\{i\}$ to coordinate frame $\{i - 1\}$

${}^{i-1}T_i$ Transformation matrix from coordinate frame $\{i\}$ to coordinate frame $\{i - 1\}$

c_i Shorthand notation for $\cos\theta_i$

J^\dagger Pseudoinverse of the Jacobian

s_i Shorthand notation for $\sin\theta_i$

List of Figures

1	The PUMA 560 robotic arm, which was the first to be used in surgery robotics in 1985	10
2	DaVinci Xi, ©2020 Intuitive Surgical, Inc. Patient Cart with the robotic arms that control the surgical tools	11
3	The Monarch™ Platform endoscopic system	12
4	DaVinci Xi ©2020 Intuitive Surgical, Inc. Surgeon Console	13
5	Motion planning pipeline	14
6	Kanban view of backlog tasks to organize all features, requirements and tasks needed to complete this thesis. The tool used to keep track of all tasks is Airtable	15
7	Joint reference frames of the KUKA iiwa14 robot	16
8	Approximation of the KUKA iiwa14 workspace, calculated with Forward Kinematics by randomly sampling the value ranges of the joints.	17
9	Reference frames of last link {7}, end-effector {TCP}, surgical tool base (center of mass) {B} and tool-tip frame {T}	18
10	Calculation of angles θ_2, θ_4	19
11	The first 4 out of 8 solutions of the Inverse Kinematics problem, using the decoupling technique	20
12	KUKA iiwa LBR14 workspace dimensions	20
13	Barrett Hand gripper (model BH8-282) dimensions	22
14	Simple tool detection in simulation based on color, using OpenCV. The green polygon is the convex hull, and the red point is the estimated center of mass	26
15	Disparity image calculated from the 2 cameras	26
16	Point cloud of surgical tools, generated from the 2 cameras and visualized in RViz	27
17	Estimation of tool's pose (position and orientation). The red dot is the center of mass and attached to that are the two orientation vectors of the tool. The green polygon is the convex hull of the tool and the white rectangle is it's ROI as calculated from the convex hull	28
18	Finding candidate grasping points from the intersections of a growing circle and the contour of the detected surgical tool	29
19	Simple trocar detection in simulation based on color, using OpenCV. In simulation, the trocar is simply considered to be a small cylindrical hole and it's center is the fulcrum point	30
20	Plot the manipulability of the robot arm at sample points of the executed trajectory .	33
21	Task space inside patients body. Colors with 0 or low value correspond to points with low dexterity	34
22	Tool pose at target point B calculated with respect to Fulcrum's reference frame {F} .	35
23	Circular trajectory of tool tip with respect to Fulcrum reference frame	38
24	Circular trajectory of tool tip with respect to Fulcrum reference frame and it's transformation via the Fulcrum Effect	39
25	Circular trajectory that lies on an a plane of arbitrary orientation with respect to the fulcrum point	40
26	Circular arc trajectory of tool tip with respect to Fulcrum reference frame and it's transformation via the Fulcrum Effect. In this trajectory 2 circular arcs are used	41
27	Line segment trajectory of tool tip with respect to Fulcrum reference frame	41
28	A Line segment trajectory and it's transformation via the Fulcrum Effect	42
29	Cubic Spline curve with 10 waypoints	43
30	A Cubic Spline trajectory and it's transformation via the Fulcrum Effect	44
31	Cubic Bézier curve calculated using cubic interpolation of 4 control points	44
32	A cubic Bézier curve calculated and plotted in MATLAB	45
33	B-Spline curve constructed from 3 Bézier curves. The first Bézier curve colored in red is a quadratic one and the following two are both cubic.	46
34	A Bézier curve trajectory and it's transformation via the Fulcrum Effect	47
35	Trajectory diagrams in joints space.	48

36	Joint trajectory generated using velocity profiles. Calculated and plotted in MATLAB	48
37	Force control on a Barrett Hand gripper finger	49
38	Position based visual servoing closed loop control	49
39	Position based visual servoing using depth from motion, stereo vision or 3D sensors, from which the desired position \mathbf{x}_d is calculated and used to drive the robot	50
40	Image based visual servoing. The robot arm is controlled using the information gained from the video frames. The frames are 2Dimensional and thus the detected objects can have only 3 degrees of freedom which means we can mainly control 3 independent variables, here the x, y, θ variables. The left image is the initial frame and the right image is the frame where the object is at the target pose.	51
41	Image based visual servoing closed loop control	51
42	Communication diagram of 2 ROS nodes with a topic and a service	52
43	Simulation environment in Gazebo	53
44	Control & Hardware Interfaces in Gazebo and ROS	54
45	RViz: Visualizing the robot state as well as the state of the perceived world In this screenshot, various poses of the robot are shown: 1) the current actual real pose of the robot, 2) the planned pose and 3) the goal pose, which can freely be moved within the RViz environment	54
46	MoveIt ROS Architecture	55
47	Experiment 1:	57
48	Experiment 2a:	58
49	Experiment 2b:	58
50	Experiment 2b: Singularity failure	59
51	Experiment 4: Simple pick-and-place experiment of a cube	61
52	Visual servo controller error diagrams. On the left image in the error graphs appear some spikes. These spikes occur from the sudden temporary detection of a nearby surgical tool. On the right image, these spikes are filtered out, and only the error graphs of the visual servoing of one tool are shown. The controller parameters are $K_p = 0.9, K_d = 0.2$	62
53	Image based visual servoing and calculation of grasp points. The yellow points are the grasp points and the thin black circumscribed circle is the growing circle that was used to calculate them.	62
54	Deformable tissue/organ medical simulation - Simulation of breast probing using the SOFA Framework. Screenshots from running the repository at https://gitlab.com/altairLab/probe-tissue-simulation	64
55	Heart endoscopy medical simulation using the CHAI3D framework. Screenshots from running the repository at https://github.com/chai3d/chai3d	64

List of Listings

1	Example ROS message with collision/contact information between one finger of the gripper and one surgical tool	23
1	RCM Taskspace calculation using MATLAB	34
2	Fulcrum Effect transformation of a trajectory, in MATLAB	37

List of Algorithms

1	Newton-Raphson numerical method	21
2	RRT Algorithm	31
3	PRM roadmap construction (preprocessing phase)	32
4	Pick and Place algorithm	32

Bibliography

- [1] Nastaran Aghakhani et al. "Task control with remote center of motion constraint for minimally invasive robotic surgery". In: *2013 IEEE International Conference on Robotics and Automation* (2013), pp. 5807–5812.
- [2] Jorge Angeles. *Fundamentals of Robotic Mechanical Systems*. Springer International Publishing, 2014. doi: 10.1007/978-3-319-01851-5. URL: <https://doi.org/10.1007/978-3-319-01851-5>.
- [3] Tadej Bajd et al. *Robotics*. Springer Netherlands, 2010. doi: 10.1007/978-90-481-3776-3. URL: <https://doi.org/10.1007/978-90-481-3776-3>.
- [4] E. Bauzano et al. "Active wrists endoscope navigation in robotized laparoscopic surgery". In: *2009 IEEE International Conference on Mechatronics*. Apr. 2009, pp. 1–6. doi: 10.1109/ICMECH.2009.4957177.
- [5] Mark de Berg et al. *Computational Geometry*. Springer Berlin Heidelberg, 2008. doi: 10.1007/978-3-540-77974-2. URL: <https://doi.org/10.1007/978-3-540-77974-2>.
- [6] E. Boctor et al. "Virtual Remote Center of Motion Control for Needle Placement Robots". In: *Computer aided surgery : official journal of the International Society for Computer Aided Surgery* 9 5 (2003), pp. 175–83.
- [7] M. C. Capolei et al. "Positioning the laparoscopic camera with industrial robot arm". In: *2017 3rd International Conference on Control, Automation and Robotics (ICCAR)*. Apr. 2017, pp. 138–143. doi: 10.1109/ICCAR.2017.7942675.
- [8] François Chaumette and Seth Hutchinson. "Visual Servoing and Visual Tracking". In: *Springer Handbook of Robotics*. Ed. by Bruno Siciliano and Oussama Khatib. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 563–583. ISBN: 978-3-540-30301-5. doi: 10.1007/978-3-540-30301-5_25. URL: https://doi.org/10.1007/978-3-540-30301-5_25.
- [9] Sachin Chitta et al. "ros_control: A generic and simple control framework for ROS". In: *The Journal of Open Source Software* (2017). doi: 10.21105/joss.00456. URL: <http://www.theoj.org/joss-papers/joss.00456/10.21105.joss.00456.pdf>.
- [10] Peter Corke. *Robotics, Vision and Control*. Springer International Publishing, 2017. doi: 10.1007/978-3-319-54413-7. URL: <https://doi.org/10.1007/978-3-319-54413-7>.
- [11] G. Cowley. "Introducing "Robodoc". A robot finds his calling—in the operating room". In: *Newsweek* 120.21 (Nov. 1992), p. 86.
- [12] Bassem Dahroug, B. Tamadazte, and N. Andreff. "3D Path Following with Remote Center of Motion Constraints". In: *ICINCO*. 2016.
- [13] Bassem Dahroug, B. Tamadazte, and N. Andreff. "Task Controller for Performing Remote Centre of Motion". In: *ICINCO*. 2016.
- [14] Bassem Dahroug, B. Tamadazte, and N. Andreff. "Visual servoing controller for time-invariant 3D path following with remote centre of motion constraint". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)* (2017), pp. 3612–3618.
- [15] Bassem Dahroug, Brahim Tamadazte, and Nicolas Andreff. "Unilaterally Constrained Motion of a Curved Surgical Tool". In: *Robotica* (2020), pp. 1–23. doi: 10.1017/S0263574719001735.
- [16] M. M. Dalvand and B. Shirinzadeh. "Remote centre-of-motion control algorithms of 6-RRCRR parallel robot assisted surgery system (PRAMiSS)". In: *2012 IEEE International Conference on Robotics and Automation* (2012), pp. 3401–3406.
- [17] Carlos Faria et al. "Position-based kinematics for 7-DoF serial manipulators with global configuration control, joint limit and singularity avoidance". In: *Mechanism and Machine Theory* 121 (2018), pp. 317–334. ISSN: 0094-114X. doi: <https://doi.org/10.1016/j.mechmachtheory.2017.10.025>. URL: <http://www.sciencedirect.com/science/article/pii/S0094114X17306559>.

- [18] Tully Foote. “tf: The transform library”. In: *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*. Open-Source Software workshop. Apr. 2013, pp. 1–6. DOI: 10.1109/TePRA.2013.6556373.
- [19] K. Fujii et al. “Gaze contingent cartesian control of a robotic arm for laparoscopic surgery”. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Nov. 2013, pp. 3582–3589. DOI: 10.1109/IROS.2013.6696867.
- [20] M. Fujii et al. “Relationship between workspace reduction due to collisions and distance between endoscope and target organ in pediatric endoscopic surgery”. In: *5th IEEE RAS/EMBS International Conference on Biomedical Robotics and Biomechatronics*. Aug. 2014, pp. 46–51. DOI: 10.1109/BIOROB.2014.6913750.
- [21] J. Funda et al. “Constrained Cartesian motion control for teleoperated surgical robots”. In: *IEEE Trans. Robotics Autom.* 12 (1996), pp. 453–465.
- [22] J. Funda et al. “Control and evaluation of a 7-axis surgical robot for laparoscopy”. In: *Proceedings of 1995 IEEE International Conference on Robotics and Automation*. Vol. 2. 1995, 1477–1484 vol.2.
- [23] Alessandro Gasparetto et al. “Path Planning and Trajectory Planning Algorithms: A General Overview”. In: *Motion and Operation Planning of Robotic Systems: Background and Practical Approaches*. Ed. by Giuseppe Carbone and Fernando Gomez-Bravo. Cham: Springer International Publishing, 2015, pp. 3–27. ISBN: 978-3-319-14705-5. DOI: 10.1007/978-3-319-14705-5_1. URL: https://doi.org/10.1007/978-3-319-14705-5_1.
- [24] G. Gras et al. “Implicit gaze-assisted adaptive motion scaling for highly articulated instrument manipulation”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. May 2017, pp. 4233–4239. DOI: 10.1109/ICRA.2017.7989488.
- [25] Techbriefs Media Group. *Surgical Robotics: The Evolution of a Medical Technology - Medical Design Briefs*. URL: <https://www.medicaldesignbriefs.com/component/content/article/mdb/features/technology-leaders/25006> (visited on 09/18/2020).
- [26] G. B. Hanna, S. Shimi, and A. Cuschieri. “Optimal port locations for endoscopic intracorporeal knotting”. In: *Surgical Endoscop* 11.4 (Apr. 1997), pp. 397–401. ISSN: 1432-2218. DOI: 10.1007/s004649900374. URL: <https://doi.org/10.1007/s004649900374>.
- [27] B. Hannaford et al. “Raven-II: An Open Platform for Surgical Robotics Research”. In: *IEEE Transactions on Biomedical Engineering* 60.4 (Apr. 2013), pp. 954–959. ISSN: 1558-2531. DOI: 10.1109/TBME.2012.2228858.
- [28] M. R. Hasan et al. “Modelling and Control of the Barrett Hand for Grasping”. In: *2013 UKSim 15th International Conference on Computer Modelling and Simulation*. Apr. 2013, pp. 230–235. DOI: 10.1109/UKSim.2013.142.
- [29] Felix C. Huang et al. “Learning kinematic mappings in laparoscopic surgery”. In: *Conference proceedings : ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Annual Conference 2010 (2010)*. PMC3280950[pmcid], pp. 2097–2102. ISSN: 1557-170X. DOI: 10.1109/IEMBS.2010.5626188. URL: <https://pubmed.ncbi.nlm.nih.gov/21095685/>.
- [30] J. Hutzl and H. Wörn. “Spatial probability distribution for port planning in minimal invasive robotic surgery (MIRS)”. In: *2015 6th International Conference on Automation, Robotics and Applications (ICARA)*. Feb. 2015, pp. 204–210. DOI: 10.1109/ICARA.2015.7081148.
- [31] Corke P. I. *Visual Control of Robots, high-performance visual servoing*. John Wiley& Sons Inc., 1996.
- [32] Craig J. Εισαγωγή στη ρομποτική, Μηχανική και Αυτόματος Ελεγχος. Τζιόλα, 2009. ISBN: 978-960-418-160-5.
- [33] Reza N. Jazar. *Theory of Applied Robotics, Kinematics, Dynamics, and Control (2nd Edition)*. Springer, Boston, MA, 2010. ISBN: 978-1-4419-1750-8. DOI: 10.1007/978-1-4419-1750-8.

- [34] S. Karaman and Emilio Frazzoli. “Sampling-based algorithms for optimal motion planning”. In: *The International Journal of Robotics Research* 30 (2011), pp. 846–894.
- [35] Abbas Karami et al. “Force, orientation and position control in redundant manipulators in prioritized scheme with null space compliance”. In: *Control Engineering Practice* 85 (2019), pp. 23–33.
- [36] Parham Kebria et al. “Kinematic and dynamic modelling of UR5 manipulator”. In: Oct. 2016, pp. 004229–004234. DOI: 10.1109/SMC.2016.7844896.
- [37] J. J. Kuffner and S. M. LaValle. “RRT-connect: An efficient approach to single-query path planning”. In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*. Vol. 2. Apr. 2000, 995–1001 vol.2. DOI: 10.1109/ROBOT.2000.844730.
- [38] I. Kuhlemann et al. “Robust inverse kinematics by configuration control for redundant manipulators with seven DoF”. In: *2016 2nd International Conference on Control, Automation and Robotics (ICCAR)*. Apr. 2016, pp. 49–55. DOI: 10.1109/ICCAR.2016.7486697.
- [39] P. Lago, Carlo Lombardi, and I. Vallone. “From laparoscopic surgery to 3-D double console robot-assisted surgery”. In: *Proceedings of the 10th IEEE International Conference on Information Technology and Applications in Biomedicine* (2010), pp. 1–4.
- [40] A. Llamosi and S. Toussaint. “Measuring Force Intensity and Direction with a Spatially Resolved Soft Sensor for Biomechanics and Robotic Haptic Capability”. In: *Soft Robot* 6.3 (June 2019), pp. 346–355.
- [41] Kevin M Lynch and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control*. English (US). Cambridge University Press, 2017. ISBN: 978-1107156302.
- [42] M. Marinho et al. “A Unified Framework for the Teleoperation of Surgical Robots in Constrained Workspaces”. In: *2019 International Conference on Robotics and Automation (ICRA)* (2019), pp. 2721–2727.
- [43] Murilo M. Marinho, Mariana C. Bernardes, and Antônio Padilha Lanari Bo. “A programmable remote center-of-motion controller for minimally invasive surgery using the dual quaternion framework”. In: *5th IEEE RAS/EMBS International Conference on Biomedical Robotics and Biomechatronics* (2014), pp. 339–344.
- [44] Q. Mei et al. “PROBOT — A computer integrated prostatectomy system”. In: *Visualization in Biomedical Computing*. Ed. by Karl Heinz Höhne and Ron Kikinis. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 581–590. ISBN: 978-3-540-70739-4.
- [45] Victor F. Muñoz et al. “Pivoting motion control for a laparoscopic assistant robot and human clinical trials”. In: *Advanced Robotics* 19 (2005), pp. 694–712.
- [46] A. A. Navarro et al. “Automatic positioning of surgical instruments in minimally invasive robotic surgery through vision-based motion analysis”. In: *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Oct. 2007, pp. 202–207. DOI: 10.1109/IROS.2007.4399150.
- [47] Jason M. O’Kane. *A Gentle Introduction to ROS*. Available at <http://www.cse.sc.edu/~jokane/agitr/>. Independently published, Oct. 2013. ISBN: 978-1492143239.
- [48] Claudio Pacchierotti. “Needle Insertion in Simulated Soft Tissue”. In: *Cutaneous Haptic Feedback in Robotic Teleoperation*. Cham: Springer International Publishing, 2015, pp. 21–36. ISBN: 978-3-319-25457-9. DOI: 10.1007/978-3-319-25457-9_2. URL: https://doi.org/10.1007/978-3-319-25457-9_2.
- [49] Jaydeep Palep. “Robotic assisted minimally invasive surgery”. In: *Journal of minimal access surgery* 5 (Feb. 2009), pp. 1–7. DOI: 10.4103/0972-9941.51313.
- [50] C. Pham et al. “Analysis of a moving remote center of motion for robotics-assisted minimally invasive surgery”. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2015), pp. 1440–1446.

- [51] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In: *ICRA Workshop on Open Source Software*. 2009.
- [52] R. Woods R. Gonzalez. *Ψηφιακή Επεξεργασία Εικόνας*. Εκδόσεις ΤΖΙΟΛΑΣ, 2018.
- [53] S. Roh et al. "Development of the SAIT single-port surgical access robot slave arm based on RCM Mechanism". In: *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)* (2015), pp. 5285–5290.
- [54] *ROS/Concepts - ROS Wiki*. URL: <https://wiki.ros.org/ROS/Concepts> (visited on 09/19/2020).
- [55] Benoît Rosa et al. "Estimation of optimal pivot point for remote center of motion alignment in surgery". In: *International Journal of Computer Assisted Radiology and Surgery* 10.2 (Feb. 2015), pp. 205–215. ISSN: 1861-6429. DOI: 10.1007/s11548-014-1071-3. URL: <https://doi.org/10.1007/s11548-014-1071-3>.
- [56] Hamid Sadeghian, Fatemeh Zokaei, and Shahram Hadian Jazi. "Constrained Kinematic Control in Minimally Invasive Robotic Surgery Subject to Remote Center of Motion Constraint". In: *Journal of Intelligent & Robotic Systems* (2019), pp. 1–13.
- [57] H. Shao et al. "A New CT-Aided Robotic Stereotaxis System". In: 1985.
- [58] Bruno Siciliano et al. *Robotics: Modelling, Planning and Control*. 1st. Springer Publishing Company, Incorporated, 2008. ISBN: 1846286417, 9781846286414.
- [59] I. A. Şucan and S. Chitta. *Moveit!* URL: <https://moveit.ros.org/> (visited on 04/28/2020).
- [60] V. Vitiello et al. "Emerging Robotic Platforms for Minimally Invasive Surgery". In: *IEEE Reviews in Biomedical Engineering* 6 (2013), pp. 111–126.
- [61] Go Watanabe, ed. *Robotic Surgery*. Springer Japan, 2014. DOI: 10.1007/978-4-431-54853-9. URL: <https://doi.org/10.1007/978-4-431-54853-9>.
- [62] Haiwang Xu et al. "Laparoscopic Robot Design and Kinematic Validation". In: Jan. 2007, pp. 1426–1431. DOI: 10.1109/ROBIO.2006.340138.
- [63] Tao Yang et al. "Mechanism of a Learning Robot Manipulator for Laparoscopic Surgical Training". In: *Intelligent Autonomous Systems 12*. Ed. by Sukhan Lee et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 17–26. ISBN: 978-3-642-33932-5.
- [64] L. Yu et al. "Research on the Trajectory Planning of the Minimally Invasive Surgical Robot 6-DOF Manipulator". In: *2010 International Conference on Biomedical Engineering and Computer Science*. Apr. 2010, pp. 1–4. DOI: 10.1109/ICBECS.2010.5462451.
- [65] Xiaoqin Zhou et al. "New remote centre of motion mechanism for robot-assisted minimally invasive surgery". In: *BioMedical Engineering OnLine* 17 (2018).
- [66] Νίκος Ασπράγκαθος. *Ρομποτική. Τηλήμα Μηχανολόγων & Αεροναυπηγών Μηχανικών, Πανεπιστήμιο Πατρών*, 2018.
- [67] Ζωή Δουλγέρη. *Ρομποτική*. Κριτική, 2007. ISBN: 978-960-218-502-5.
- [68] Κωνσταντίνος Μουστάκας. *Τυπολογιστική Γεωμετρία και Εφαρμογές 3Δ Μοντελοποίησης*. Ανακτήθηκε την Κυριακή, 19 Ιουλίου 2020 από <https://eclass.upatras.gr/courses/EE844/>.