UNIVERSITY OF PATRAS
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Division: Systems and Automatic Control

Perception, control and path planning of robotic laparoscopic surgical system

Diploma Thesis

Alexios Karadimos

Supervisor

Evangelos Dermatas
Associate Professor, University of Patras

Co-Supervisor

Anthony Tzes
Professor, NYU Abu Dhabi

Patras - February 2022

University of Patras, Department of Electrical and Computer Engineering.
Alexios Karadimos

CERTIFICATION

It is certified that the Diploma Thesis titled

Perception, control and path planning of robotic laparoscopic surgical system

of the Department of Electrical and Computer Engineering student

Alexios Karadimos of Loukas

Registration Number: 1046820

was presented publicly at the Department of Electrical and Computer Engineering at

02/02/2022

and was examined by the following examining committee:

Evangelos Dermatas, Associate Professor(supervisor)
Konstantinos Moustakas, Professor (committee member)
Kyriakos Sgarbas, Associate Professor (committee member)

The supervisor

The Director of the Division

Evangelos Dermatas
Associate Professor

Nikolaos Koussoulas
Professor

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ

Τομεας: Συστηματων και Αυτοματου Ελεγχου

Αντίληψη, έλεγχος και σχεδιασμός διαδρομής ρομποτικού λαπαροσκοπικού χειρουργικού συστήματος

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΑΛΕΞΙΟΣ ΚΑΡΑΔΗΜΟΣ

Επιβλέπων

Ευάγγελος Δερματάς
Αναπληρωτής Καθηγητής, Πανεπιστήμιο Πατρών

Συνεπιβλέπων

Αντώνιος Τζες
Καθηγητής, NYU Abu Dhabi

Πάτρα - Φεβρουάριος 2022

Πανεπιστήμιο Πατρών, Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών.
Καραδήμος Αλέξιος

ΠΙΣΤΟΠΟΙΗΣΗ

Πιστοποιείται ότι η Διπλωματική Εργασία με τίτλο

Αντίληψη, έλεγχος και σχεδιασμός διαδρομής ρομποτικού λαπαροσκοπικού χειρουργικού συστήματος

του φοιτητή του Τμήματος Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών

Αλέξιος Καραδήμος του Λουκά

Αριθμός Μητρώου: 1046820

Παρουσιάστηκε δημόσια στο Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών στις

02/02/2022

και εξετάστηκε από την ακόλουθη εξεταστική επιτροπή:

Ευάγγελος Δερματάς, Αναπληρωτής Καθηγητής (επιβλέπων)
Κωνσταντίνος Μουστάκας, Καθηγητής (μέλος επιτροπής)
Κυριάκος Σγάρμπας, Αναπληρωτής Καθηγητής (μέλος επιτροπής)

Ο Επιβλέπων

Ευάγγελος Δερματάς
Αναπληρωτής Καθηγητής

Ο Διευθυντής του Τομέα

Νικόλαος Κούσουλας
Καθηγητής

Abstract

This thesis studies all the stages involved in the perception, control and manipulation of robotic laparoscopic tools with emphasis given to the pivot trajectories and the RCM constrained motion planning. The forward and inverse kinematics of the 7 DoF KUKA iiwa14 industrial robot arm was studied followed by kinematics of the attached Barrett hand gripper for grasping purposes. Minimal Invasive Surgery (MIS) necessitates the study of Remote Center-of-Motion (RCM) constraint for the pivot motions, the elbow-up constraint to avoid collisions as well as the workspace constraints and singularities. The transformation of the surgical task space into the robot's taskspace and the joint space is subsequently analyzed along with the robot's manipulability. Emphasis was given in calculating various geometric paths for the robot to follow inside the surgical task. The equations for circular, circular arc, line segment, helical, cubic spline, b-spline, higher-order polynomial and trapezoid and s-curve velocity profile trajectories are studied in detail in order to generate pivot motions with a wide variety. Simulations were conducted using the ROS framework and libraries like Gazebo, RViz and MoveIt using the RRTConnect path planning algorithm. The simulations were evaluated with measurements of time, position accuracy and RCM distance deviation. This thesis also briefly studies a simple recognition of a laparoscopic tool and the estimation of its position and orientation using computer vision as well as the calculation of three points on the surgical tool where the gripper's fingers will be placed in order to grasp the object with a satisfactory force closure. Finally this thesis studies some control system schemes like for example the RCM tracking and pivot motion control.

Περίληψη

Η παρούσα διπλωματική εξετάζει όλα τα στάδια που εμπλέκονται στην αναγνώριση, τον έλεγχο και τον χειρισμό των λαπαροσκοπικών εργαλείων με μια ολιστική προσέγγιση αλλά με μεγαλύτερη έμφαση να δίνεται στις τροχιές περιστροφής γύρω από σημείο και στον προγραμματισμό περιορισμένης κίνησης RCM. Το πρώτο βήμα ήταν να μελετηθεί το ευθύ και αντίστροφο κινηματικό πρόβλημα του βιομηχανικού ρομποτικού βραχίονα KUKA iiwa14 (με την τεχνική αποσύνδεσης θέσης- προσανατολισμού για ρομπότ 6 βαθμών ελευθερίας, αφήνοντας τον επιπλέον βαθμό ελευθερίας να καθοριστεί από άλλους περιορισμούς) καθώς και η κινηματική της αρπάγης Barrett ώστε να υπολογιστούν λαβές του χειρουργικού εργαλείου. Δεδομένου ότι η ρομποτική MIS επιβάλλει πολλούς περιορισμούς, το επόμενο βήμα ήταν να μελετηθεί ο περιορισμός RCM για τις κινήσεις περιστροφής (pivot), ο περιορισμός στον οποίο ο αγκώνας του ρομπότ πρέπει να είναι προς τα πάνω για την αποφυγή συγκρούσεων καθώς και οι περιορισμοί και τα σημεία ενικότητας του χώρου εργασίας. Κατά το σχεδιασμό εφαρμογών ρομπότ είναι σημαντικό να μελετηθεί ο χώρος εργασίας του ρομπότ και για το λόγο αυτό, αυτή στη διπλωματική αυτή μελετάται επιπλέον ο χώρος χειρουργικών εργασιών και πώς αυτός μετατρέπεται στον χώρο εργασιών του ρομπότ και στον χώρο αρθρώσεών του. Μελετήθηκε επίσης ο δείκτης επιδεξιότητας του ρομπότ και η κατάλληλη διάταξη του ρομπότ σε σχεση με το περιβάλλον του έτσι ώστε όλες οι τροχιές να είναι καλά προσβάσιμες και να μπορουν να εκτελεστούν με ευκολία. Δόθηκε μεγάλη έμφαση στον υπολογισμό διαφόρων γεωμετρικών μονοπατιών που έπρεπε να ακολουθήσει το ρομπότ μέσα στο χώρο χειρουργικής εργασίας. Οι εξισώσεις για τροχιές κύκλου, κυκλικού τόξου, ευθύγραμμου τμήματος, έλικα, κυβικού spline, b-spline, πολυωνύμων υψηλότερης τάξης και τροχιές με προφίλ ταχύτητας τραπεζοειδές και καμπύλης-s, μελετώνται λεπτομερώς, προκειμένου να δημιουργηθούν χειρουργικές ρομποτικές κινήσεις με μεγάλη ποικιλία. Όλα τα πειράματα υλοποιήθηκαν χρησιμοποιώντας το περιβάλλον ROS και δημοφιλή εργαλεία και βιβλιοθήκες όπως τα Gazebo, RViz και MoveIt χρησιμοποιώντας τον αλγόριθμο σχεδιασμού διαδρομής RRTConnect. Τα πειράματα αξιολογήθηκαν με μετρήσεις χρόνου, ακρίβειας θέσης και απόκλισης απόστασης RCM. Στη διπλωματική αυτή μελετάται επίσης εν συντομία η απλή αναγνώριση ενός λαπαροσκοπικού εργαλείου και η εκτίμηση της θέσης και του προσανατολισμού του με χρήση υπολογιστικής όρασης καθώς και ο υπολογισμός 3 σημείων πάνω στο χειρουργικό εργαλείο όπου θα τοποθετηθούν τα δάχτυλα της αρπάγης για να πιάσει το αντικείμενο με μία ικανοποιητική λαβή. Τέλος, σε αυτή τη διπλωματική μελετώνται ορισμένα συστήματα ελέγχου όπως για παράδειγμα ενός συστήματος για την παρακολούθηση RCM και τον έλεγχο κίνησης περιστροφής.

Acknowledgements

The subject of this thesis is a subject that I have been passionate about for many years and from the beginning of my university studies. For this reason I would like to thank my supervisors, Evangelos Dermatas and Anthony Tzes, for their guidance and for giving me the opportunity to dive deep into this fascinating subject. I would also like to thank Nikolaos Evangeliou (PostDoctoral Associate at NYU Abu Dhabi) for giving me feedback on this thesis. Last but not least, I would like to thank my friends and family for their huge support and for pushing me during the pandemic to conquer my academic goals.

Contents

Chapter 1

Introduction

1.1 Surgical robotics

1.1.1 Historical Overview of Surgical robotics

In surgical robotics, the surgeon operates on the patient using a computer-controlled robotic arm, 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 followed by several laparoscopic operations. Such surgical operations are characterised as **minimally invasive**, because the surgical incisions made at the patient are very small leading to a small probability of infection of the patient. a reduction of the hospitalization time and an overall faster patient recovery.
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 while being very susceptible to the surgeon's micro movements and tremble.
The first application of robotics in surgery appears in 1985, when Kwoh et al. [64] 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 [50], which was developed at the Imperial College and used in a prostatectomy operation. Another example of an early surgery robot was the **ROBODOC** system [12] 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).

Some other important surgery robots are listed in the sequel:

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

Figure 1.1: First Surgical Robot PUMA 560 (circa 1985)

- **daVinci Surgical System®**: The most popular surgery robots in a master-slave configuration, implying that the operation commands are sent uni-directionally 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 tooling system, one for each hand, called EndoWrist®. It is officially approved by the FDA for laparoscopic surgeries.
- **SOCRATES Robotic Telecollaboration System**
- **Raven-II** [30]: An open platform for collaborative research on surgical robotics.
- **Monarch™Platform** by Auris Health Inc., an endoscopic system for robotic-assisted bronchoscopy

Figure 1.2: DaVinci Xi, ©2020 Intuitive Surgical, Inc. Patient Cart with the robotic arms that control the surgical tools

Figure 1.3: The Monarch™Platform endoscopic system [1]

1.1.2 Surgical Robotic 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 followed by an insertion of the laparoscopic tools. After the patient is prepared and after the patient cart, which carries the robotic arms, is successfully positioned and callibrated, the surgeon sits on a console, from where he/she controls the robot via special sensitive joysticks. The surgeon has 3D-visual access 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.

The advantages offered by Surgical robotics include but not limited to:

- **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

[1] https://www.aurishealth.com/patients/robotic-bronchoscopy-patient-about-monarch-platform
[2] https://www.intuitive.com/en-us/about-us/press/press-resources

Figure 1.4: DaVinci Xi ©2020 Intuitive Surgical, Inc. Surgeon Console [2]

  - 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**: the surgeon operates while he/she sits on a special **ergonomic** console, which makes the long procedures more comfortable and efficient.

1.2 Thesis Strategic Goals

The goal of this thesis is to design the kinematic models and algorithms necessary for a robotic arm to detect, grasp and manipulate a laparoscopic surgical tool. To achieve these goals the robotic arm should be capable of the following:

- Visually detect the scene and laparoscopic tools and with the help of stereoscopic vision, and calculate the relative position and orientation of the center of mass of each tool
- 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 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 Related Past Overview

Surgery robotics, in terms of robot mechanics, can be divided into two categories, specialized surgical robots and generic robotic arms that are programmed for surgical tasks. The specialized surgical robots have specific mechanisms which ensure that the RCM constraint is satisfied mechanically. These robots are easier to control but are only suitable for very specific operations and are usually large in size, with complicated structures that require more maintenance.

A new methodology to perform MIS-procedures is by using industrial robots which are programmable to respect the RCM constraint. This methodology is studied in [49], where a controller is designed to minimize the deviation from the RCM point using kinematics with the dual quaternion framework. A similar approach is studied in [5], where a controller is designed for an active wrist robot to mimic the passive wrist mechanism, which means not exerting pressure on the abdominal wall and respecting the RCM constraint while executing the trajectory with smaller error.

This thesis follows a similar approach, where the laparoscopic tool is attached on an active wrist end-effector of a popular non-surgical industrial robot (KUKA lwr iiwa14) and the RCM constraint is satisfied with non specialized mechanisms or a passive wrist.

Minimizing the RCM error is important in order to execute precise trajectories but also to minimize the force exerted to the patient. In [4], this deviation error is studied more thoroughly and is used to study the force interaction model of the surgical tool (endoscopic camera) and the abdominal wall. The force that is exerted can be calculated from the fulcrum displacement and the abdominal wall elasticity constant which can be estimated experimentally.

The trajectories that a laparoscopic tool must follow, pivot around the RCM constraint point. This point is also known in the bibliography as the fulcrum point, pivot point, trocar point, and incision point. To design a trajectory, this point is assumed to be known within some bounds. In [18] an algorithm is proposed that estimates the position of the trocar during a robot-assisted endoscopic surgery. The trocar position is estimated with the Least Squares algorithm which is used to calculate the intersection of surgical tool axes measured from a given number of consecutive robot configurations. A different approach of estimating the Fulcrum point is presented in [28], where the fulcrum point is estimated using an Extended Kalman FIlter which uses a stochastic measurement and prediction model and it is able to produce adeqiuate results in the presence of noise.

When designing a control system for the robot, capable of executing the pivot motions, there are two approaches. The first approach is to directly design a control system whose control parameters adapt to the errors (direct adaptive control) and the second approach is to initially estimate the fulcrum parameters or assume they are known and then apply a usual control system (indirect adaptive control). An example of the second approach is studied in [51], where a geometrical estimator is used to estimate the outside penetration of the surgical tool, which is then used in a PI controller and a trajectory generator.

1.4 Adopted Methodology & Approach

In designing a robotic arm to execute a surgery task, there is a need 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.

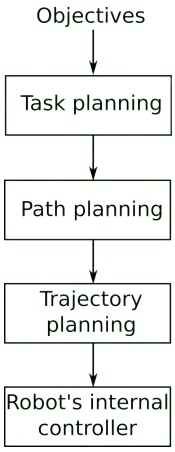

Figure 1.5: Motion planning pipeline

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

To quickly test out ideas on how to build the robot's environment and the simulation layout, as well as some simple trajectories, the CoppeliaSim simulator software was used (also known previously as VREP). CoppeliaSim allowed to do quick prototyping using the intuitive drag-and-drop interface and the embedded scripts, before implementing the actual simulation in ROS which is more complex and time-consuming (but also more feature-rich).

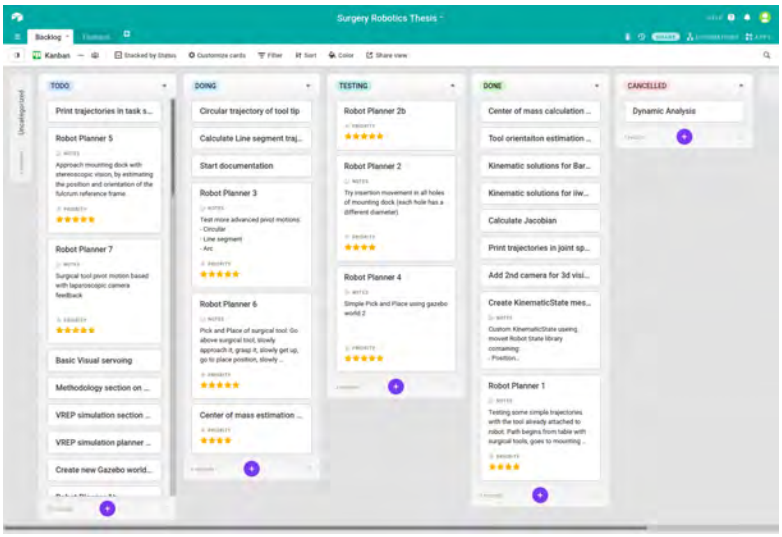

Figure 1.6: Kanban view of backlog tasks to organize all features, requirements and tasks needed to complete this thesis. The tool used to keep track of all tasks is Airtable

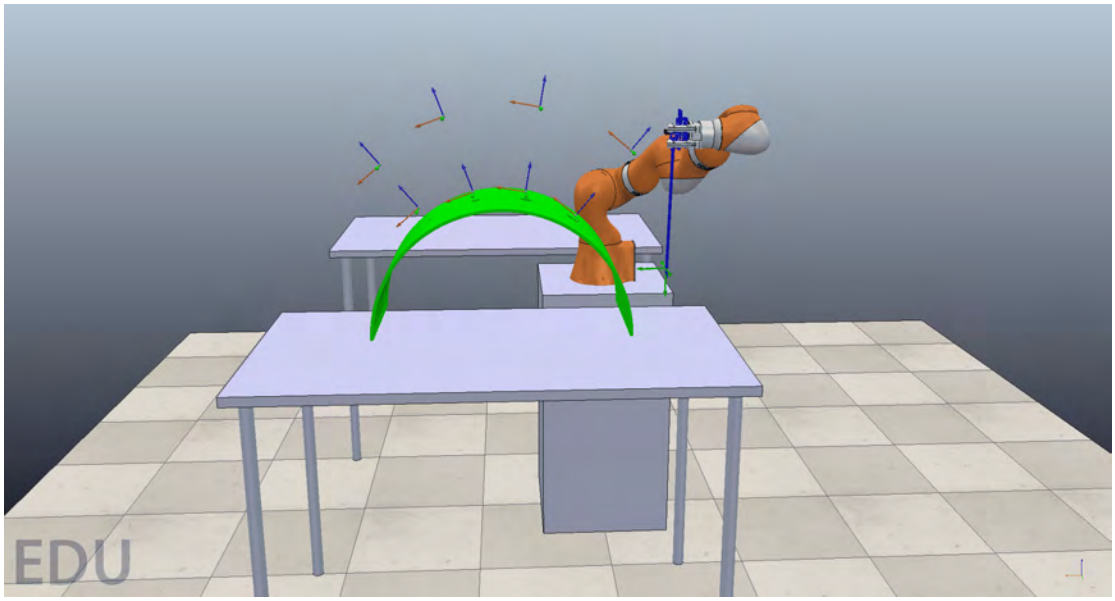

Figure 1.7: Quick Prototyping using the VREP (CoppeliaSim) simulation environment

Chapter 2

Robotic arm kinematic Analysis

2.1 Robotic arm, D-H parameters & Forward Kinematics

The Forward Kinematics (FK) problem seeks to specify the transformations between every pair of consecutive robot links. The four**Denavit-Hartenberg** (D-H) parameters can describe the kinematic chain. 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

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	$\pi/2$

Table 2.1: D-H parameters for Kuka iiwa14

The goal of the FK-problem is to find the transformation of the reference frame of the end-effector with respect to the universal reference frame. To achieve that, we must first calculate the transformations between each pair of joints. To simplify each such transformation, i.e. the transformation from a frame $\{i-1\}$ to a frame $\{i\}$, we consider 3 intermediary reference frames $\{A\}$, $\{B\}$ and $\{C\}$. Then the transformations between

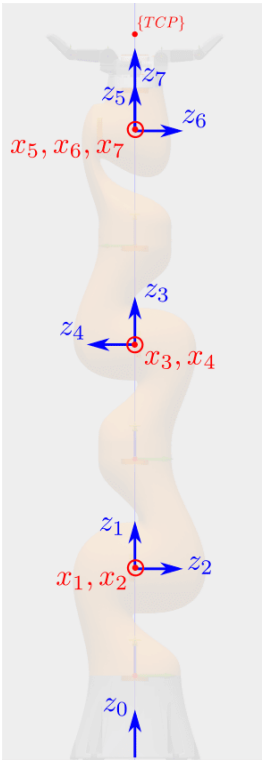

Figure 2.1: Joint reference frames of the KUKA iiwa14 robot

the two joints is given by

$$ {}^{i-1}T_i = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & L_{i-1} \\ s\theta_i ca_{i-1} & c\theta_i ca_{i-1} & -sa_{i-1} & -sa_{i-1}d_i \\ s\theta_i sa_{i-1} & c\theta_i sa_{i-1} & ca_{i-1} & ca_{i-1}d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{2.1.0.1} $$

where $c\theta_i = \cos(\theta_i)$, $s\theta_i = \sin(\theta_i), sa_i = \sin(a_i)$ and $ca_i = \cos(a_i)$.
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 can be found by

$$ {}^0T_N = {}^0T_1 \cdot {}^1T_0 \cdots {}^{N-1}T_N. \tag{2.1.0.2} $$

The orientation is the upper-left 3×3 matrix and the position is given by the fourth column of the matrix ${}^0T_N$.
The workspace of the robot arm is the set of all points in $\mathbb{R}^3$ that the tip of the manipulator can reach.

2.1.1 End-effector to tool-tip transformations

The surgical tool is attached to the end-effector of the arm; the corresponding frames are in agreement with the drawing shown in Figure 2.3. The corresponding transformations

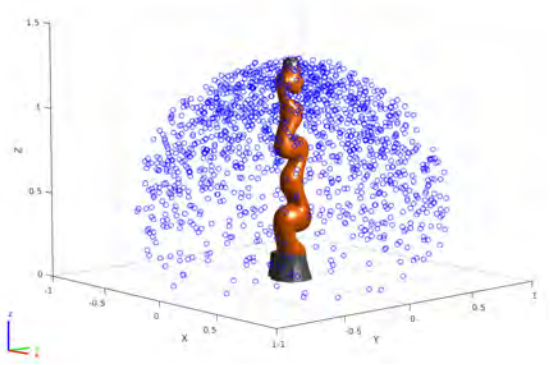

Figure 2.2: KUKA iiwa14 workspace calculated with FK by randomly sampling the value ranges of the joints.

can be found as

$$^{7}T_{TCP} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.094 \\ 0 & 0 & 0 & 1 \end{bmatrix}, ^{TCP}T_{B} = \begin{bmatrix} 1 & 0 & 0 & 0.05 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, ^{B}T_{T} = \begin{bmatrix} 1 & 0 & 0 & -0.01 \\ 0 & 1 & 0 & 0.022 \\ 0 & 0 & 1 & 0.455 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

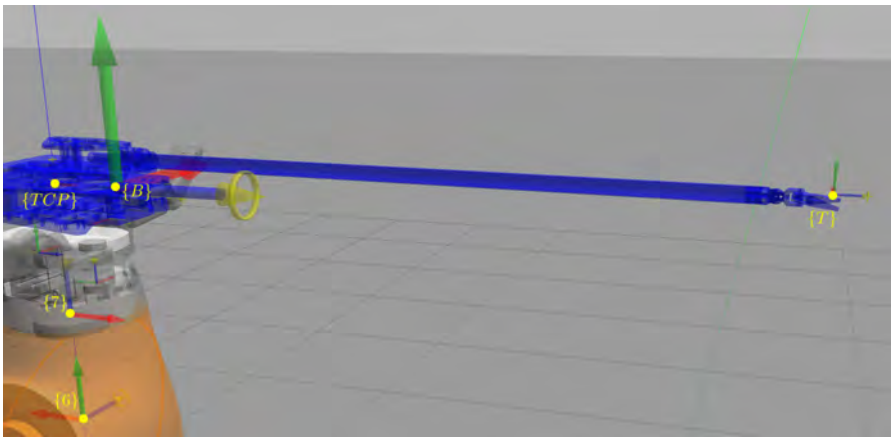

Figure 2.3: Reference frames of last link {7}, end-effector $\{TCP\}$, surgical tool base (center of mass) $\{B\}$ and tool-tip frame $\{T\}$

2.2 Inverse Kinematics

The Inverse Kinematics (IK) problem seeks to find those joint values that make the robot's end effector to be at a specific desired position and orientation. For a robot with more Degrees of Freedom (DoF) than the demanded six to fully position and orient a robot in 3D space (3 for position and 3 for orientation), the IK has infinite solutions. Thus 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**.

2.2.1 Decoupling Technique

In this section the IK problem is solved for 6 out of the 7 DoF, where the third joint is not used in this analysis and its angle is set to zero $\theta_3 = 0$. Using the decupling technique, the IK-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 ${}^UT_{tcp}$ 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

$$ {}^UT_{TCP} = {}^UT_0\ {}^0T_7\ {}^7T_{TCP},\ {}^0T_7 = {}^UT_0^{-1}\ {}^UT_{TCP}\ {}^7T_{TCP}^{-1},\ {}^0T_7 = \begin{bmatrix} R_t & \mathbf{p}_t \\ 0 & 1 \end{bmatrix} \quad (2.2.1.1) $$

where ${}^UT_0$, ${}^7T_{tcp}$ are translation transformations by a constant distance and R_t, $\mathbf{p}_t$ are the target's orientation and position respectively.
Simple geometric principles can be used to infer the relationships for all angles.

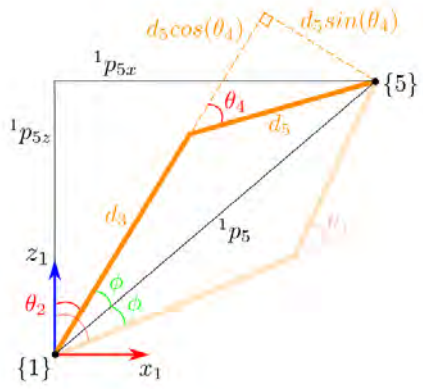

Figure 2.4: Calculation of angles θ_2, θ_4

$$ {}^0\mathbf{p}_5 = {}^0T_4\,{}^4\mathbf{p}_5 = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \quad (2.2.1.2) $$

$$ \theta_1 = \begin{cases} atan2\,(p_y, p_x) \\ \pi - atan2\,(p_y, p_x) \end{cases} \quad (2.2.1.3) $$

$$ \varphi = acos\left(\frac{d_3^2 + \|{}^1p_5\|^2 - d_5^2}{2d_3\|{}^1p_5\|}\right),\ \theta_2 = atan2\left(\sqrt{p_x^2 + p_y^2}, {}^1p_{5z}\right) \pm \varphi \quad (2.2.1.4) $$

$$ c_4 = \frac{\|{}^1p_5\|^2 - d_3^2 - d_5^2}{2d_3d_5},\ \theta_4 = atan2\left(\pm\sqrt{1 - c_4^2}, c_4\right) \quad (2.2.1.5) $$

Having computed $\theta_i,\ i = 1, \ldots, 4$, the orientation matrix of the wrist can be calculated as following

$$\begin{aligned} R_{target} &= \begin{bmatrix} i_x & j_x & k_x \\ i_y & j_y & k_y \\ i_z & j_z & k_z \end{bmatrix} \\ \theta_6 &= atan2\left(\pm\sqrt{1-k_y^2}, k_y\right) && (2.2.1.6) \\ \theta_7 &= atan2\left(-j_y, i_y\right) && (2.2.1.7) \\ \theta_5 &= atan2\left(-k_z, k_x\right) && (2.2.1.8) \end{aligned}$$

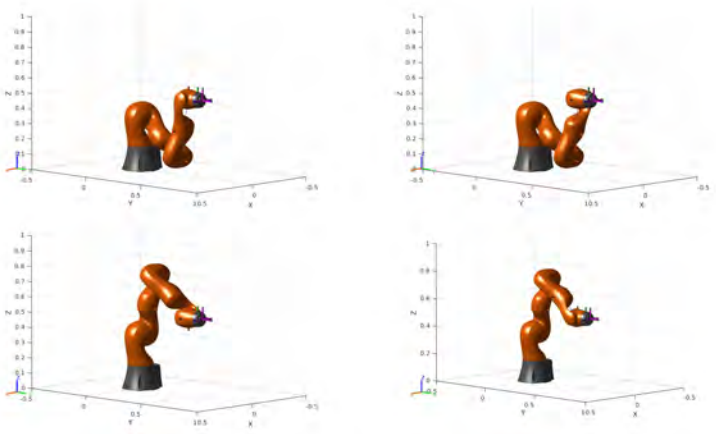

Figure 2.5: The first 4 out of 8 solutions of the IK-problem

2.2.2 Numerical solutions for 7 DoF robot arm

Tand the joint derivatives. These two vectors are related by a matrix known as the The **Jacobian** of the manipulator is

$$\dot{\mathbf{x}} = J(\mathbf{q})\dot{\mathbf{q}} = \begin{bmatrix} \frac{\partial \mathbf{f}_1(\mathbf{q})}{\partial q_1} & \cdots & \frac{\partial \mathbf{f}_1(\mathbf{q})}{\partial q_7} \\ \vdots & \ddots & \vdots \\ \frac{\partial \mathbf{f}_6(\mathbf{q})}{\partial q_1} & \cdots & \frac{\partial \mathbf{f}_6(\mathbf{q})}{\partial q_7} \end{bmatrix} \dot{\mathbf{q}}, \qquad (2.2.2.1)$$

where $\mathbf{x} \in \mathbb{R}^6$ is the position and orientation of the robot's end-effector and $\mathbf{q} = [\theta_1, \ldots, \theta_7]^\top \in \mathbb{R}^7$; in the sequel rather than using θ_i the adopted symbol q_i is selected. $J(\mathbf{q})$ is non rectangular owing to the additional 7th DoF and its pseudoinverse is computed as $J^\dagger = J^\top(JJ^\top)^{-1}$. Then a sequential algorithm is used to compute $\mathbf{q}$, as

Algorithm 1: Newton-Raphson numerical method for computing $\mathbf{q}$.

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 ε 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

The convergence to local minima needs attention and a recursive form that uses the previously computed solution to the 6 DoFs is used to provide $\theta_i,\ i \in \{1, \ldots, 7\}/3$. The columns of the Jacobian matrix are

$$J_i = \begin{bmatrix} {}^0\mathbf{z}_i \times ({}^0\mathbf{p}_8 - {}^0\mathbf{p}_i) \\ {}^0\mathbf{z}_i \end{bmatrix},\ i = 1, \ldots, 7. \tag{2.2.2.2}$$

2.2.3 Constraints & Singularity points

2.2.3.1 Workspace constraints & Singularity points

When studying the kinematics of a robotic arm it is important to study the singularity points, which must be known before path planning in order to be avoided. From a mathematical point of view, singularity points are defined as these spatial joint coordinates where the kinematics equations are not defined, e.g. when dividing by a quantity that becomes zero or is very close to zero or when the determinant of the robot's Jacobian approaches zero (as shown in Figure 5.2), or even when the IK solutions are infinite. Singularity points typically reduce the kinematic capability of the robot. An example of that are points where the end-effector cannot move at all or it cannot move in some directions. An other example of singularity points, are those in which small displacements/velocities in the taskspace require very large displacements/velocities in the joint space and sometimes with velocities much higher than the rated values of the actuators For the Kuka iiwa 7 DoF arm, the singularity points are:

- When $p_x^2 + p_y^2 = 0$ then the end-effector lies on the z-axis and θ_1 is not defined
- When the axes z_1, z_7 coincide, then θ_1, θ_7 can no longer be defined
- When $\sin(\theta_6) = 0$ then the angles θ_5, θ_7 are not defined. According to the robot's manual and the aforementioned equations of IK there are an infinite number of combinations of θ_5, θ_7 to get the same position on the end-effector.
- When the shoulder is fully extended, θ_1 and θ_3 are not defined (there are infinite combinations for the same result)

Other kinematic singularities that are explained in more details in KUKA's Sunrise.OS 1.11 manual are the following:

- Extended position $\theta_4 = 0$: motion is blocked in the direction of z_3 and z_5 axes
- When $\theta_4 = \frac{\pi}{2}$ and $\theta_6 = 0$ the motion parallel to z_6 or z_2 is blocked
- When $\theta_2 = 0$ and $\theta_3 = \pm\frac{\pi}{2}$ the the motion is blocked in the direction of the robot or parallel to axes z_2, z_5

- When $\theta_5 = \frac{\pi}{2}$ and $\theta_6 = 0$ the motion parallel to axis z_6 is blocked

The workspace of the Kuka iiwa LBR14 is shown in the sequel taking into account the allowable region of the joint coordinates

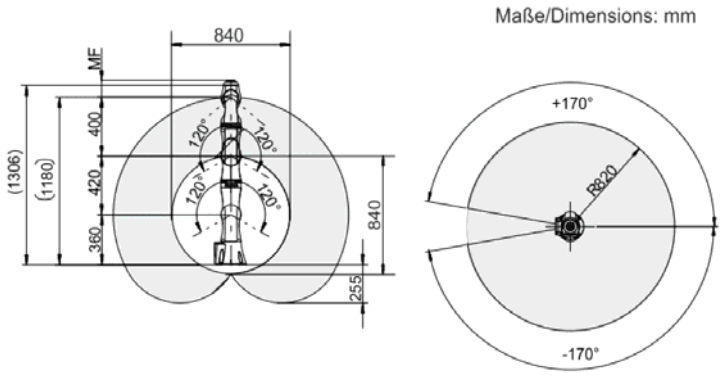

Figure 2.6: KUKA iiwa LBR14 workspace

2.2.3.2 Remote Center of Motion constraint

The Remote Center of Motion (RCM) constraint is encountered frequently in MIS. The RCM point, also known in the bibliography as fulcrum point, trocar point, incision point and/or insertion point, is a virtual point in space around which the robot is constrained to rotate (pivot). The RCM constraint means that at all times one point along the axis of the surgical tool must coincide with the RCM point. This constraint reduces the number of degrees of freedom from 6 (position and orientation in Cartesian 3D space) to only four:

1. **insertion** and **retraction**: translation along the $\hat{\mathbf{r}}$ vector of the spherical coordinate system of the Fulcrum Reference frame

2. **roll**, also known as pan: rotation around the $\hat{\mathbf{z}}$ unit vector by an angle ϕ

3. **pitch**, also known as tilt: rotation around the $\hat{\mathbf{y}}$ unit vector by an angle θ

4. **yaw**, also known as spin: rotation around the $\hat{\mathbf{x}}$ unit vector by an angle ψ

The RCM constraint does not have a direct implication in the solutions of the IK solution, because it only reduces the taskspace to a very specific subspace and a very specific set of robot positions and orientations.
To satisfy the RCM constraint many MIS robots have special mechanical structures that make it easier for the laparoscopic tools to pivot around the fulcrum point. The most common examples of these types of mechanisms are parallelograms, spherical linkages and circular tracking arcs, all of which have usually 2 RCM DoF. Other types of RCM mechanisms are isocenters, synchronous belt transmission, parallel manipulators, passive RCMs and compliant mechanisms [43].

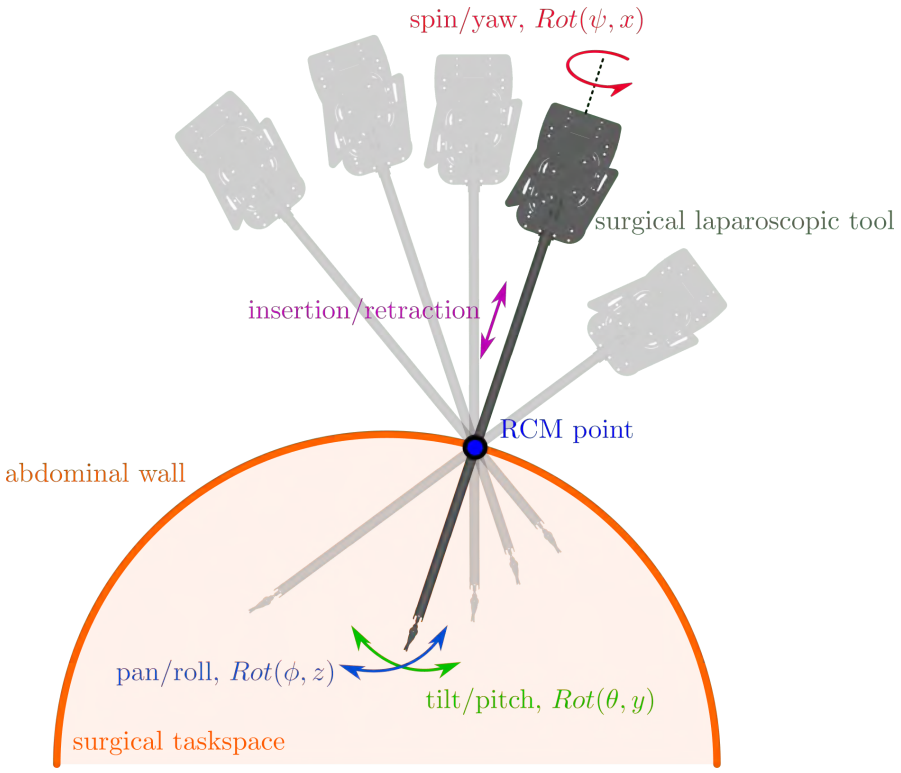

Figure 2.7: Pivoting motion of surgical laparoscopic tool around RCM point.

2.2.3.3 Elbow-up constraint

The IK typically provides 8 different solutions (see Figures 2.4 and 2.8). These solutions offer more flexibility to each case, and a different solution is chosen based on the conditions (e.g. previous robot pose) and requirements (e.g. collision avoidance). In this thesis, there are two distinct use cases in terms of choosing a specific robot configuration picking the surgical tools from one table, which does not impose any constraint and the insertion and pivoting of the surgical tool at the mounting dock, which imposes the RCM constraint (see 2.2.3.2) and a constraint to of collision avoidance between the robot arm and the mounting dock. In the latter use case, in order to satisfy the collision avoidance, all path planning kinematic solutions must be in a elbow-up configuration at all times.

Observing Figure 2.8, there are two ways to mathematically describe the elbow-up constraint (see Figure 2.9), either using the distance between the robot base and the third link or by using the relative angle of the base link and the third link. From the Kuka iiwa specifications, the following calculations are implemented using $d_1 = 360$mm and $d_3 = 420$mm.
The distance constraint is $d_{\min} \leq d \leq d_{\max}$, where $d_{\min} = \sqrt{d_1^2 + d_3^2} = 553$mm and $d_{\max} = d_1 + d_3 = 780$mm. The distance-based description of the elbow-up constraint is not very convenient, because it can not be easily forward-transformed to a description that uses the robot's forward kinematic transformations and reference frames, but it can be easily used after the IK solution to check which solutions satisfy this distance constraint. The angle-based description can sometimes be more convenient because it

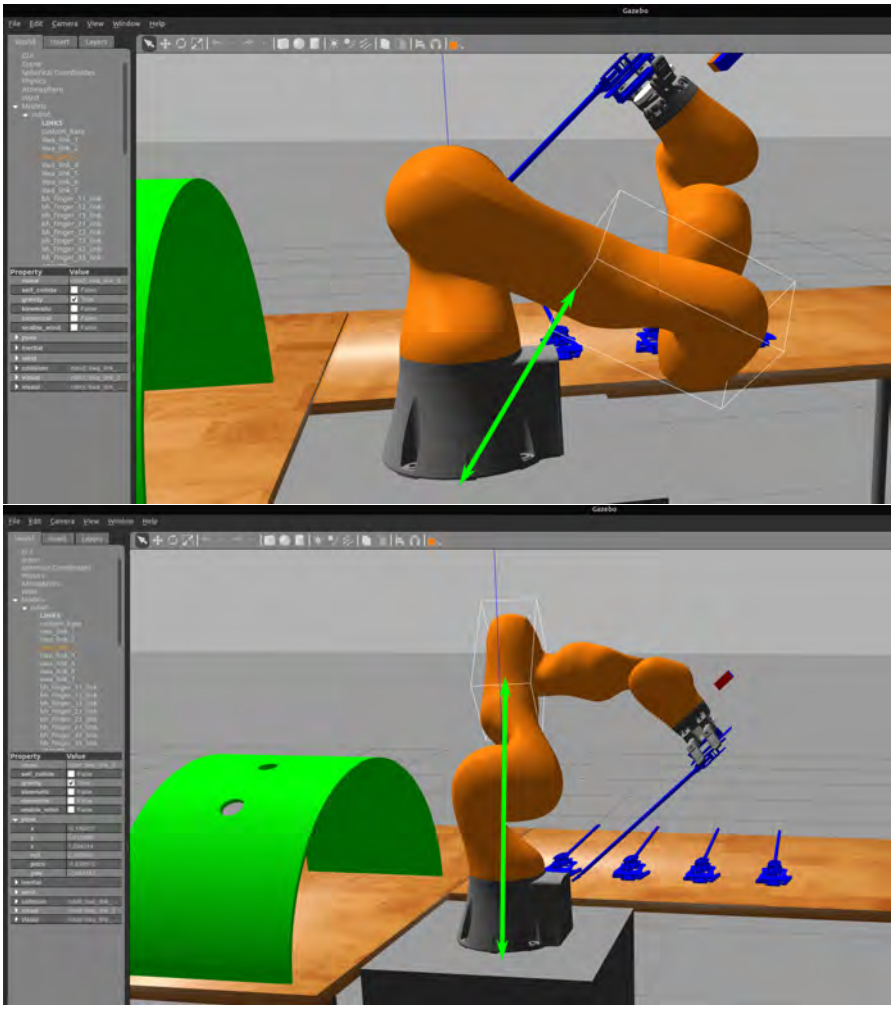

Figure 2.8: Top: elbow-down solution, bottom: elbow-up solution

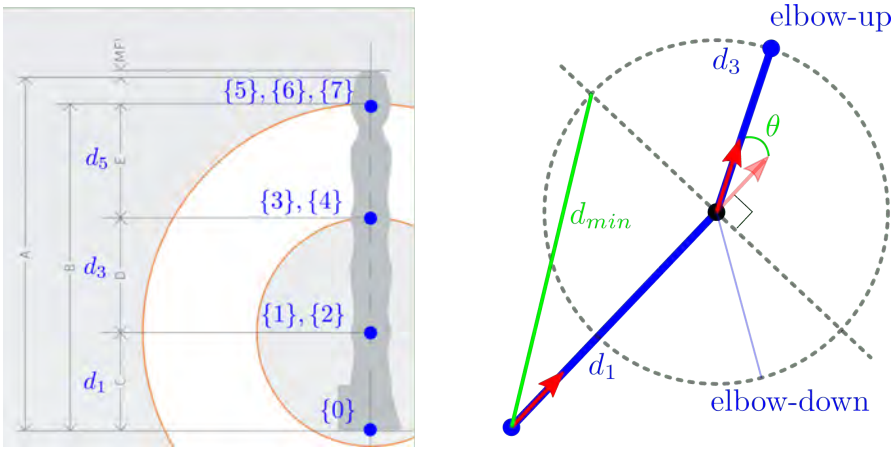

Figure 2.9: Elbow-up constraint description with relative pose between links with lengths d_1 and d_3

directly describes the orientation that the third reference frame must have, with respect to the reference frame of the base. Thus for each orientation angle (pitch, yaw, roll) the following constraint must be satisfied $-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2}$. When the IK problem is solved, then some of the solutions may be rejected from the elbow-up constraint after the following calculations. For each solution the forward kinematics up to the third joint must be calculated. Note that although the forward kinematics from the universal frame to the end-effector should be the same for all solutions, the same does not hold for the forward kinematic transformations of the intermediate links, since ${}^0T_3 = {}^UT_0^{-1}\ {}^UT_{tcp}\ {}^3T_{tcp}^{-1}$.

Let the calculated ${}^0T_3$ be ${}^0T_3 = \begin{bmatrix} {}^0R_3 & {}^0\mathbf{p}_3 \\ 0 & 1 \end{bmatrix}$, then the distance that must satisfy the inequality $d_{\min} \leq d \leq d_{\max}$ is

$$d = \|{}^0\mathbf{p}_3\| \tag{2.2.3.1}$$

In a similar way, the angle version of the elbow-up constraint can be used to check if a solution is accepted. Let the orientation matrix ${}^0R_3$ of the calculated pose ${}^0T_3$ have the following form

$${}^0R_3 = [{}^0\hat{\mathbf{x}}_3 \quad {}^0\hat{\mathbf{y}}_3 \quad {}^0\hat{\mathbf{z}}_3] \tag{2.2.3.2}$$

then the angle that must satisfy the inequality $-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2}$ is

$$\theta = acos\left(\frac{{}^0\hat{\mathbf{z}}_3 \cdot {}^0\hat{\mathbf{z}}}{\|{}^0\hat{\mathbf{z}}_3\|\|{}^0\hat{\mathbf{z}}\|}\right) \tag{2.2.3.3}$$

where ${}^0\hat{\mathbf{z}}$ is the unit vector of the z-axis of the frame $\{0\}$ with respect to itself, i.e. ${}^0\hat{\mathbf{z}} = [0, 0, 1]^\top$

Chapter 3

Grasping

A three finger gripper configuration is assumed to be placed at the end effector of the Kuka iiwa arm. This gripper corresponds to the Barrett Hand, shown in Figure 3.1.

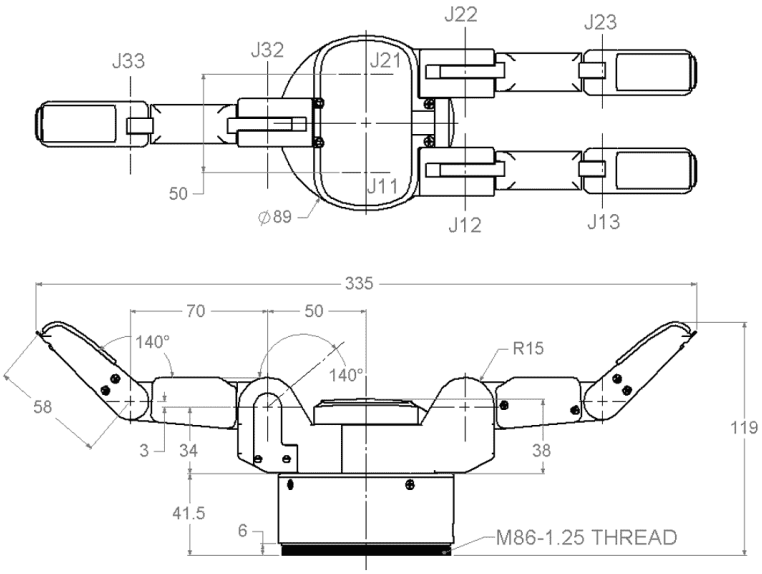

Figure 3.1: Barrett Hand gripper (model BH8-282) dimensions

3.1 Gripper & Forward Kinematics

The dimensions and angle parameters of each finger of this hand appear in Figure 3.2 and the following three tables.

i	θ_i (rad)	L_{i-1} (m)	d_i (m)	α_{i-1} (rad)
1 (J11)	$\theta_{11} - \pi/2$	0.025	0.0034	0
2 (J12)	$\theta_{12} + 0.04$	0.05	0	$\pi/2$
3 (J13)	$\theta_{13} + 0.69$	0.07	0	0

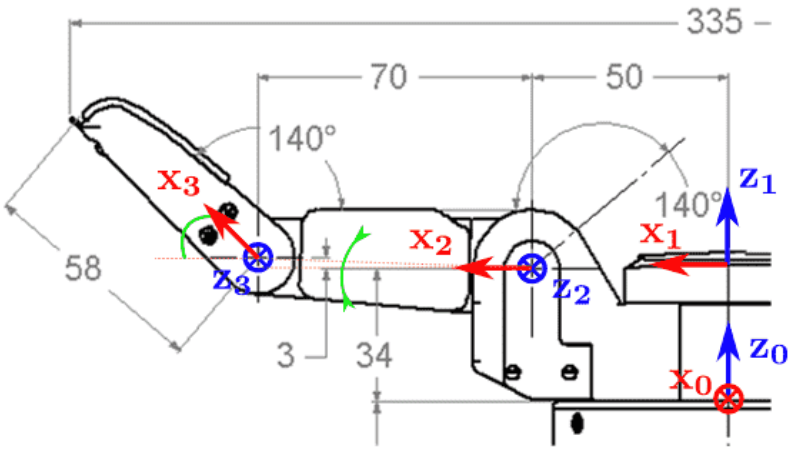

Figure 3.2: Coordinates systems for a generalized finger of the gripper

i	θ_i (rad)	L_{i-1} (m)	d_i (m)	α_{i-1} (rad)
1 (J21)	$\theta_{21} - \pi/2$	0.025	0.0034	0
2 (J22)	$\theta_{22} + 0.04$	0.05	0	$\pi/2$
3 (J23)	$\theta_{23} + 0.69$	0.07	0	0

i	θ_i (rad)	L_{i-1} (m)	d_i (m)	α_{i-1} (rad)
1	$\pi/2$	0	0.0034	0
2 (J32)	$\theta_{32} + 0.04$	0.05	0	$\pi/2$
3 (J33)	$\theta_{33} + 0.69$	0.07	0	0

3.2 Gripper Inverse Kinematics

The following IK analysis referes 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.

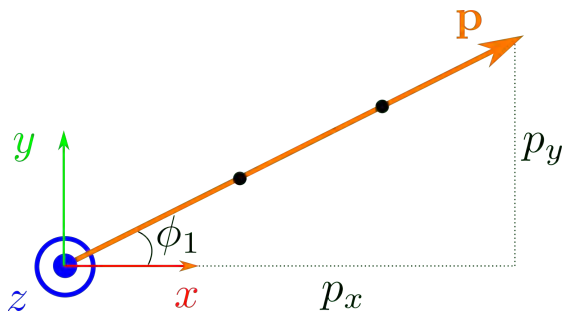

Figure 3.3: Top view of single finger of the gripper with 3 joints (RRR kinematic chain)

Let $\mathbf{p} = [p_x, p_y, p_z]^T$ be the position of the grasp point for one finger. The first angle can easily be calculated using the top view of Figure 3.3 from $\varphi_1 = atan2\,(p_y, p_x)$

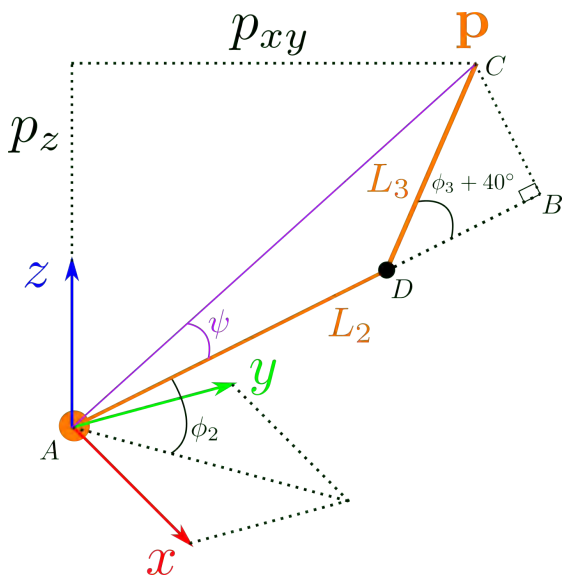

Figure 3.4: Side view of single finger of the gripper with 3 joints (RRR kinematic chain)

Geometric inspection of the noted quantities in Figure 3.4 provides

$$BD = L_3 \cos\left(\varphi_3 + \frac{2\pi}{9}\right),\ BC = L_3 \sin\left(\varphi_3 + \frac{2\pi}{9}\right),\ p_{xy} = \sqrt{p_x^2 + p_y^2} \tag{3.2.0.1}$$

Next, we calculate the third angle based on the law of cosines applied on the triangle ACD (see Figure 3.4)

$$\cos\left(\pi - \varphi_3 - \frac{2\pi}{9}\right) = \frac{L_2^2 + L_3^2 - p^2}{2L_2L_3},\ \cos\left(\varphi_3 + \frac{2\pi}{9}\right) = \frac{p^2 - L_2^2 - L_3^2}{2L_2L_3} \tag{3.2.0.2}$$

$$\varphi_3 = atan2\left[\pm\sqrt{1 - \left(\frac{p^2 - L_2^2 - L_3^2}{2L_2L_3}\right)^2}, \frac{p^2 - L_2^2 - L_3^2}{2L_2L_3}\right] - \frac{2\pi}{9} \tag{3.2.0.3}$$

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 not be negative. Having calculated φ_3 we calculate φ_2

$$tan\left(\psi + \varphi_2\right) = \frac{p_z}{\sqrt{p_x^2 + p_y^2}} \tag{3.2.0.4}$$

$$\psi + \varphi_2 = atan2\left(p_z, \sqrt{p_x^2 + p_y^2}\right) \tag{3.2.0.5}$$

$$tan\left(\psi\right) = \frac{L_3 \sin\left(\varphi_3 + \frac{\pi}{4}\right)}{L_2 + L_3 \cos\left(\varphi_3 + \frac{\pi}{4}\right)} \tag{3.2.0.6}$$

$$\varphi_2 = atan2\left(pz, \sqrt{p_x^2 + p_y^2}\right) - atan2\left[L_3 \sin\left(\varphi_3 + \frac{2\pi}{9}\right), L_2 + L_3 \cos\left(\varphi_3 + \frac{2\pi}{9}\right)\right] \tag{3.2.0.7}$$

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 used Gazebo simulator program, the information of the contact between a gripper's finger and an object is easily available as shown in listing 3.2. 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 [45].

Listing 3.1: ROS message with collision/contact information between one finger of the gripper and 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]
```

Listing 3.2: Example values from the Barrett Hand tactile array. Units N/cm^2 message type: bhand_controller/TactArray https://github.com/RobotnikAutomation/barrett_hand/tree/kinetic-devel/bhand_controller

```
header:
seq: 15277
stamp:
  secs: 1408685936
  nsecs: 928946018
frame_id: ""
finger1: [
    0.09000000357627869, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.019999999552965164, 0.009999999776482582,
    0.0, 0.009999999776482582, 0.0, 0.0, 0.019999999552965164,
    0.0, 0.03999999910593033, 0.0,
  ]
finger2: [
    0.0, 0.019999999552965164, 0.0, 0.03999999910593033, 0.029999999329447746,
    0.029999999329447746, 0.019999999552965164, 0.019999999552965164,
    0.029999999329447746, 0.019999999552965164, 0.019999999552965164,
    0.019999999552965164, 0.029999999329447746, 0.029999999329447746,
    0.07000000029802322, 0.09000000357627869, 0.0, 0.09000000357627869,
    0.009999999776482582, 0.019999999552965164, 0.05999999865889549,
    0.009999999776482582, 0.0, 0.03999999910593033,
  ]
finger3: [
    0.0, 0.0, 0.0, 0.0, 0.019999999552965164, 0.029999999329447746,
    0.0, 0.0, 0.019999999552965164, 0.0, 0.009999999776482582,
    0.029999999329447746, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.019999999552965164, 0.0, 0.0, 0.0,
  ]
palm: [
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
  ]
```

Chapter 4

Visual Perception

The detection and recognition of the surgical tools and other objects of the simulation scene is the topic of this chapter. For complexity reduction 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.

The used **Camera setup** has the following parameters:

- 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: 30 FpS

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. 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 $h = \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$. The filtered image is the result of the convolution of the image with the filter and is calculated as $g(i,j) = \sum_{k,l=-1}^{1} f(i+k, j+l)h(k,l)$, where $g(\cdot,\cdot)$ is the output image and $f(\cdot,\cdot)$ is the input image.
After the noise is removed the image is transformed into a binary one by setting a threshold, below which the pixels will be black and the rest will be white. The conversion to this binary format, simplifies the extraction of the boundaries of the black shapes, which will corresopnd to the boundaries of the objects of the initial image.

4.2 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

$$(\bar{x}, \bar{y}) = \left(\frac{1}{N} \sum_{i=1}^{N} x_i, \frac{1}{N} \sum_{i=1}^{N} y_i \right) \tag{4.2.0.1}$$

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 it's area, this method is not very accurate. For a more accurate value for the center of mass, the pixels that are inside the detected object's contour are used. The simplest way 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. Pixel subsampling can be used to speed up the execution.

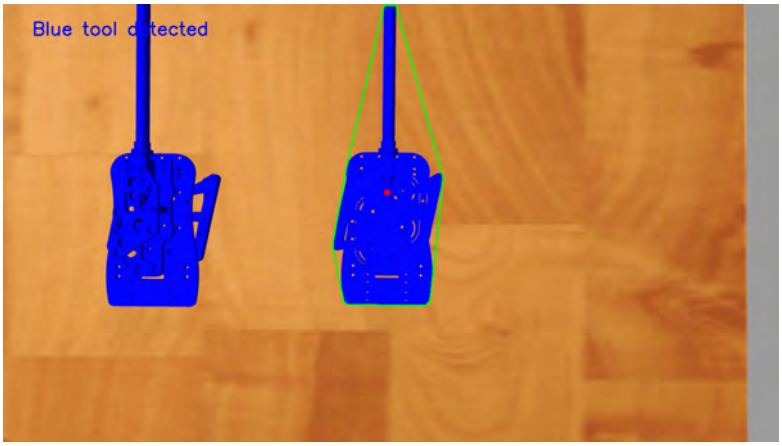

Figure 4.1: Simple tool detection in simulation based on color, using OpenCV. The green polygon is the convex hull, and the red point is the estimated center of mass

This approach can be further optimized by accounting only those pixels inside the **Region of Interest** (ROI) of the tool. ROI is a rectangular bounding box that fits exactly an object to be studied. 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 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 $\mathbf{Cv} = \lambda\mathbf{v}$, where $\mathbf{C}$ is the covariance matrix given by

$$\mathbf{C} = \begin{bmatrix} \sigma(x,x) & \sigma(x,y) \\ \sigma(y,x) & \sigma(y,y) \end{bmatrix}, \text{ and } \sigma(x,y) = \frac{1}{n-1}\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y}). \tag{4.2.0.2}$$

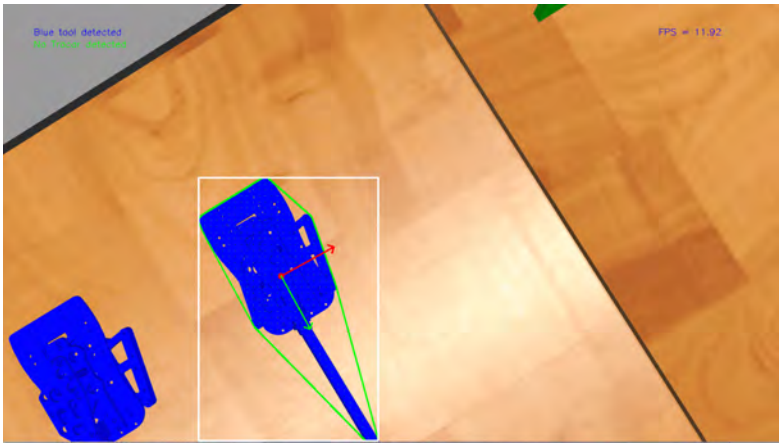

Figure 4.2: Estimation of tool's pose (position and orientation). The red dot is the center of mass and attached to that are the two orientation vectors of the tool. The green polygon is the convex hull of the tool and the white rectangle is its ROI.

4.3 Calculation of grasping points

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 preferably with force closure. The adopted method is by using an expanding **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 three intersections are found owing to the three finger-configuration of the hand.
the set of intersection points $\mathbb{G}$ is calculated from

$$\mathbb{G} = \arg\max_{(x,y)} I_1(x,y) \odot I_2(x,y). \tag{4.3.0.1}$$

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.

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

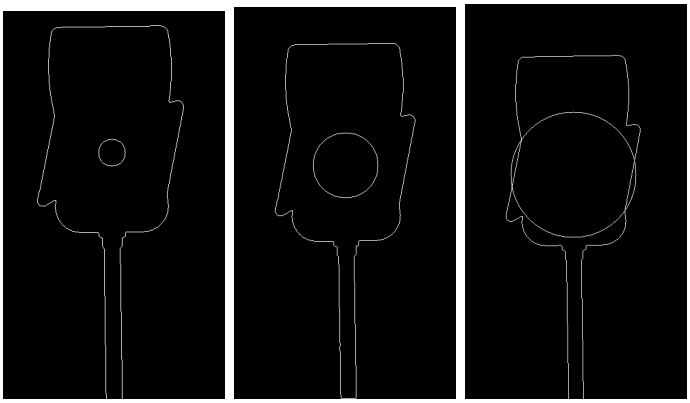

Figure 4.3: Finding candidate grasping points from the intersections of a growing circle and the contour of the detected surgical tool

4.4 Trocar detection & Estimation of fulcrum point

The detection of the trocar is accomplished using a similar simple trivial detection algorithm as the one used in Section 4.1. The fulcrum point is assumed to be at the trocar's (cylindrical shape) center.

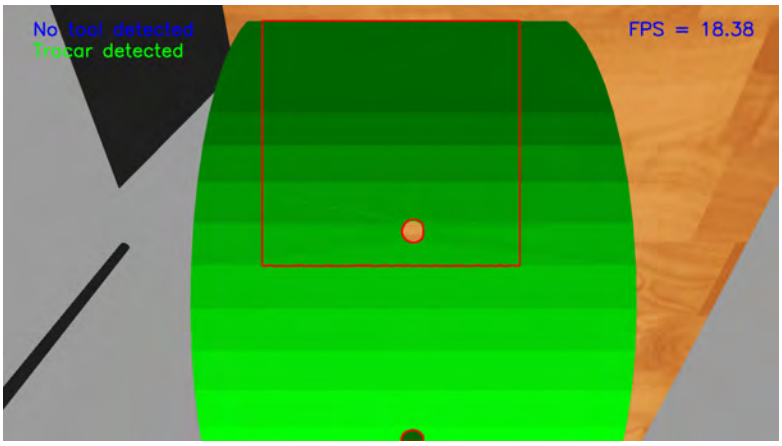

Figure 4.4: Simple color-based trocar using OpenCV.

Chapter 5

Path Planning

Path Planning is a geometric problem, related to finding a path from a starting point to a goal point while 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 Path planning sampling methods

Sampling planning methods use random functions to choose a sample from the configuration or joint space. Unlike discretizing methods, sampling ones are computationally inexpensive cannot compute optimal solutions.

5.1.1 RRT Algorithm

The **Rapidly-exploring Random Trees** algorithm searches for an obstacle-free motion plan from an initial state x_{init} to a set of goal states $\mathcal{X}_{goal}$ that are within the allowed position and orientation tolerances.

Algorithm 2: RRT Algorithm

```
foreach replanning attempt do
    initialize vertices V ← {x_init};
    initialize edges E ← ∅;
    initialize search tree T ← (V, E);
    while time ≤ maxPlanningTime do
        x_rand ← getSampleStateFrom(X);
        x_nearest ← getNearestNodeInTreeToState(T, x_rand);
        x_new ← findLocalPlanFromTo(x_nearest, x_rand);
        if isPathCollisionFree(x_nearest, x_rand) then
            V ← V ∪ {x_new};
            E ← E ∪ {(x_nearest, x_rand)};
            if x_new ∈ 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: a) **TRRT** Transition-based RRT, b) **BiTRRT**

Bidirectional Transition-based RRT, c) **RRT***

RRTConnect with is the default OMPL path planner in ROS, d) **LBTRRT** Lower Bound Tree RRT.

5.1.2 PRM Algorithms

The **Probabilistic Roadmap** (PRM) algorithm 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. The sampling of the free configuration space 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 $E \leftarrow \varnothing$;
initialize roadmap graph $G \leftarrow (V, E)$;
for $i = 1, \ldots, n$ **do**
  $x_{rand,i} \leftarrow$ getSampleStateFrom($\mathcal{X}$);
  $\mathcal{N}(x_{rand,i}) \leftarrow$ 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 *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: i) **PRM***, ii) **LazyPRM**, and iii) **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

Picking and placing small objects results in a straightforward path planning. When the object to be picked has at least one dimension significantly larger than the others, like a rod such similar to the surgical tools, then the path planning becomes more complicated, because of the possible

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

When the robot arm is in the pick-and-place phase where it detects the surgical tool and grasps and then move to the mounting dock to insert it, then at this phase, the taskspace is the same as the robot's taskspace. However, when the surgical tool is inserted in the patient's body and starts executing pivoting motions, then there are two taskspaces tto be studied. The first one is the surgical taskspace, which is where the surgical movements are planned (sutures, laparoscopic camera movements etc.) and the second taskspace is the one outside of patient's body in which the planned paths are "mirrored" so that the robot can execute them. The surgical taskspace $\mathbb{S}$ is the one shown in Figure 5.3. The paths inside the surgical task are transformed via the Fulcrum effect transformation, described in more detail in Section 6.1, to the pivot paths taskspace $\mathbb{P}$ which is a subset of the robot's taskspace $\mathbb{T}$ (we assume that all pivot motions can be executed, i.e. that all motions are reachable and fully inside the robot's taskspace). The paths are then used as input to a trajectory generator whose output is transformed via the IK equations to the joint angles that belong to the joint space $\mathbb{J}$.
Alternatively in a mathematical notation, the fulcrum effect transformation is $\Phi : \mathbb{S} \longrightarrow \mathbb{P}$, while the IK transformation is $IK : \mathbb{P} \subset \mathbb{T} \longrightarrow \mathbb{J}$ and similarly the forward kinematics transformation can be described as $FK : \mathbb{J} \longrightarrow \mathbb{T}$

Figure 5.1: Relationships beteeen Surgical and pivot path taskspace.

The Dexterity metric of the tool's task space is

$$\mathcal{D} = \mathcal{L}_q \mathcal{M}, \text{where} \tag{5.3.0.1}$$

$$\mathcal{M} = \sqrt{det(JJ^\top)} \tag{5.3.0.2}$$

$$\mathcal{L}_q = 1 - \exp\left\{-\kappa \prod_{i=1}^{n_k} \frac{(q_i - q_{i,\min})(q_{i,\max} - q_i)}{(q_{i,\max} - q_{i,\min})^2}\right\} \tag{5.3.0.3}$$

Equation (5.3.0.3) calculates a joint limit measure which is multiplied with the manipulability measure and gives the dexterity measure and the following: a) 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, implying that the robot has **no dexterity at the boundary of the task space**, b) If the value of q_i is close to its boundary value then the dexterity approaches zero. The distance from the boundary (or in other words how fast the exponential term converges) depends on the parameter κ, and c) The q_{min}, q_{max} are calculated from the geometry of the task-space.
For maximum dexterity throughout the 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 (5.3.0.3) is difficult and time-consuming at task spaces with more intricate geometries. A similar and more practical equation to (5.3.0.3) can be written for calculating the dexterity of the robot in task space:

$$\mathcal{L}_p = 1 - e^{-\kappa(r_{max} - r)}, \tag{5.3.0.4}$$

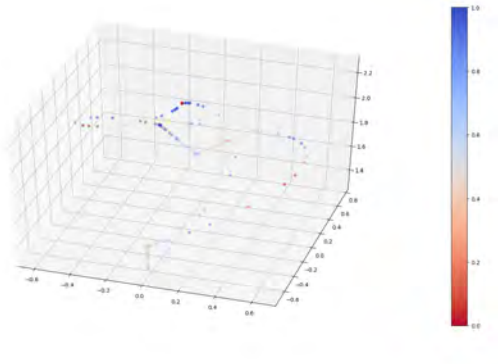

Figure 5.2: Manipulability index of robot arm during trajectory

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, $L^2 = r^2 + z^2$. Equation (5.3.0.4) 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 (5.3.0.1).

```
x = []; y = []; z = []; r = [];
s = 0:0.005:0.5; L = 0.48; k = 4.5;
for z0=-L:0.01:0
    if abs(z0)*sqrt(2) <= L
        rmax = abs(z0);
    else
        rmax = sqrt(L^2-z0^2);
    end
    for r0=0:0.02:rmax
        x = [x, r0*\cos(2*pi*s)];
        y = [y, r0*\sin(2*pi*s)];
        z = [z, z0*ones(size(s))];
        lp = 1-exp(-k*(rmax-r0));
        r = [r, lp*ones(size(s))];
    end
end
```

Listing 1: RCM Taskpace calculation in MATLAB

All of the metrics above, measure how much reachable are various points of the surgical taskspace, but they are calculated from different perspectives, using input values from different spaces.

$$\mathcal{L}_q, \mathcal{M} : \mathbb{J} \longrightarrow \mathbb{R} \quad \text{and} \quad \mathcal{L}_p : \mathbb{S} \longrightarrow \mathbb{R} \tag{5.3.0.5}$$

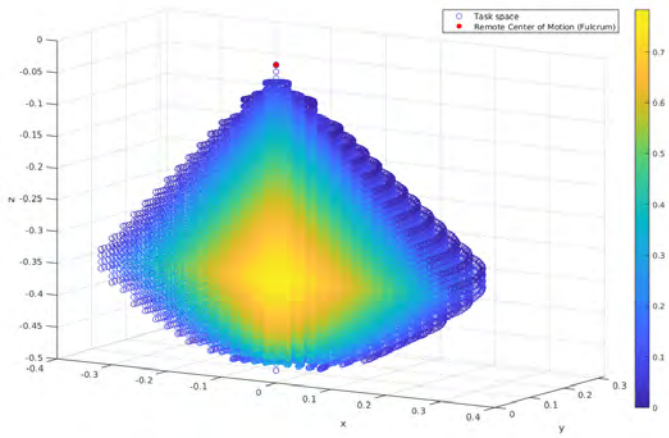

Figure 5.3: Task space inside patient's body. Colors with 0 or low value correspond to points with low dexterity.

Chapter 6

Trajectory Planning - Laparoscopic tool manipulation

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

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

$$ {}^{F}T_{B} = \begin{bmatrix} {}^{F}R_{B} & {}^{F}\mathbf{p}_{B} \\ \mathbf{0} & 1 \end{bmatrix} \quad where \quad {}^{F}R_{B} = \begin{bmatrix} \hat{\mathbf{x}}_{B} & \hat{\mathbf{y}}_{B} & \hat{\mathbf{z}}_{B} \end{bmatrix} $$

$$ \hat{\mathbf{x}}_{B} = \hat{\varphi} = -\sin(\varphi)\hat{\mathbf{x}}_{F} + \cos(\varphi)\hat{\mathbf{y}}_{F} = \begin{bmatrix} -\sin(\varphi) \\ \cos(\varphi) \\ 0 \end{bmatrix} \tag{6.1.0.1} $$

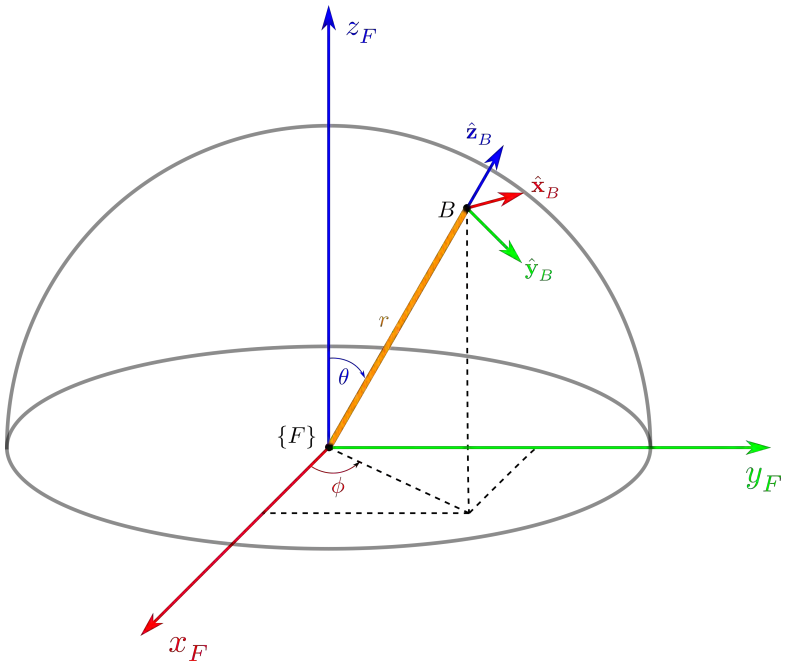

Figure 6.1: Tool pose at target point B calculated with respect to Fulcrum's reference frame $\{F\}$

$$\hat{\mathbf{y}}_B = \hat{\theta} = \cos(\theta)\cos(\varphi)\hat{\mathbf{x}}_F + \cos(\theta)\sin(\varphi)\hat{\mathbf{y}}_F - \sin(\theta)\hat{\mathbf{z}}_F = \begin{bmatrix} \cos(\theta)\cos(\varphi) \\ \cos(\theta)\sin(\varphi) \\ -\sin(\theta) \end{bmatrix} \tag{6.1.0.2}$$

$$\hat{\mathbf{z}}_B = \hat{\mathbf{r}} = \sin(\theta)\cos(\varphi)\hat{\mathbf{x}}_F + \sin(\theta)\sin(\varphi)\hat{\mathbf{y}}_F + \cos(\theta)\hat{\mathbf{z}}_F = \begin{bmatrix} \sin(\theta)\cos(\varphi) \\ \sin(\theta)\sin(\varphi) \\ \cos(\theta) \end{bmatrix} \tag{6.1.0.3}$$

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

$${}^F\mathbf{p}_B = \begin{bmatrix} \rho\sin(\beta)\cos(\alpha) \\ \rho\sin(\beta)\sin(\alpha) \\ \rho\cos(\beta) \end{bmatrix} = \rho\hat{\mathbf{r}} \tag{6.1.0.4}$$

After having calculated the $\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}}, \mathbf{p}$, then these are rotated according to how the tool is attached on the end-effector (e.g. in this thesis the $\hat{\mathbf{x}}$ vector points towards the origin of the frame, see figure 9.6) of the robot and they are also translated (see also figure 9.6) based on the offset between the end-effector and the tool (distance between end-effector and TCP). Note, that one can also use the φ, θ angles as two of the Euler rotation angles and thus calculate the orientation with these angles instead of using the spherical coordinate unit vectors method above. The final transformed goal pose 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.

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

$$
\begin{aligned}
{}^{U}T_0\ {}^{0}T_7\ {}^{7}T_{TCP} &= {}^{U}T_F\ {}^{F}T_B \\
{}^{0}T_7 &= {}^{U}T_0^{-1}\ {}^{U}T_F\ {}^{F}T_B\ {}^{7}T_{TCP}^{-1}
\end{aligned} \tag{6.1.0.5}
$$

```
function tcp = fulcrumEffectTrajectory(P, L)
    tcp = zeros(size(P));
    for i=1:size(P)
        px = P(i,1); py = P(i,2); pz = P(i,3);
        r = sqrt(px^2+py^2+pz^2);
        th = atan2(sqrt(px^2+py^2), pz);
        phi = atan2(py, px);
        vx = [cos(th)*cos(phi); cos(th)*sin(phi); -sin(th)];
        vy = [-sin(phi); cos(phi); 0];
        vz = [-sin(th)*cos(phi); -sin(th)*sin(phi); cos(th)];
        vp = r*[sin(th)*cos(phi); sin(th)*sin(phi); cos(th)];
        T = zeros(4,4);
        R = [vx, vy, vz];
        T(1:3,1:3) = R;
        T(1:3,4) = vp;
        T(4,4) = 1;
        Td = eye(4);
        Td(1:3,4) = (r-L)/r*vp;
        tcp_point = Td*inv(T)*[P(i,:).'; 1];
        tcp(i,:) = tcp_point(1:3).';
    end
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 (see figure 6.2).

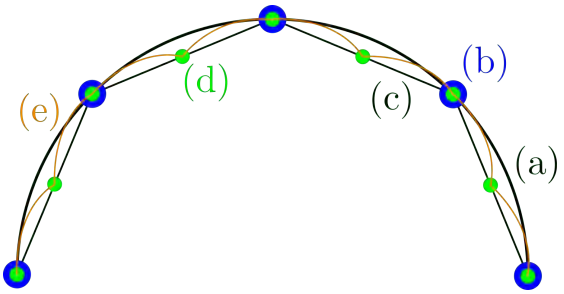

Figure 6.2: Path/trajectory interpolation done at two stages. (a) the initial desired circular trajectory, (b) sample waypoints that will approximate the circle and shape a polygon, (c) the line segments connecting the waypoints and which the robot must theoretically follow exactly, (d) lower-level interpolation of the line segments (made by MoveIt in ROS or by the robot's controller when executing PtP), (e) the actual trajectory that the robot will follow

6.2.1 Circular trajectory of tool tip

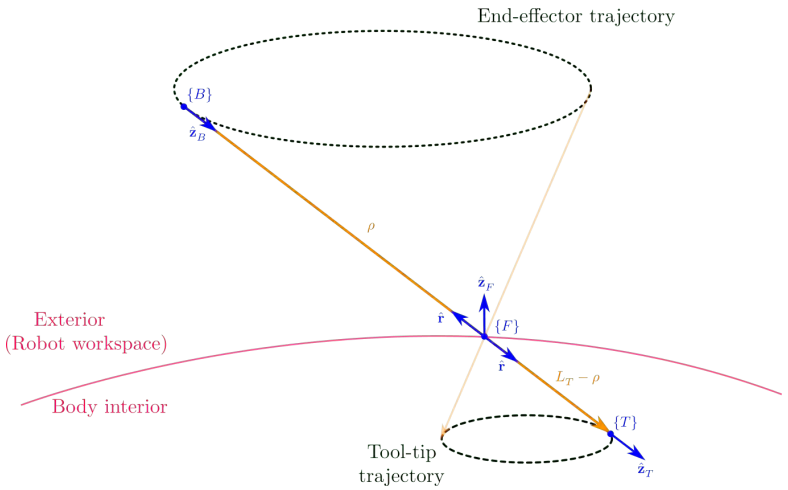

Figure 6.3: Circular trajectory of tool tip with respect to Fulcrum reference frame

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, where the 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.

$$(x_F - x_{F0})^2 + (y_F - y_{F0})^2 = r_0^2, \quad z_F = z_{F0} \tag{6.2.1.1}$$

It is more convenient to express trajectories in a parametric form,

$$\begin{cases} x_F = r_0 \cos(2\pi s) + x_{F0} \\ y_F = r_0 \sin(2\pi s) + y_{F0} \\ z_F = z_{F0} \end{cases}, \quad s \in [0, 1] \tag{6.2.1.2}$$

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

$$\begin{cases} r = \sqrt{x_F^2 + y_F^2 + z_F^2} \\ \theta = atan2\left(\sqrt{x_F^2 + y_F^2}, z_F\right) \\ \varphi = atan2(y_F, x_F) \end{cases} \tag{6.2.1.3}$$

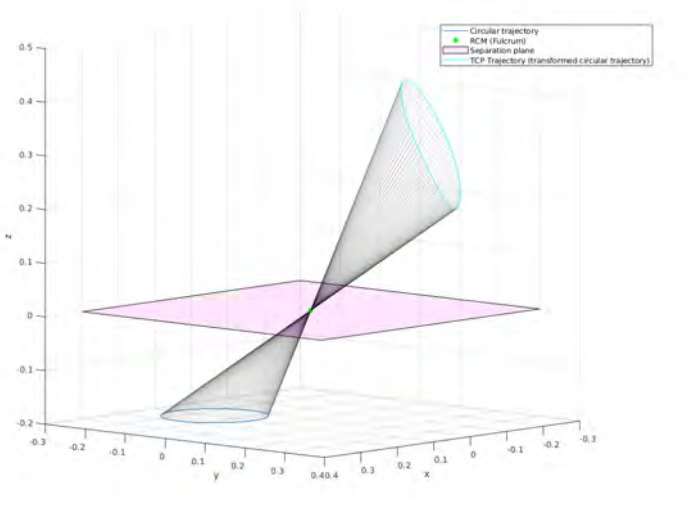

Figure 6.4: Circular trajectory of tool tip with respect to Fulcrum reference frame and it's transformation via the Fulcrum Effect

Although the equations 6.2.1.2 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 6.2.1.2 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.

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 that 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:

$$\begin{cases} x_F = r_0 \cos(2\pi s + \varphi_0) + x_{F0} \\ y_F = r_0 \sin(2\pi s + \varphi_0) + y_{F0} \\ z_F = z_{F0} \end{cases} \quad , \quad s \in \left[0, \frac{\varphi}{2\pi}\right] \tag{6.2.2.1}$$

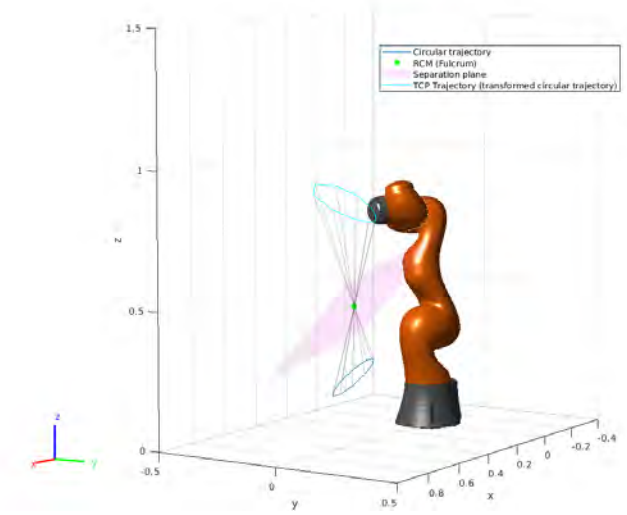

Figure 6.5: Circular trajectory that lies on an a plane of arbitrary orientation with respect to the fulcrum point

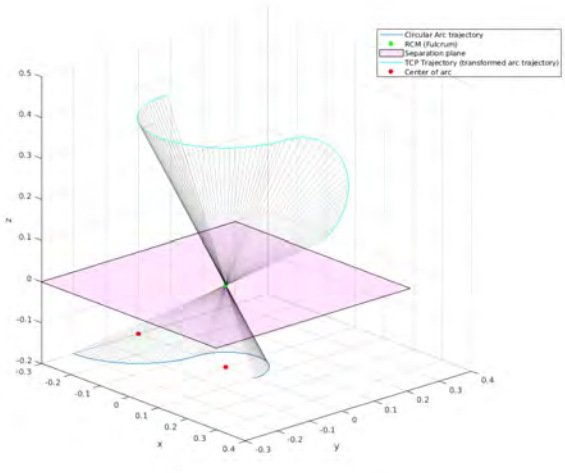

Figure 6.6: Circular arc trajectory of tool tip with respect to Fulcrum reference frame and it's transformation via the Fulcrum Effect. In this trajectory 2 circular arcs are used

6.2.3 Helical trajectory of tool tip

The helical trajectory is another useful trajectory that was studied in this thesis. The importance of this trajectory lies in the fact that the helical movement of the surgical tool can be considered as an ideal approximation of a suturing trajectory, a very common task in surgery which is extensively studied and researched in surgery robotics. A helical trajectory can be expressed by the following parametric equations:

$$\begin{cases} x_F = r_0 \cos(2\pi s) + x_{F0} \\ y_F = r_0 \sin(2\pi s) + y_{F0} \\ z_F = \pm \beta s \end{cases} \tag{6.2.3.1}$$

where $s \in [0, \tau]$ expresses the range along the trajectory, τ the cycles, and β / r_0 is the slope (or also known as the pitch which is given by $2\pi\beta$)

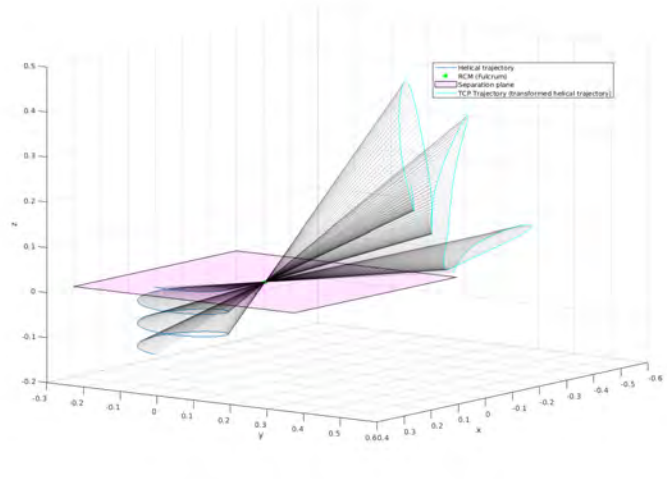

Figure 6.7: Helical trajectory of tool tip with respect to Fulcrum reference frame and it's transformation via the Fulcrum Effect

6.2.4 Line segment trajectory of tool tip

$$\mathbf{d} = {}^F\mathbf{p}_{T2} - {}^F\mathbf{p}_{T1} = [l, m, n]^\top$$

$${}^F\mathbf{p}_T = [x_F, y_F, z_F]^\top$$

$${}^F\mathbf{p}_T = {}^F\mathbf{p}_{T1} + s\mathbf{d}$$

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

$$\begin{cases} x_F = sl + x_{F1} = (1-s)x_{F1} + sx_{F2} \\ y_F = sm + y_{F1} = (1-s)y_{F1} + sy_{F2} \\ z_F = sn + z_{F1} = (1-s)z_{F1} + sz_{F2} \end{cases} \tag{6.2.4.1}$$

After having calculated the cartesian coordinates we can calculate the spherical coordinates using the 6.2.1.3 equations.

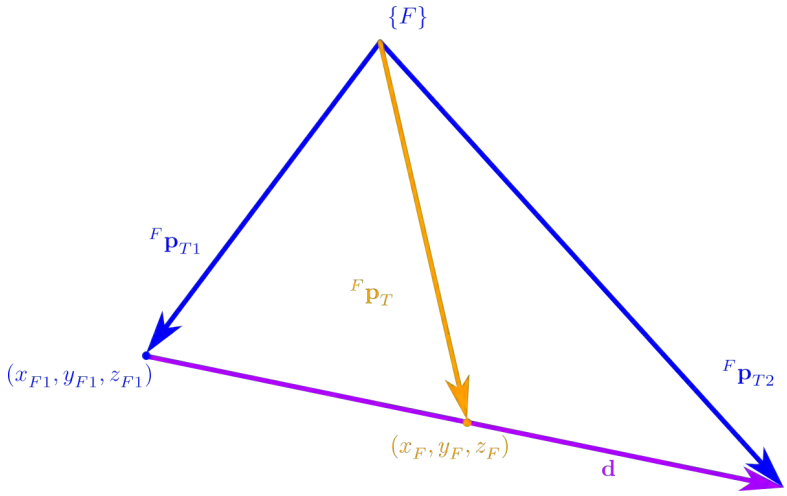

Figure 6.8: Line segment trajectory of tool tip with respect to Fulcrum reference frame

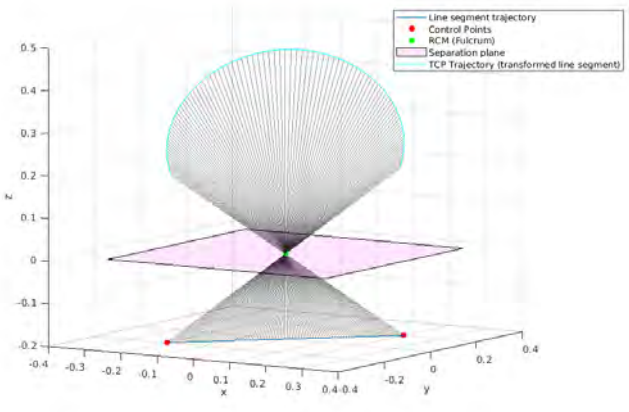

Figure 6.9: A Line segment trajectory and its transformation via the Fulcrum Effect

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.

6.2.5 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 $\{\mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P}_n\}$ 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

$$x_i(s) = a_i(s - s_i)^3 + b_i(s - s_i)^2 + c_i(s - s_i) + d_i, \qquad s_i \leqslant s \leqslant s_{i+1} \tag{6.2.5.1}$$

The polynomial in equation 6.2.5.1 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.

$$x_i(s_i) = x_i \tag{6.2.5.2}$$

$$x_i(s_{i+1}) = x_{i+1} \tag{6.2.5.3}$$

$$\dot{x}_i(s_i) = \dot{x}_i \tag{6.2.5.4}$$

$$\dot{x}_i(s_{i+1}) = \dot{x}_{i+1} \tag{6.2.5.5}$$

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

$$d_i = x_i(s_i) = x_i \tag{6.2.5.6}$$

and by taking the derivative of 6.2.5.1, we can calculate c_i

$$\dot{x}_i(s_i) = 3a_i(s - s_i)^2 + 2b_i(s - s_i) + c_i \tag{6.2.5.7}$$

$$c_i = \dot{x}_i(s_i) = \dot{x}_i \tag{6.2.5.8}$$

By substituting $s = s_{i+1}$ in 6.2.5.1 and 6.2.5.7, by using equations 6.2.5.8, 6.2.5.6 and if we set $\sigma = s_{i+1} - s_i$ for brevity, we get the following two equations

$$x_{i+1} = x_i(s_{i+1}) = a_i\sigma^3 + b_i\sigma^2 + c_i\sigma + x_i \tag{6.2.5.9}$$

and

$$\dot{x}_{i+1} = \dot{x}_i(s_{i+1}) = 3a_i\sigma^2 + 2b_i\sigma + \dot{x}_i \tag{6.2.5.10}$$

By multiplying 6.2.5.10 by σ and 6.2.5.9 by -3 and add them together we get

$$\dot{x}_{i+1}\sigma - 3x_{i+1} = -b_i\sigma^2 - 2\dot{x}_i\sigma - 3x_i$$

$$b_i = \frac{1}{\sigma^2}(3x_{i+1} - 3x_i - \dot{x}_{i+1}\sigma - 2\dot{x}_i\sigma) \tag{6.2.5.11}$$

Similarly, by multiplying 6.2.5.10 by σ and 6.2.5.9 by -2 and add them together we get

$$\dot{x}_{i+1}\sigma - 2x_{i+1} = a_i\sigma^3 - \dot{x}_i\sigma - 2x_i$$

$$a_i = \frac{1}{\sigma^3}(\dot{x}_{i+1}\sigma - 2x_{i+1} + \dot{x}_i\sigma + 2x_i) \tag{6.2.5.12}$$

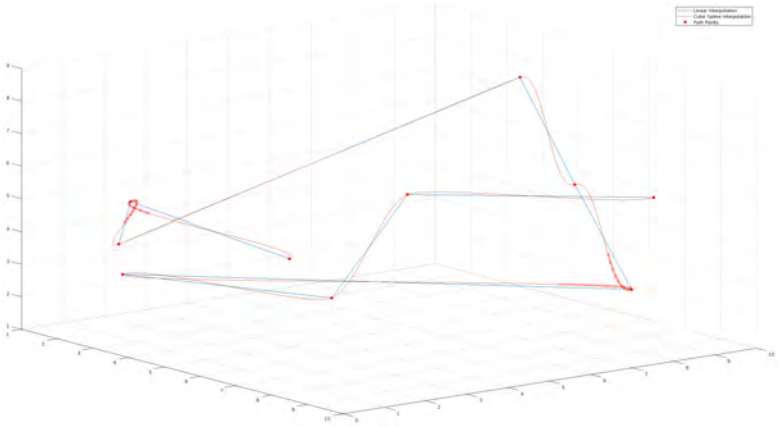

Figure 6.10: Cubic Spline curve with 10 waypoints

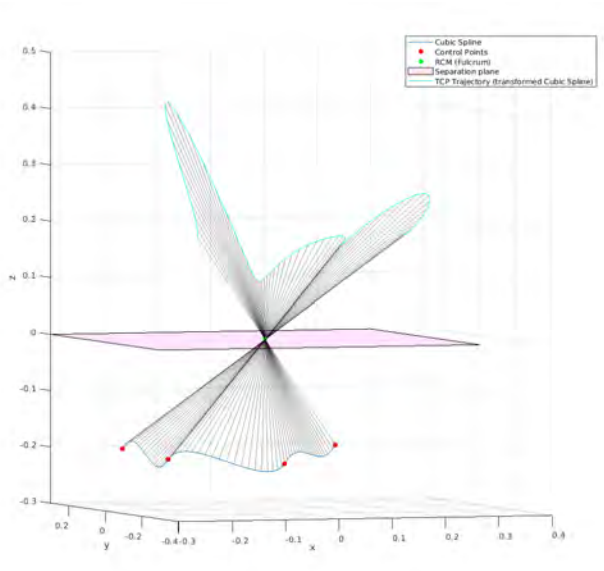

Figure 6.11: A Cubic Spline trajectory and it's transformation via the Fulcrum Effect

6.2.6 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.

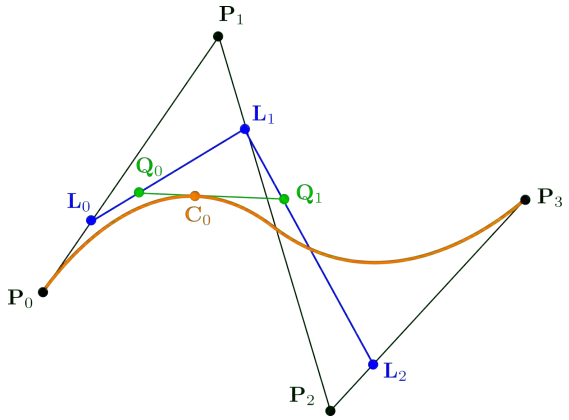

Figure 6.12: Cubic Bézier curve calculated using cubic interpolation of 4 control points

We first calculate the linear interpolation of the control points

$$\mathbf{L}_0(s) = (1-s)\mathbf{P}_0 + s\mathbf{P}_1 \tag{6.2.6.1}$$

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

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

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

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

$$\mathbf{Q}_0(s) = (1-s)^2\mathbf{P}_0 + 2(1-s)s\mathbf{P}_1 + s^2\mathbf{P}_2 \tag{6.2.6.2}$$

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

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

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

$$\mathbf{C}_0(s) = (1-s)^3\mathbf{P}_0 + 3(1-s)^2s\mathbf{P}_1 + 3(1-s)s^2\mathbf{P}_2 + s^3\mathbf{P}_3 \tag{6.2.6.3}$$

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

$$\mathbf{C}_0(s) = \begin{bmatrix}\mathbf{P}_0 & \mathbf{P}_1 & \mathbf{P}_2 & \mathbf{P}_3\end{bmatrix} \begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 3 & 0 & 0\\ 1 & 0 & 0 & 0\end{bmatrix} \begin{bmatrix}s^3\\ s^2\\ s\\ 1\end{bmatrix} \tag{6.2.6.4}$$

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.

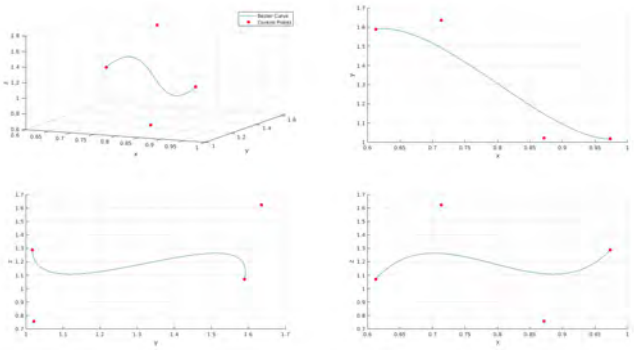

Figure 6.13: A cubic Bézier curve calculated and plotted in MATLAB

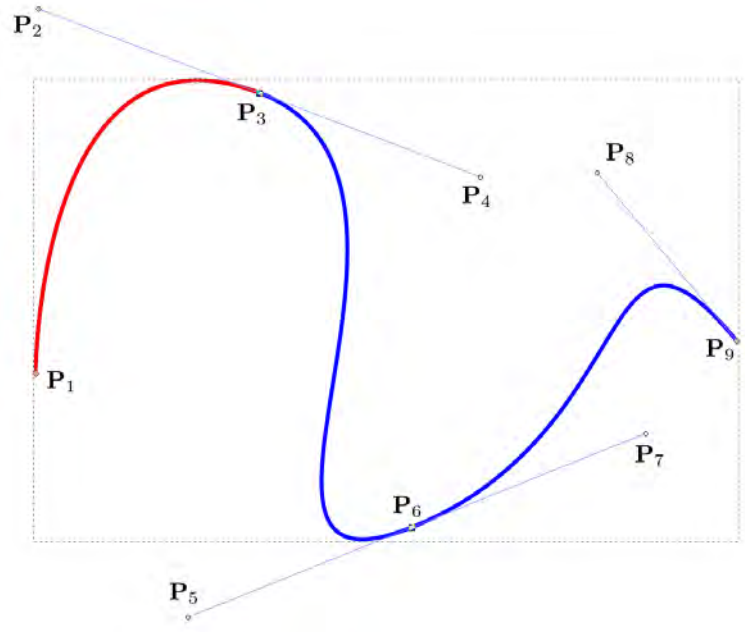

Figure 6.14: B-Spline curve constructed from 3 Bézier curves. The first Bézier curve colored in red is a quadratic one and the following two are both cubic.

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 6.14, 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 6.14 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 6.14, $\mathbf{P}_2, \mathbf{P}_3, \mathbf{P}_4$ 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. The robot arm may reach it's 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.

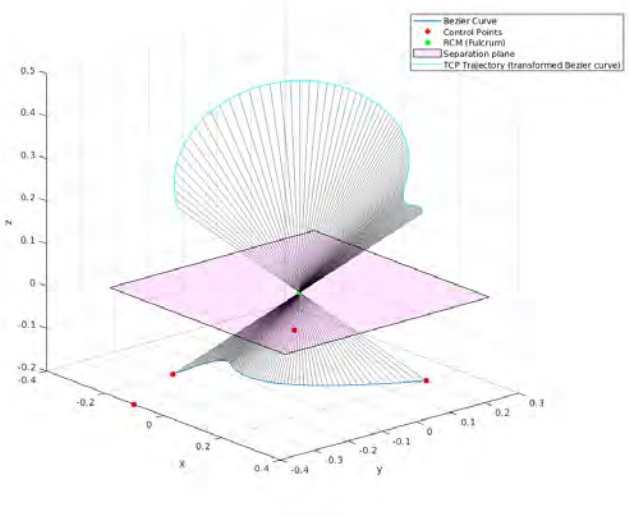

Figure 6.15: A Bézier curve trajectory and it's transformation via the Fulcrum Effect

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.

6.3.1 Polynomials of 5th order

Sometimes for a smooth trajectory between 2 consecutive path points, polynomials of higher degree are used. One such common family of polynomials are those of 5th degree with which we can specify the desired position, velocity, as well as acceleration at the beginning and at the end of the trajectory segment. This polynomial has the following closed form

$$q_i(t) = a_i + b_i t + c_i t^2 + d_i t^3 + e_i t^4 + f_i t^5 \tag{6.3.1.1}$$

where

$$q_i = q_i(t_i), \quad q_{i+1} = q_i(t_{i+1}) = q_{i+1}(t_{i+1}) \quad \text{and} \quad t_i \leq t \leq t_{i+1} \tag{6.3.1.2}$$

The constants $a_i, b_i, \cdots, f_i$ are computed from the boundary conditions, i.e. the position, and its derivatives, evaluated at the beginning and at the end of the trajectory segment.

$$\begin{aligned} q_i &= a_i \\ q_{i+1} &= a_i + b_i t_{i+1} + c_i t_{i+1}^2 + d_i t_{i+1}^3 + e_i t_{i+1}^4 + f_i t_{i+1}^5 \\ \dot{q}_i &= b_i \\ \dot{q}_{i+1} &= b_i + 2c_i t_{i+1} + 3d_i t_{i+1}^2 + 4e_i t_{i+1}^3 + 5f_i t_{i+1}^4 \\ \ddot{q}_i &= 2c_i \\ \ddot{q}_{i+1} &= 2c_i + 6d_i t_{i+1} + 12e_i t_{i+1}^2 + 20f_i t_{i+1}^3 \end{aligned} \tag{6.3.1.3}$$

Solving the system of boundary condition equations 6.3.1.3, the solution for the constants is the following

$$\begin{aligned} a_i &= q_i \\ b_i &= \dot{q}_i \\ c_i &= \frac{\ddot{q}_i}{2} \\ d_i &= \frac{20q_{i+1} - 20q_i - (8\dot{q}_{i+1} + 12\dot{q}_i)t_{i+1} - (3\ddot{q}_i - \ddot{q}_{i+1})t_{i+1}^2}{2t_{i+1}^3} \\ e_i &= \frac{30q_i - 30q_{i+1} - (14\dot{q}_{i+1} + 16\dot{q}_i)t_{i+1} + (3\ddot{q}_i - 2\ddot{q}_{i+1})t_{i+1}^2}{2t_{i+1}^4} \\ f_i &= \frac{12q_{i+1} - 12q_i - (6\dot{q}_{i+1} + 6\dot{q}_i)t_{i+1} - (\ddot{q}_i - \ddot{q}_{i+1})t_{i+1}^2}{2t_{i+1}^5} \end{aligned} \tag{6.3.1.4}$$

6.3.2 Planning with velocity profiles

6.3.2.1 Trapezoid velocity profile - LSPB

Trapezoid velocity profiles are a family of trajectories in motion control, that combines (blends) a linear segment with two parabolic segments. This type of trajectory is also known in the bibliography as Linear Segments with Parabolic Blends or LSPB. The equations of motions are studied in 3 intervals $[t_i, t_a], [t_a, t_d]$ and $[t_d, t_{i+1}]$ where $t_a = t_i + \tau$ and $t_d = t_{i+1} - \tau$ where τ is the time duration for both the acceleration and deceleration phases (symmetric trapezoid profile).

For $t_i \leq t \leq t_a$ the equations of the trajectory are the following:

$$q_i(t) = c_0 + c_1 t + c_2 t^2, \quad c_0, c_1, c_2 \in \mathbb{R} \tag{6.3.2.1}$$

$$\dot{q}_i(t) = c_1 + 2c_2 t \tag{6.3.2.2}$$

Using the initial conditions $q_i(0) = q_i$, $\dot{q}_i(0) = 0$ and $\dot{q}_i(t_a) = \dot{q}_{ic}$ we can solve for the constants c_i. $\dot{q}_{ic}$ is the desired constant velocity for the linear segment.

$$c_0 = q_i \quad \text{and} \quad c_1 = 0 \tag{6.3.2.3}$$

$$\dot{q}_i + 2c_2 t_a = \dot{q}_{ic}$$

$$c_2 = \frac{\dot{q}_{ic}}{2t_a} \tag{6.3.2.4}$$

The trajectory solution for $t_i \leq t \leq t_a$ is

$$\begin{aligned} q_i(t) &= q_i + \frac{\dot{q}_{ic}}{2t_a} t^2 \\ \dot{q}_i(t) &= \frac{\dot{q}_{ic}}{t_a} t \\ \ddot{q}_i(t) &= \frac{\dot{q}_{ic}}{t_a} \end{aligned} \tag{6.3.2.5}$$

For $t_a \leq t \leq t_d$ the equations of the trajectory are the following:

$$q_i(t) = q_i(t_a) + \dot{q}_{ic}(t - t_a) \tag{6.3.2.6}$$

Due to the symmetry of the trajectory, at half time the following equation must hold, that incorporates the 2 boundary conditions for the joint position

$$q_i(t_h) = \frac{q_i + q_{i+1}}{2}, \quad \text{where} \quad t_h = \frac{t_{i+1} - t_i}{2} \tag{6.3.2.7}$$

and

$$q_i(t_h) = q_i(t_a) + \dot{q}_{ic}(t_h - t_a) \tag{6.3.2.8}$$

Combining 6.3.2.7 and 6.3.2.8 the joint position at t_a is

$$q_i(t_a) = \frac{q_i + q_{i+1}}{2} - \dot{q}_{ic}(t_h - t_a) \tag{6.3.2.9}$$

The trajectory solution for $t_a \leq t \leq t_d$ is

$$\begin{aligned} q_i(t) &= \frac{q_i + q_{i+1}}{2} - \dot{q}_{ic}(t_h - t_a) + \dot{q}_{ic}(t - t_a) \\ \dot{q}_i(t) &= \dot{q}_{ic} \\ \ddot{q}_i(t) &= 0 \end{aligned} \tag{6.3.2.10}$$

For $t_d \leq t \leq t_{i+1}$ the equations of the trajectory are similar to those of the first interval. Let q_{i1} be the parabolic function of the first segment. From that the other parabolic segment will be calculated by mirroring q_{i1} by the t-axis, shift to t_{i+1} on the t-axis and translate by $q_{i+1} + q_i$ on the y-axis.

$$q_i(t) = -q_{i1}(t - t_{i+1}) + (q_{i+1} + q_i) \tag{6.3.2.11}$$

The trajectory solution for $t_d \leq t \leq t_{i+1}$ is

$$\begin{aligned} q_i(t) &= q_{i+1} - \frac{\dot{q}_{ic}}{2t_a}t^2 \\ \dot{q}_i(t) &= -\frac{\dot{q}_{ic}}{t_a}t \\ \ddot{q}_i(t) &= -\frac{\dot{q}_{ic}}{t_a} \end{aligned} \tag{6.3.2.12}$$

It is important to note that in order for the motion control using a trapezoid velocity profile to be feasible, the desired constant velocity $\dot{q}_{ic}$ must satisfy a velocity constraint. The constraint is calculated using the equations at time $t_a = t_i + \tau$

$$q_i(t_a) = \frac{q_i + q_{i+1}}{2} - \dot{q}_{ic}\left(\frac{t_f}{2} - t_a\right) = q_i + \frac{\dot{q}_{ic}}{2t_a}t_a^2 \tag{6.3.2.13}$$

where $t_f/2 = t_h$

$$\frac{q_i + q_{i+1} - \dot{q}_{ic}t_f}{2} + \dot{q}_{ic}t_a = q_i + \frac{\dot{q}_{ic}}{2}t_a \tag{6.3.2.14}$$

$$\frac{q_{i+1} - q_i - \dot{q}_{ic}t_f}{2} = -\frac{\dot{q}_{ic}}{2}t_a \tag{6.3.2.15}$$

$$t_a = \frac{\dot{q}_{ic}t_f + q_i - q_{i+1}}{\dot{q}_{ic}} \tag{6.3.2.16}$$

For the time constant t_a the following inequalities must hold (to simplify calculations we consider the case where $t_i = 0$)

$$0 \leq t_a \leq \frac{t_f}{2} = t_h \tag{6.3.2.17}$$

Substituting t_a with 6.3.2.16

$$0 \leq \frac{\dot{q}_{ic}t_f + q_i - q_{i+1}}{\dot{q}_{ic}} \leq \frac{t_f}{2} \tag{6.3.2.18}$$

$$0 \leq 2\dot{q}_{ic}t_f + 2(q_i - q_{i+1}) \leq \dot{q}_{ic}t_f \tag{6.3.2.19}$$

$$-2\dot{q}_{ic}t_f \leq 2(q_i - q_{i+1}) \leq -\dot{q}_{ic}t_f \tag{6.3.2.20}$$

$$-\frac{t_f}{q_i - q_{i+1}} \geq \frac{1}{\dot{q}_{ic}} \geq -\frac{t_f}{2(q_i - q_{i+1})} \tag{6.3.2.21}$$

$$\frac{q_i - q_{i+1}}{t_f} \leq \dot{q}_{ic} \leq \frac{2(q_i - q_{i+1})}{t_f} \tag{6.3.2.22}$$

6.3.2.2 S-Curve velocity profile

The s-curve trajectories introduce a cubic term to joint position polynomial, which has as a result the derivative of the acceleration, also known in the bibliography as jerk, to be non-zero. When introducing jerk in the equations of motion, then we can no longer have discontinuities in the acceleration which means generating a smoother trajectory than the trapezoid profile's trajectory. The equations of motions are studied in 7 intervals $[t_i, t_a], [t_a, t_b], [t_b, t_c], [t_c, t_d], [t_d, t_e], [t_e, t_g]$ and $[t_g, t_{i+1}]$ where

$$\begin{aligned} t_a &= t_i + \tau_1 \\ t_b &= t_i + \tau_1 + \tau_2 \\ t_c &= t_i + 2\tau_1 + \tau_2 \\ t_d &= t_{i+1} - 2\tau_1 - \tau_2 \\ t_e &= t_{i+1} - \tau_1 - \tau_2 \\ t_g &= t_{i+1} - \tau_1 \end{aligned} \tag{6.3.2.23}$$

where τ_1 is the time duration for both the acceleration and deceleration phases (symmetric trapezoid acceleration profile) and τ_2 is the time duration of constant, non-zero acceleration.

For $t_i \leq t \leq t_a$ the equations of the trajectory are the following:

$$q_i(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^3, \quad c_0, c_1, c_2, c_3 \in \mathbb{R} \tag{6.3.2.24}$$

$$\dot{q}_i(t) = c_1 + 2c_2 t + 3c_3 t^2 \tag{6.3.2.25}$$

$$\ddot{q}_i(t) = 2c_2 + 6c_3 t \tag{6.3.2.26}$$

Using the following initial conditions,

$$q_i(0) = q_i, \quad \dot{q}_i(0) = 0, \quad \text{and} \quad \ddot{q}_i(0) = 0 \tag{6.3.2.27}$$

the constants c_0, c_1 and c_2 can be calculated

$$c_0 = q_i, \quad c_1 = 0, \quad \text{and} \quad c_2 = 0 \tag{6.3.2.28}$$

Using a desired acceleration $\ddot{q}_a$ at time $t_a = t_i + \tau_1$, the constant c_3 can be calculated

$$q^{(4)}(t) = 6c_3 = J_a \quad \text{and} \quad J_a = \frac{\ddot{q}_a}{\tau_1} \tag{6.3.2.29}$$

The trajectory solution for $t_i \leq t \leq t_a = t_i + \tau_1$ is

$$\begin{aligned} q_i(t) &= q_i - \frac{\ddot{q}_a}{6\tau_1} t^3 \\ \dot{q}_i(t) &= \frac{\ddot{q}_a}{2\tau_1} t^2 \\ \ddot{q}_i(t) &= \frac{\ddot{q}_a}{\tau_1} t \end{aligned} \tag{6.3.2.30}$$

The trajectory solution for $t_a \leq t \leq t_b$ is

$$\begin{aligned} q_i(t) &= q_a + \dot{q}_a (t - t_a) + \frac{\ddot{q}_a}{2} (t - t_a)^2 \\ \dot{q}_i(t) &= \dot{q}_a + \ddot{q}_a (t - t_a) \\ \ddot{q}_i(t) &= \ddot{q}_a \end{aligned} \tag{6.3.2.31}$$

The trajectory solution for $t_b \leq t \leq t_c$ is

$$\begin{aligned} q_i(t) &= q_b + \dot{q}_b (t - t_b) + \frac{\ddot{q}_a}{2} (t - t_b)^2 - \frac{\ddot{q}_a}{6\tau_1} (t - t_b)^3 \\ \dot{q}_i(t) &= \dot{q}_\beta + \ddot{q}_a (t - t_b) - \frac{\ddot{q}_a}{2\tau_1} (t - t_b)^2 \\ \ddot{q}_i(t) &= \ddot{q}_a - \frac{\ddot{q}_a}{\tau_1} (t - t_b) \end{aligned} \tag{6.3.2.32}$$

The trajectory solution for $t_c \leq t \leq t_d$ is

$$\begin{aligned} q_i(t) &= q_c + \dot{q}_c(t - t_c) \\ \dot{q}_i(t) &= \dot{q}_c \\ \ddot{q}_i(t) &= 0 \end{aligned} \tag{6.3.2.33}$$

The trajectory solution for $t_d \leq t \leq t_e$ is

$$\begin{aligned} q_i(t) &= q_d + \dot{q}_c(t - t_d) - \frac{\ddot{q}_a}{6\tau_1}(t - t_d)^3 \\ \dot{q}_i(t) &= \dot{q}_c - \frac{\ddot{q}_a}{2\tau_1}(t - t_d)^2 \\ \ddot{q}_i(t) &= -\frac{\ddot{q}_a}{\tau_1}(t - t_d) \end{aligned} \tag{6.3.2.34}$$

The trajectory solution for $t_e \leq t \leq t_g$ is

$$\begin{aligned} q_i(t) &= q_e + \dot{q}_e(t - t_e) - \frac{\ddot{q}_a}{2}(t - t_e)^2 \\ \dot{q}_i(t) &= \dot{q}_e - \ddot{q}_a(t - t_e) \\ \ddot{q}_i(t) &= -\ddot{q}_a \end{aligned} \tag{6.3.2.35}$$

The trajectory solution for $t_g \leq t \leq t_{i+1}$ is

$$\begin{aligned} q_i(t) &= q_g + \dot{q}_g(t - t_g) - \frac{\ddot{q}_a}{2}(t - t_g)^2 + \frac{\ddot{q}_a}{6\tau_1}(t - t_g)^3 \\ \dot{q}_i(t) &= \dot{q}_g - \ddot{q}_a(t - t_g) + \frac{\ddot{q}_a}{2\tau_1}(t - t_g)^2 \\ \ddot{q}_i(t) &= -\ddot{q}_a + \frac{\ddot{q}_a}{\tau_1}(t - t_g) \end{aligned} \tag{6.3.2.36}$$

The equations of the s-curve profile trajectories can also be calculated numerically using integrals, which is however susceptible to small error accumulations. The integral trajectory solution for $t_k < t \leq t_{k+1}$ is

$$\begin{aligned} q_k(t) &= q_k + \int_{t_k}^{t} \dot{q}_k(\tau)d\tau \\ \dot{q}_k(t) &= \dot{q}_k + \int_{t_k}^{t} \ddot{q}_k(\tau)d\tau \\ \ddot{q}_k(t) &= \ddot{q}_k + \int_{t_k}^{t} \dddot{q}_k(\tau)d\tau \end{aligned} \tag{6.3.2.37}$$

In order to better impose the boundary conditions for the start and end joint positions as well as fix any calculation errors, the following correction (normalization) formula can be used. Let $\tilde{q}_1$ and $\tilde{q}_2$ be the calculated start and end joint positions respectively. Then for each calculated joint position $\tilde{q}_k = q_i(t_k), \quad t_i \leq t_k \leq t_{i+1}$, the corrected joint position q_k is calculated using the following equation

$$q_k = (\tilde{q}_k - q_1)\frac{q_2 - q_1}{\tilde{q}_2 - \tilde{q}_1} + q_1 \tag{6.3.2.38}$$

6.3.2.3 Comparison of velocity profiles

Both trapezoid and s-curve velocity profiles are widely used in industrial control systems, to design trajectories that are smooth and have a linear segment of constant velocity. Although

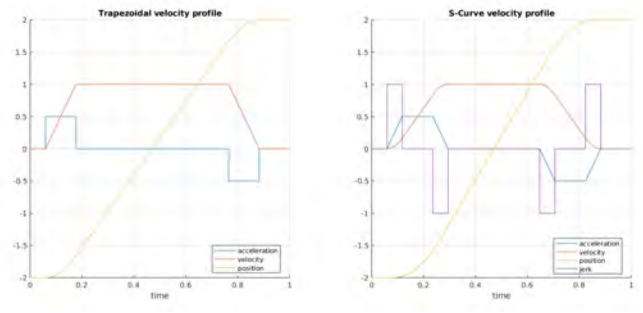

Figure 6.16: Joint trajectory generated using velocity profiles. Calculated and plotted in MATLAB

both profiles are continuous in position and velocity, the trapezoid profile does not have continuous acceleration and thus the generated trajectory is not smooth. These discontinuities in acceleration cause load oscillation or vibrations. This phenomenon can be easily observed in a cart-pendulum system like the one illustrated in 6.17. The s-curve velocity profile can generate a smooth trajectory which at the start and end of the motion will have significantly less oscillation (overshoot in position). Experimental data showing this difference in motion (in one case there is overshoot/oscillation and in the other case there is not), for the cart- pendulum system are available at the following article https://www.pmdcorp.com/resources/type/articles/get/s-curve-profiles-deep-dive-article. However, it is very important to note, that using smooth acceleration profiles in trajectories in the joint space $\mathbb{J}$ does not necessarily mean that the acceleration profile in the task space $\mathbb{P}$ will be equally smooth or any smooth at all due to the non-linear transformation between the spaces $\mathbb{J} \longleftrightarrow \mathbb{P}$ as well as the limits in the joint angles' ranges. This means that although smooth trajectories will cause less vibrations in the joint space, the vibrations/ oscillations are not necessarily avoided in the task space.

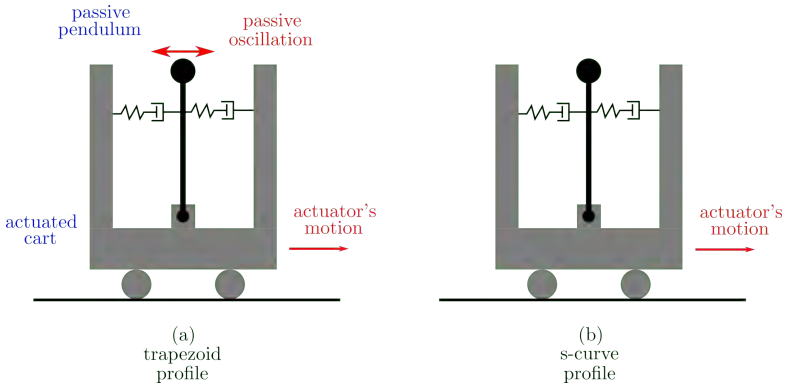

Figure 6.17: Trapezoid vs s-curve velocity profiles applied to the cart-inverted-pendulum system.

The smoothness and absence of vibration/oscillation/overshoot in the point-to-point trajectory of a s-curve trajectory are especially useful in surgical robots that use passive wrists. A **passive wrist** is a mechanical device attached at the end-effector of a robot arm and it usually has 3 degrees of freedom which are passive, i.e. not actuated. Passive wrists are very useful in minimally-invasive surgery because they allow to use surgical tools (usually endoscopic cameras) that are inserted in the human body at the surgical site, through the trocar and the incision, without exerting much pressure. Passive wrists, however have the downside of passiveness (e.g.

they can not be used for active tasks such as lifting an object, suturing etc.) and the difficulty in position control. Experimental data showing the overshoot and oscillation in position control of a laparoscopic camera attached to a passive wrist, are available at figure 7 of article [5].

An active-robot-passive-wrist system and the effect of trapezoid vs s-curve profile trajectories, can be studied using the analogous system of a cart-pendulum system as the one show in figure 6.17. In figure 6.17 the cart is the actuated mechanism (the one that will have a motor) and the pendulum is the passive mechanism (i.e. the joint that attached the pendulum to the cart is not actuated). The pendulum is also attached to a spring-damper system to better simulate the oscillating response to sudden changes in acceleration/force. The pendulum, although not actuated, moves based on the sudden changes of motion of the cart. Using this analogy, the actuated cart can be thought of as the active robotic arm's end-effector and the passive pendulum can be thought of as the passive wrist which is attached to the end-effector.

Trapezoid and s-curve velocity profiles can be further extended and generalized using higher order s-curves where the jerk is also bounded and continuous. Based on each use case the higher order s-curves can be used to generated trajectories that are both controllable smooth and time-optimal. Higher order s-curve profiles can be found at [53].

Chapter 7

Robotic MIS System Control

7.1 RCM Tracking

This section describes how to use an RCM-error metric in order to implement a control system that makes sure that the laparoscopic tool is always aligned with the fulcrum point and it satisfies the RCM constraint. To calculate this error, which can then be used as a feedback signal to a control system, the line of the long axis of the surgical tool must be first defined. To define this line, two points are calculated using the transformation of the surgical tool, which is attached to the robot's TCP on the end-effector. Let the following be the pose of the surgical tool with respect to the global reference frame ${}^{U}T_{T0} = \begin{bmatrix} \hat{\mathbf{x}} & \hat{\mathbf{y}} & \hat{\mathbf{z}} & \mathbf{p} \\ 0 & 0 & 0 & 1 \end{bmatrix}$. Using this pose, let A, B be the points such that $\overrightarrow{O_F A} = \mathbf{p}$, and $\overrightarrow{O_F B} = \mathbf{p} + \hat{\mathbf{x}}$, where O_F is the origin point of the fulcrum reference frame. The the line of interest is defined as the line l that passes through the points A and B.

The alignment error is calculated using a known position and orientation for the second fulcrum reference frame. In real-case scenarios the exact position and orientation of the uncertain fulcrum points are estimated. To model the uncertainty of the pose, the pose message of the fulcrum point includes also a covariance matrix. The ROS message that was used was the **PoseWithCovarianceStamped** which needs the following information (with respect to the global frame) to be fully defined: $(\mathbf{p}, \mathbf{q}, \mathbf{C})$, $\mathbf{p} \in \mathbb{R}^3, \mathbf{q} \in \mathbb{H}, \mathbf{C} \in \mathbb{R}^{6\times 6}$, where

$$\mathbf{C} = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} & \sigma_{x\psi} & \sigma_{x\theta} & \sigma_{x\varphi} \\ \sigma_{xy} & \sigma_{yy} & \sigma_{yz} & \sigma_{y\psi} & \sigma_{y\theta} & \sigma_{y\varphi} \\ \sigma_{xz} & \sigma_{yy} & \sigma_{zz} & \sigma_{z\psi} & \sigma_{z\theta} & \sigma_{z\varphi} \\ \sigma_{x\psi} & \sigma_{y\psi} & \sigma_{z\psi} & \sigma_{\psi\psi} & \sigma_{\psi\theta} & \sigma_{\psi\varphi} \\ \sigma_{x\theta} & \sigma_{y\theta} & \sigma_{z\theta} & \sigma_{\psi\theta} & \sigma_{\theta\theta} & \sigma_{\theta\varphi} \\ \sigma_{x\varphi} & \sigma_{y\varphi} & \sigma_{z\varphi} & \sigma_{\psi\varphi} & \sigma_{\theta\varphi} & \sigma_{\varphi\varphi} \end{bmatrix} \tag{7.1.0.1}$$

The covariance matrix and its coefficients can be calculated using he mean values of the estimations of $x, y, z, \psi, \theta, \varphi$.

Since both the origin of the fulcrum reference frame and the line of the long axis of the tool are known then the alignment error can be calculated as the distance of the line l from the point O_F $e_{rcm} = d(l, O_F)$.
To calculate the distance $d(l, O_F)$ there is available a method in the **Eigen** C++ library that calculates the distance of a line that passes through 2 points, from a third point. Alternatively, this distance can be calculated using 7.1.0.2.

$$d(l, O_F) = \frac{\|\overrightarrow{O_F A} \times \hat{\mathbf{x}}\|}{\|\hat{\mathbf{x}}\|}. \tag{7.1.0.2}$$

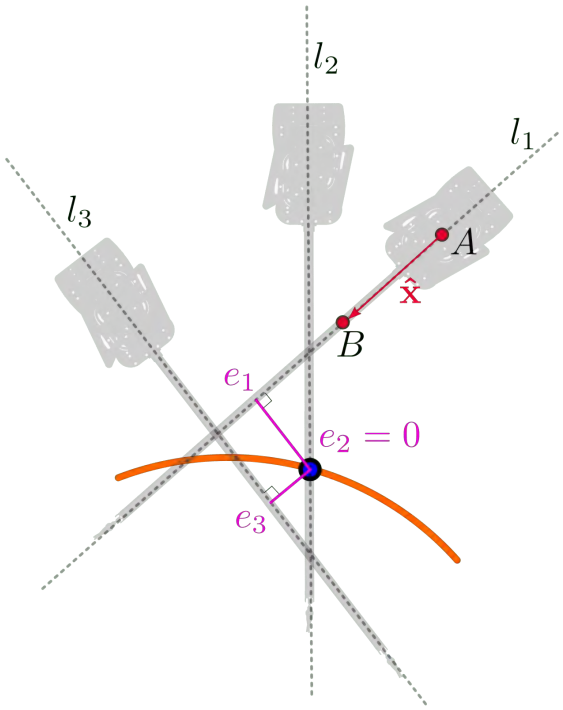

Figure 7.1: Geometric calculation of the RCM alignment error e using the distance between the line l and the RCM point.

Similar approaches to calculate this error/distance are presented in bibliography [18, 5].
If $e_{rcm} \leq 3_{\max} = 1$mm then the surgical tool axis passes through the fulcrum point and executes RCM trajectories, otherwise the robot is considered to have slipped from alignment, it does not execute RCM motion and probably generates a force $f \propto e$ which exerts pressure to the incision point and the abdominal wall (or the tissue around the incision point), which can have negative side-effects in the recovery of the incision. In [5] a different distance e_s is calculated as the RCM error as shown in Figure 7.2, which is along the x-axis of the fulcrum reference frame or the axis of the abdominal wall. Using this deviation distance e_s the force interaction between the surgical tool and the abdominal wall is $\|\mathbf{f}_s\| = \lambda e_s$, where λ is the elasticity of the abdominal wall and can be measured experimentally and $e_s = \frac{1}{cos\gamma} e$.

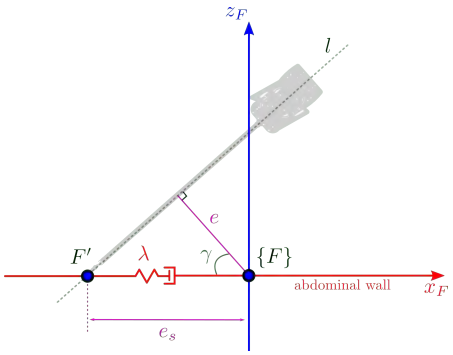

Figure 7.2: Force interaction model of the laparoscopic tool and the abdominal wall around the fulcrum point (RCM point)

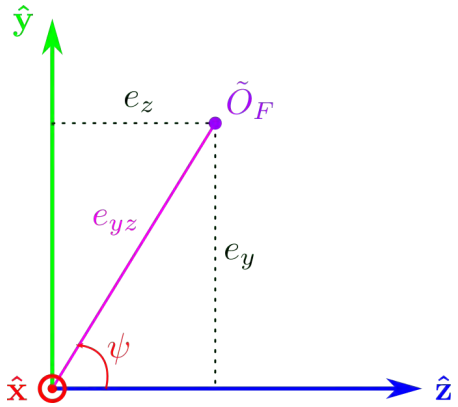

Figure 7.3: RCM error calculation in yz plane. The RCM error or yz-error is the distance between the line of the $\hat{\mathbf{x}}$ vector (here seen as a point) and the estimated position of the origin of the fulcrum reference frame $\tilde{O}_F$

The error e_{rcm} can provide further information if the distance of the line l (or of the $\hat{\mathbf{x}}$ axis) from the fulcrum point, is seen from a different perspective than that of Figure 7.1. If this distance is seen from a plane that is perpendicular to $\hat{\mathbf{x}}$, then the line is seen as a point and the distance $d(l, O_F)$ is seen as a distance between 2 points, as illustrated in Figure 7.3. This perspective of the RCM error (will also be referenced as yz-error e_{yz}) is more useful because it decomposes the error distance in 2 components that can be used to correct the goal pose of the robot so that is fixes the RCM misalignment, as in $e_y = e_{yz}sin\psi$ and $e_z = e_{yz}cos\psi$. The angle ψ that is used to split e_{rcm} in the two components, is already known from the robot's pose and it is the yaw (also known as spin) angle of the surgical tool.
Using the estimated RCM error $\tilde{e}_{rcm}$ and the estimated position of the origin of the fulcrum reference frame, an adaptive motion control system can be designed that corrects the trajectory to avoid RCM misalignment. The proposed control system is illustrated in block diagram 7.4 following along a similar pivoting motion control system in [51].

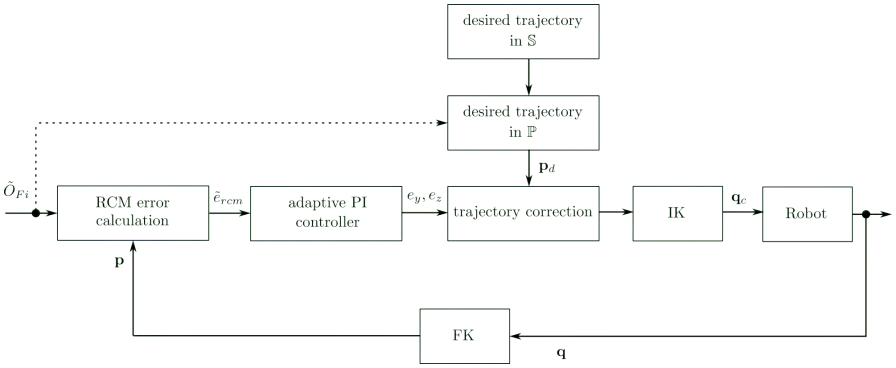

Figure 7.4: RCM tracking proposed control system.

7.2 Visual Servoing

7.2.1 Position based servoing

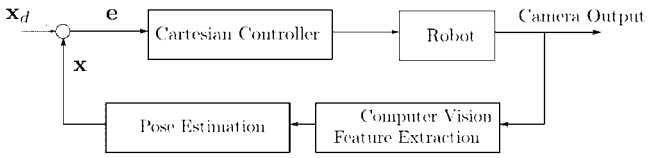

Figure 7.5: Position based visual servoing closed loop control

- **Photogrammetric technique**
- **Stereoscopic vision**: This methodology uses two separate views of the scene from two cameras and calculates the depth of various objects using information from both views.
- **Extracting depth from motion** This methodology is very similar to stereoscopic vision and is also known as *monocular* or *motion stereo*. It 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**

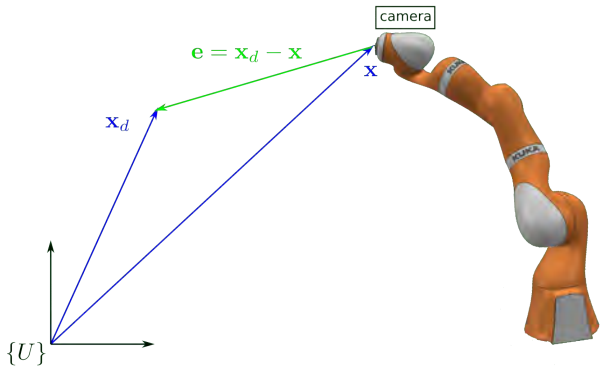

Figure 7.6: Position based visual servoing using depth from motion, stereo vision or 3D sensors, from which the desired position $\mathbf{x}_d$ is calculated and used to drive the robot.

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 Φigure 7.7 left and right frames). The commands 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 7.7: 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 depicted in Figure 7.8 consists of the **Image Controller**, the **Plant** (robot) and the **feedback term**. The Image controller is a simple **PD Controller** 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), as $\mathbf{x}[(k+1)T] = \mathbf{x}[kT] + \mathbf{u}[kT]$, where $\mathbf{x}[kT] = [x, y, z, \theta, \varphi, \psi]^\top$ and the discrete PID control law is given by equation 7.2.2.1

$$\mathbf{u}[kT] = K_p \left[\mathbf{e}[kT] + \frac{T}{T_i} \sum_{i=0}^{k-1} \mathbf{e}[iT] + \frac{T_d}{T} \left(\mathbf{e}[kT] - \mathbf{e}[(k-1)T] \right) \right] \tag{7.2.2.1}$$

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

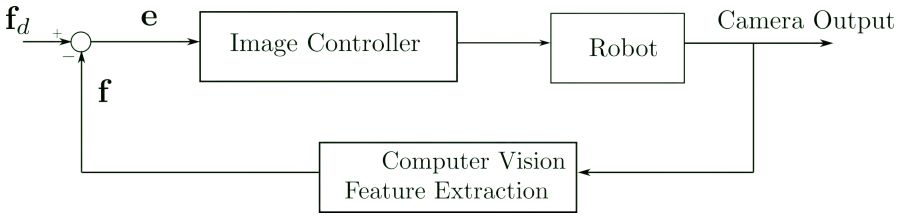

Figure 7.8: Image based visual servoing closed loop control

7.3 Firm grasping algorithm & Force control

In order to control the Barrett hand gripper and make sure that the gripper grasps firmly the surgical tool, a hybrid force control scheme must be implemented. This control scheme is hybrid because it combines positions measurements with force measurements. In the context of the Barrett hand, the position of a finger is known using the forward kinematics of the finger's joint angles and the forces are measured using the tactile sensors array which are distributed in the surface of the gripper and the fingers. The proposed force control system is illustrated in the block diagram 7.9.

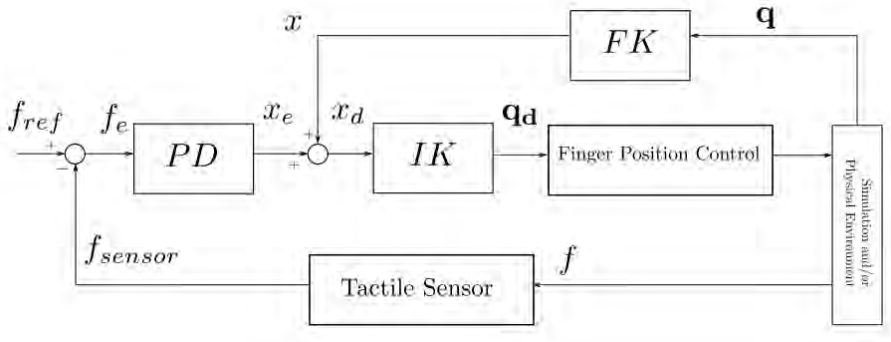

Figure 7.9: Force control on a Barrett Hand gripper finger

To ensure a firm grasp of the surgical tool, so that it can later be precisely manipulated to do pivot surgical motions, a reference force is required to be defined. The following analysis is for one of the 3 fingers of the gripper and it is the same for all of them. Following the block diagram 7.9, if the actual measured force is 0, then the finger is not in contact, which means that there is an error in force. This force error produces a position error via a first PD controller. This position error will be positive and will be added to the current measured position and thus will make the finger move closer to the object. When the finger contacts for the first time the object, then the measured force is non-zero, but it is still less than the desired force that is required for a firm grasp. This means that there is still a force error, which again produces a positions error and again makes the finger move even closer ("squeeze") the object. This control loop repeats until a satisfactory contact is reached and until the desired force is exerted on the object. If the measured force is bigger than the desired value, then the force error is negative and thus a negative position error is generated which in its turn makes the finger to move slightly away from the object, in order to reduce the exerted force. More about force control principles can be found at http://www.osrobotics.org/osr/control/principles.html as well as an example of force control in ROS and Gazebo at http://www.osrobotics.org/osr/control/examples.html.

Chapter 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**

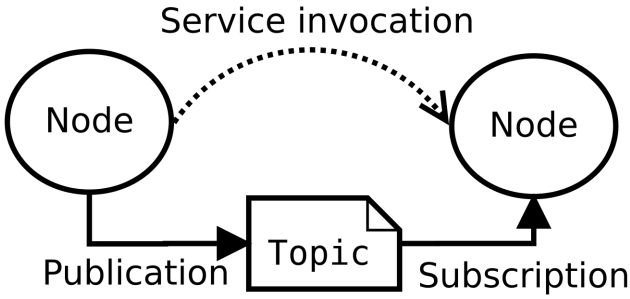

Figure 8.1: Communication diagram of 2 ROS nodes with a topic and a service

- **Filesystem tools**

A **node** is a small piece of code that runs in a process and executes a task. Usually nodes should execute one task and in cases of more complex robot applications, multiple nodes are combined and run together to run more complicated tasks. Nodes are combined together into a graph and communicate with each other using topics, remote procedure call services and the parameter server. Splitting the robot application into multiple smaller nodes is more fault tolerant, because an issue or a crash in one node does not have affect much the other nodes (individual nodes are isolated). Scalability (i.e. the ability to easily add new features and code) using nodes is also much easier, but the code complexity increases with the number of nodes, the more the number of nodes, the more the interconnections between them and the harder it is to debug issues or to run and maintain the nodes.

Topics are namespaced buses over which nodes exchange messages. A topic is a stream of messages with a **publish and subscribe** mechanism, which decouples the production of information from its consumption. Topics connect nodes in a many-to-many relationship, which means that all nodes that are interested in the topic data subscribe to the relevant topic and all nodes that generate data publish to the relevant topic. Topic data have a predefined **message** type that can be a String, Float, Integer, Array or even more complicated custom object structures.

8.2 Gazebo simulation environment

Gazebo http://gazebosim.org/ is the most popular simulation software used in most ROS projects and it is a very important tool that mimics the real world and how the robot would realistically interact with it. It comes with a wide range of features such as:

- Dynamics simulation with physics engines including ODE, Bullet and others, to efficiently simulate forces, torques and collisions
- Advanced 3D graphics utilizing OGRE that provide realistic environment rendering, lighting, shadows and textures.
- Sensors like 2D/3D cameras (optionally with simulated noise), laser range finders, contact sensors, force and torque sensors
- customizable plugins, pre-made 3D objects, environments and robots and many more

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

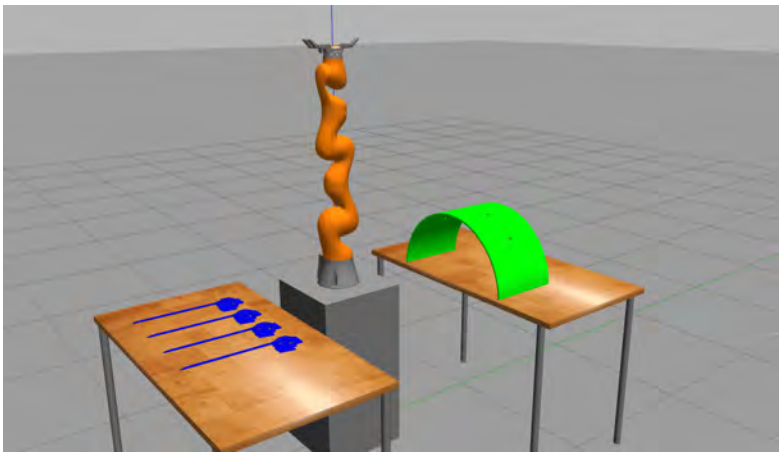

Figure 8.2: Simulation environment in Gazebo

- 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.

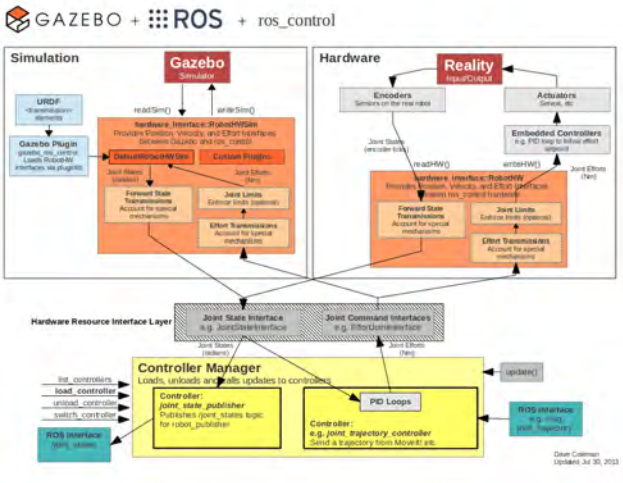

Figure 8.3: Control & Hardware Interfaces in Gazebo and ROS

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.

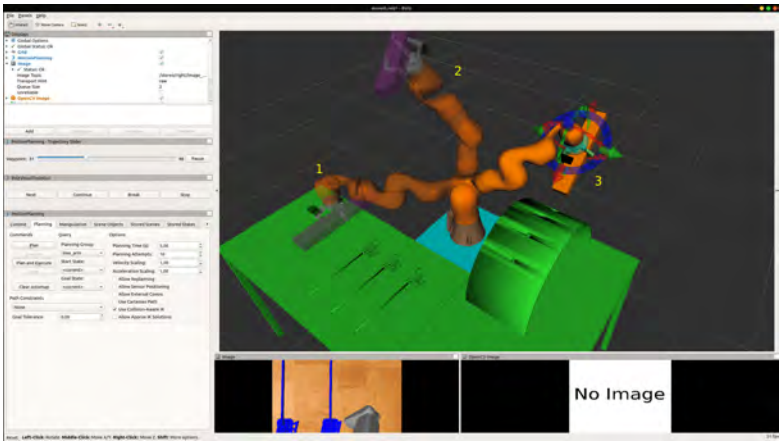

Figure 8.4: RViz: Visualizing the robot state as well as the state of the perceived world In this screenshot, various poses of the robot are shown: 1) the current actual real pose of the robot, 2) the planned pose and 3) the goal pose, which can freely be moved within the RViz environment

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

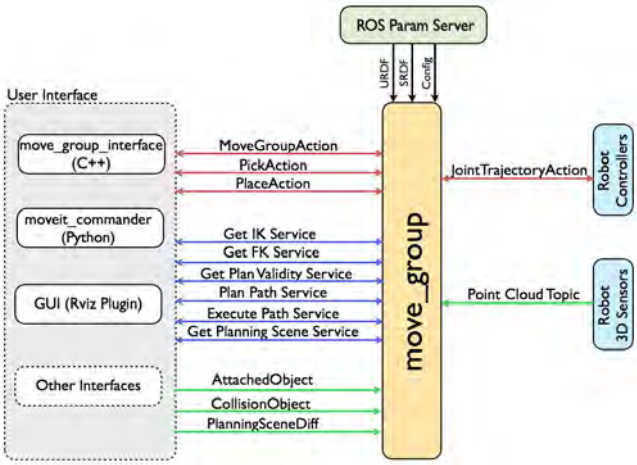

Figure 8.5: MoveIt ROS Architecture

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 maxiumum 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: 50 - 500μm
- Orientation tolerance: 0.00005 deg
- Planning time: 5-10s
- Replanning allowed: true

Typical motion planning parameter values inside surgical site:

- Position tolerance: 5μm
- Orientation tolerance: 0.000005deg
- Planning time: 5s
- Replanning allowed: true
- End-effector interpolation step: 1mm
- Maximum velocity scaling factor: 0.5
- Maximum planning attempts: 6

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:

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

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

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

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

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

8.5 Tools, Packages and Libraries

- **moveit** libraries for path planning and kinematics solutions (see more in 8.4). More specifically the following libraries, among others, were used moveit_core, moveit_ros_planning_interface, moveit_ros_planning, moveit_visual_tools
- **tf2** is an important library that allows to keep track of multiple coordinate frames over time as well as calculate poses with respect to a given reference frame and apply transformations to vectors or other transformations (see more in bibliography in [20] or at http://wiki.ros.org/tf2).
- **geometry_msgs**: used mostly for the Pose and PoseWithCovarianceStamped messages to exchange robot and object poses between nodes and give them as input to moveit's path planners
- **Eigen**: a c++ library to efficiently do linear algebra calculations (matrix multiplications, inversions, vector calculations and simple geometrix calculations like finding the distance between a line and a point)
- **OpenCV2** together with the **cv_bridge** ROS package for the computer vision tasks of this thesis
- **numpy**
- Stereo Vision with **stereo_image_proc**
- State machines with **Smach**

- **actionlib**
- **barrett_hand** https://github.com/RobotnikAutomation/barrett_hand.git
- **gazebo-pkgs** https://github.com/JenniferBuehler/gazebo-pkgs.git
- **moveit-pkgs** https://github.com/JenniferBuehler/moveit-pkgs.git
- **general-message-pkgs** https://github.com/JenniferBuehler/general-message-pkgs.git

Chapter 9

Simulation Studies

9.1 Robot Planner 1: Simple MoveIt planning

In this first simulation 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 simulation is to test the overall behaviour of the robot inside the work space, before implementing more complex path planning algorithms. In the sequel all simulations start from the home position, while the used parameters are: tolerances: 0.000005, max planning time: 5 seconds, replanning: true, max planning attempts: 6. Furthermore, the 'execution status' is 1 for success and 0 for failure.

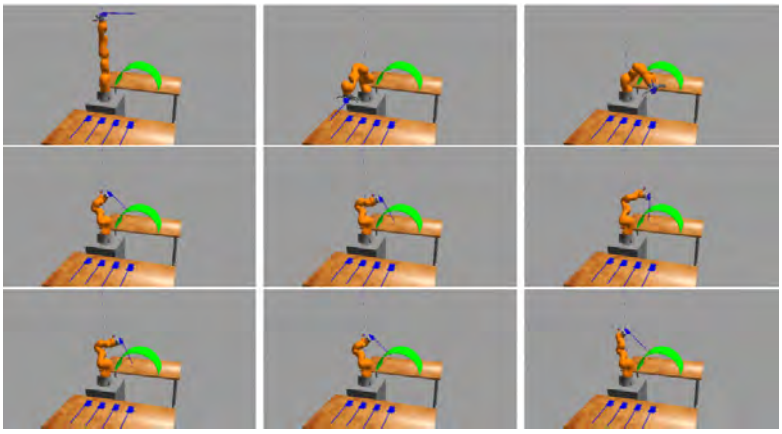

Figure 9.1: Simulation 1

Planner 1	**RRTConnect**
	Fake tool picking trajectory

Simulation	Approach planning time	Execution status	Move away planning time	Execution status
1	0.060402	1	0.144881	1
2	0.06053	1	0.058341	1
3	0.064706	1	0.060981	1
4	0.059789	1	0.119351	1
5	0.164789	1	0.330373	1
6	0.147037	1	0.289805	1
7	0.18233	1	0.093362	1
8	0.054469	1	0.062209	1
9	0.05313	1	0.409191	1
10	0.105909	1	0.307963	1
Average	0.0953091	1	0.1876457	1
Standard deviation	0.048230	-	0.125796	-
	Insertion & Pivot trajectories			
Simulation	Insertion planning time	Execution status	Pivot planning time	Execution status
1	0.216565	1	3.639193	1
2	0.057143	1	0.057187	1
3	0.073242	1	0.072451	1
4	0.056255	1	0.068156	1
5	0.16178	1	0.150123	1
6	0.071426	1	0.065059	1
7	0.155125	1	-	0
8	0.315921	1	0.194663	1
9	0.127381	1	0.19026	1
10	0.396182	1	0.075866	1
Average	0.163102	1	0.5014398	0.9
Standard deviation	0.110001	-	1.110582	-
	Reverse pivot & retraction trajectories			
Simulation	Reverse pivot planning time	Execution status	retraction planning time	Execution status
1	2.419514	1	2.388166	1
2	1.386188	1	2.874223	0
3	-	0	5.106926	1
4	5.466561	1	4.562506	1
5	5.62329	1	5.587679	1
6	5.488728	0	-	0
7	-	0	-	0
8	-	0	-	0
9	5.291442	1	5.096234	1
10	5.381595	1	5.576362	1
Average	4.436760	0.6	4.456014	0.6
Standard deviation	1.628910	-	1.204682	-

Table 9.1: Time results for Robot Planner (RP) 1 using the RRTConnect path planner algorithm. Planning time is the sum of solution time and path simplification time.

Planner 1	**RRT***			
	Fake tool picking trajectory			
Simulation	Approach planning time	Execution status	Move away planning time	Execution status
1	5.096772	1	5.023175	1
2	5.015857	1	5.034447	1
3	5.01843	1	5.101766	1
4	5.074703	1	5.089464	1
5	5.019108	1	5.056875	1
6	5.011631	1	5.217069	1
7	5.119638	1	5.027692	1
8	5.121159	1	5.040244	1
9	5.12812	1	5.098381	1
10	5.343964	1	5.374314	1
Average	5.0949382	1	5.1063427	1
Standard deviation	0.094669	-	0.104655	-
	Insertion & Pivot trajectories			
Simulation	Insertion planning time	Execution status	Pivot planning time	Execution status
1	5.200983	1	5.044568	1
2	5.009846	1	-	0
3	5.064216	1	5.085314	1
4	5.145115	1	5.059245	1
5	5.019322	1	5.173925	1
6	5.015804	1	-	0
7	5.033637	1	-	0
8	5.159212	1	5.100395	1
9	5.134356	1	5.081432	1
10	5.052	0	-	0
Average	5.083449	0.9	5.090813	0.6
Standard deviation	0.066253	-	0.041338	-
	Reverse pivot & retraction trajectories			
Simulation	Reverse pivot planning time	Execution status	retraction planning time	Execution status
1	-	0	-	0
2	-	0	-	0
3	-	0	5.508861	1
4	-	1	5.417935	1
5	5.057058	1	5.093558	1
6	-	0	-	0
7	-	0	-	0
8	-	0	-	0
9	-	0	-	0
10	-	0	5.358954	0
Average	inconclusive	0.2	inconclusive	0.3
Standard deviation	inconclusive	-	inconclusive	-

Table 9.2: Time results for robot planner 1 using the RRT* path planner algorithm. Planning time is the sum of solution time and path simplification time.

9.2 Robot Planner 2: Simulation layout and reachability studies

In this simulation, 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 study is 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.

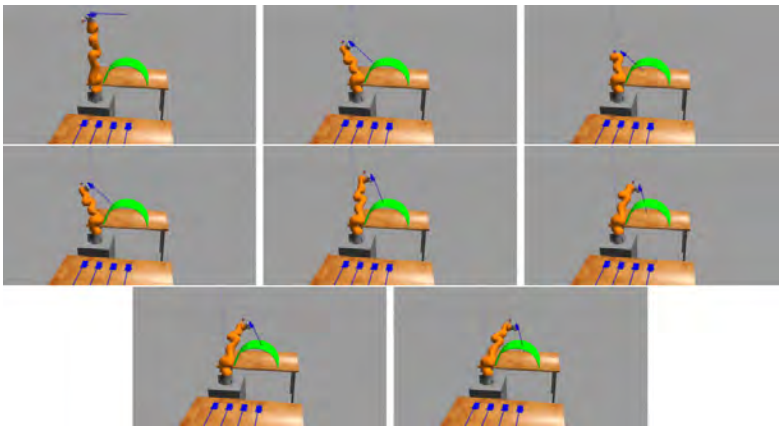

Figure 9.2: Simulation 2a:

Planner 2a	**RRTConnect**			
	Insertion & Pivot trajectories			
Simulation	Trocar1 insertion time	Execution status	Trocar1 retraction time	Execution status
1	0.060728	1	2.875072	1
2	0.070508	1	5.235667	1
3	0.071487	1	5.239014	1
4	0.059288	1	1.733352	1
5	0.053619	1	3.376982	1
6	0.062	1	5.389454	1
7	0.084007	1	-	0
8	0.058306	1	5.254352	1
9	0.046925	1	4.90185	1
10	0.060491	1	5.073171	1
Average	0.062736	1	4.342102	0.9
Standard deviation	0.010348	-	1.335526	-
	Insertion & Pivot trajectories			
Simulation	Approach trocar2 time	Execution status	Trocar2 insertion time	Execution status
1	2.67618	1	0.168283	1
2	5.187283	1	0.179048	1
3	0.050117	1	0.057194	1

4	1.538822	1	0.184641	1
5	1.62114	1	0.253224	1
6	5.397757	1	0.268188	1
7	5.379305	1	0.384924	1
8	5.401832	1	0.193265	1
9	2.845741	1	0.267069	1
10	5.355111	1	0.250265	1
Average	3.545329	1	0.220610	1
Standard deviation	2.038595	-	0.086008	-
	Insertion & Pivot trajectories			
Simulation	Trocar2 retraction time	Execution status	Approach trocar3 time	Execution status
1	0.152233	1	0.214332	1
2	0.321677	1	0.361712	1
3	5.404121	1	5.271597	1
4	0.147981	1	0.15915	1
5	0.333447	1	0.298799	1
6	0.246624	1	0.209347	1
7	0.381983	1	0.473779	1
8	0.219369	1	0.167211	1
9	0.227563	1	0.297372	1
10	5.149372	1	5.3578	1
Average	1.258437	1	1.281110	1
Standard deviation	2.120018	-	2.128103	-
	Insertion & Pivot trajectories			
Simulation	Trocar3 insertion time	Execution status	Trocar3 retraction time	Execution status
1	0.196175	1	0.855452	0
2	0.412455	1	1.133213	0
3	0.355923	1	0.814161	0
4	0.194839	1	0.885119	0
5	0.351561	1	0.364089	0
6	0.355251	1	0.826636	0
7	0.336123	1	0.30715	1
8	0.147867	1	1.053035	0
9	0.347956	0.6	0.80884	0.37
10	0.28364	0.6	0.229825	1
Average	0.298179	0.92	0.727752	0.237
Standard deviation	0.088385	-	0.314852	-

Table 9.3: Time results for RP 2a using the RRTConnect path planner algorithm. Planning time is the sum of solution time and path simplification time.

To overcome the reachability issue shown in Figure 9.2, 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.

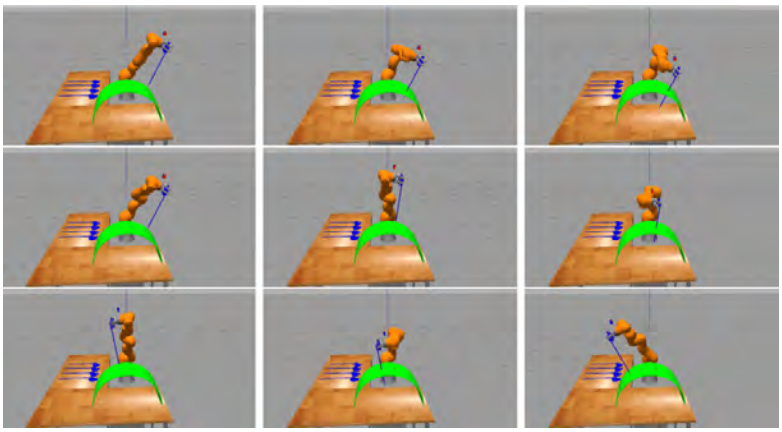

Figure 9.3: Simulation 2b: Running robot planner 2b with a different table layout so that all target poses have improved reachability. Note that all trajectories shown in the screenshots are collision-free, because the robotic arm is in an elbow-up configuration, so that it avoids the collision between the 3rd link of the robot and the mounting dock (green object shown in simulation).

Observing the screenshots at Figure 9.3, it is clear that the robot must always be in an elbow-up configuration in order to avoid collisions with the mounting dock. This constraint can be described either using the relative distance of the third link with the base or the relative angles of the third link's axis with respect to the axis of the base, as described with more detail in 2.2.3.3. The latter description with angles is easier to implement using the ROS MoveIt! framework, but it would require to solve an extra inverse and forward kinematics problem targeting the third link, instead of the end-effector. To avoid these additional, computationally expensive calculations and simplify the simulations, instead of mathematically describing the constraint, some extra points were added in the trajectory. These points are usually close to the robot's axis and are such, that moveit will be forced to find a solution with an elbow-up configuration (because there will be no other solution satisfying the target point). Starting with an elbow-up configuration makes the path planner stick with that configuration for most of the path, because otherwise it would make a big jump in the joint state space which is not allowed from the parameters used in the simulations.

Planner 2b	**RRTConnect**			
	Insertion & Pivot trajectories			
Simulation	Trocar1 insertion time	Execution status	Trocar1 retraction time	Execution status
1	0.269901	1	0.355588	1
2	0.220509	1	0.255968	1
3	0.260483	1	5.28604	1
4	0.267778	1	0.267796	1
5	0.388487	1	5.321684	1
6	0.316275	1	0.2088	1
7	0.202614	1	0.14469	1
8	0.289368	1	0.275917	1
9	0.353631	1	0.353631	1

10	0.185345	1	0.154755	1
Average	0.275439	1	1.262487	1
Standard deviation	0.064555	-	2.131175	-
	Insertion & Pivot trajectories			
Simulation	Approach trocar2 time	Execution status	Trocar2 insertion time	Execution status
1	0.322578	1	0.283497	1
2	0.209871	1	0.410543	1
3	5.099769	1	0.296228	1
4	0.216997	1	0.342776	1
5	5.328204	1	0.394472	1
6	0.450936	1	0.249248	1
7	0.336209	1	0.165375	1
8	0.397699	1	0.202118	1
9	0.235555	1	0.333129	1
10	0.320059	1	0.343595	1
Average	1.291788	1	0.302098	1
Standard deviation	2.069297	-	0.079229	-
	Insertion & Pivot trajectories			
Simulation	Trocar2 retraction time	Execution status	Approach trocar3 time	Execution status
1	5.374646	1	5.451335	1
2	0.452231	1	0.253669	1
3	0.22638	1	0.383933	1
4	5.371822	1	5.611649	1
5	0.395787	1	0.309212	1
6	5.15406	1	5.515007	1
7	4.189377	1	5.570568	1
8	5.167346	1	5.522056	1
9	5.101735	1	5.452301	1
10	5.273342	1	5.467273	1
Average	3.670673	1	3.953700	1
Standard deviation	2.311081	-	2.511216	-
	Insertion & Pivot trajectories			
Simulation	Trocar3 insertion time	Execution status	Trocar3 retraction time	Execution status
1	0.372382	1	5.34059	1
2	0.280432	1	0.261765	1
3	0.179581	1	0.187589	1
4	0.121094	1	5.152808	1
5	0.472468	1	-	0
6	0.251195	1	0.442282	0
7	0.20647	1	0.318481	1

8	0.178941	1	0.185758	1
9	0.188906	1	0.222821	1
10	0.165681	1	0.170876	1
Average	0.241715	1	1.364774	0.8
Standard deviation	0.107534	-	2.202951	-

Table 9.4: Time results for RP 2b using the RRTConnect path planner algorithm and a different table layout from RP 2a. Planning time is the sum of solution time and path simplification time.

Due to the probabilistic nature of the motion planner (in these simulations 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

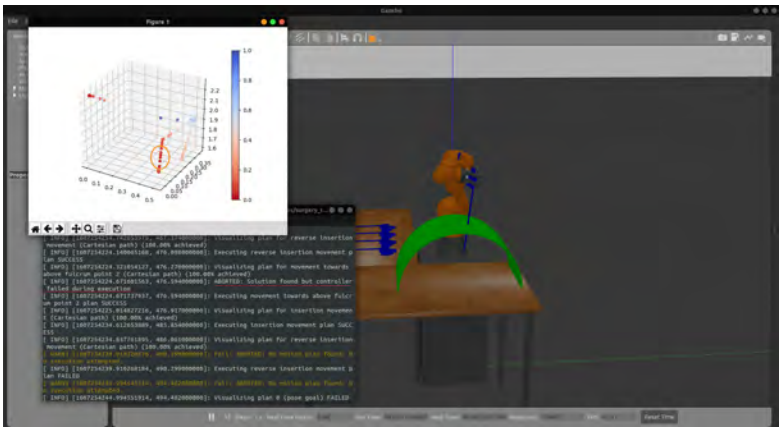

Figure 9.4: Simulation 2b: Singularity failure

Sometimes, when planning or executing a trajectory the robot may fail to complete the task because it has approached a singularity point. The points of singularity or of low dexterity are those points shown in the screenshot 9.4 which have red color. The color coding is calculated using the equation 5.3.0.1. The values of the Jacobian matrix are calculated at every kinematic state using a custom ROS node that subscribes to the robot's kinematic state (joints' angle positions, velocities, etc) and publishes the values of the Jacobian matrix as well as the forward kinematics solutions, to the /kinematic_state topic (see figure 9.5).

Figure 9.5: Custom kinematic state node that subscribes to the joint values and publishes forward kinematics solutions and the jacobian matrix values.

9.3 Robot Planner 3: Trajectory planning

The goal of this third simulation 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 group of simulations are the following:

- Line segment
- Circle
- Cubic spline
- B-Spline
- Quintic polynomial in joint space
- Trapezoidal velocity profile in joint space
- S-Curve velocity profile in joint space
- Helix

9.3.1 Line segment trajectories in task space

The goal of this simulation is to generate a line segment trajectory inside the surgical taskspace which will then be transformed via the fulcrum transformation to a trajectory that the robotic arm can execute. To define a line segment only two points are needed, the coordinates of the start and those of the end of the line segment. An other, way of defining a line segment trajectory, is by passing as parameters the coordinates of the start point and a vector attached to that point, whose and points to the end point. The second definition of the line segment can be very useful in cases where the direction of the line is needed.
In the Figure 9.7 one can see the RCM error values with respect to time. The x-axis shows the ROS time in seconds and the y-axis the RCM error in logarithmic scale. The robot starts from the home position where the RCM error is larger, then approaches the fulcrum point, where the error decreases and then it executes the line segment pivot trajectory where the error is bounded in the magnitude of less than a micrometer $10^{-6}m$ then stays constant, then changes again during the execution of the reverse line-segment trajectory and then finally it stays constant while the robot stays still in the initial insertion position. The RCM error while the robot is inserted but still and the error while the robot executes a pivot trajectory are studied separately.

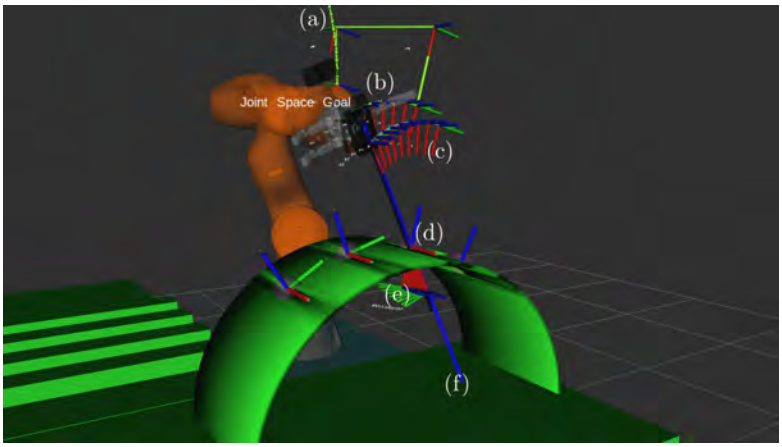

Figure 9.6: Simulation 3b: Create the line segment trajectory inside the surgical site (below the green mounting dock) and transform it via the fulcrum transformation to a trajectory for the robot's TCP. (a) Preparatory path to achieve elbow-up pose, (b) transformed line segment trajectory for the end-effector to follow, (c) transformed line segment trajectory with respect to the tool base frame (there is an offset from the end-effector see 2.1.1), (d) fulcrum reference frame 2 $\{F_2\}$, (e) line segment trajectory in surgical taskspace, (f) line of the axis along the length of the surgical tool, which is used to calculated the distance error (RCM deviation)

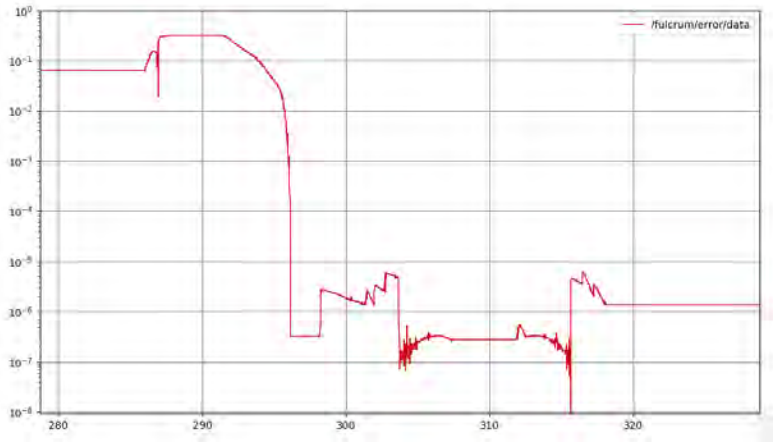

Figure 9.7: Simulation 3b: RCM error diagram from home position to line and reverse-line segment trajectories.

	Average [m] (accuracy)	Standard Deviation [m] (repeatability)	sample size
while pivoting	$2.112649 \cdot 10^{-6}$	$1.609277 \cdot 10^{-6}$	2309
while inserted and still	$2.948652 \cdot 10^{-7}$	$2.948652 \cdot 10^{-7}$	2696

Table 9.5: Accuracy and repeatability of the line segment simulation, calculated using measurements from 10 iterations of the same simulation

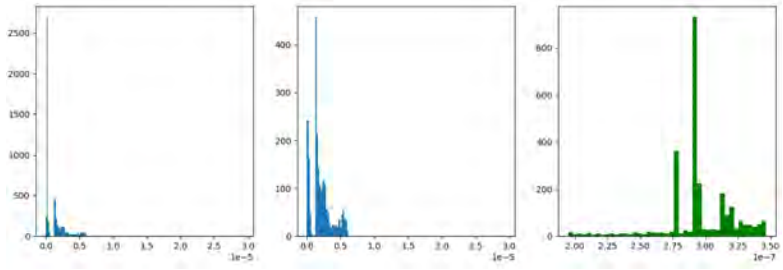

Figure 9.8: Simulation 3b: RCM error distributions, measurements from 10 iterations of the same simulation. From left to right: distribution of all measurements, distribution of measurements while the robot was pivoting, distribution of measurements while the robot was inserted but still.

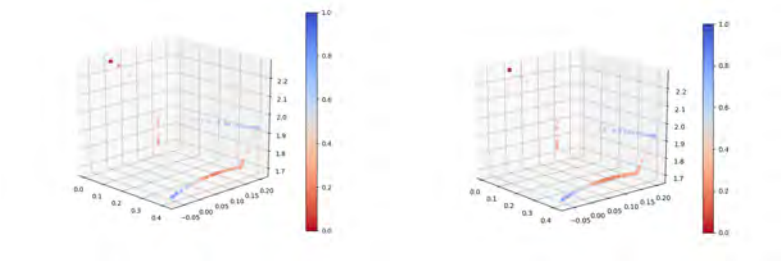

Figure 9.9: Simulation 3b: Manipulability plots of the whole trajectory the robot executed during 2 iterations of the same simulation

Note that although this type of trajectory is trivial in RP, this specific line segment trajectory should not be confused with ROS MoveIt line segment trajectories. The line-segment studied in this simulation is linear when defined in the surgical taskpsace, but it is highly non-linear when transformed via the fulcrum transformation and it has a curved shape. The only exceptions to this transformation is for line segments that lie on the direction defined by the radius unit vector $\hat{\mathbf{r}}$ of the spherical coordinate system of the Fulcrum reference frame, or equivalently the points A, B defining the line-segment and the origin O of the reference frame, must be collinear. These exceptions can also be planned by the MoveIt framework and are also known as and referenced throughout this thesis, as insertion movements. Let $\overrightarrow{AB}$ be the vector representing the line segment, then for the points A, B the following must hold, in order for the line segment to be invariant under the fulcrum transformation in terms of shape (i.e. the line to remain a line).

$$\overrightarrow{AB} = \overrightarrow{OB} - \overrightarrow{OA}, \text{and } \overrightarrow{OB} = r\overrightarrow{OA}, \quad r \in \mathbb{R} \tag{9.3.1.1}$$

9.3.2 Circular trajectories in task space

The goal of this simulation is to generate a circular trajectory inside the surgical taskspace which will then be transformed via the fulcrum transformation to a trajectory that the robotic arm can execute. To define a circular trajectory in the taskspace, two parameters are required: the (x, y, z) coordinates of the circle's center and its radius r.

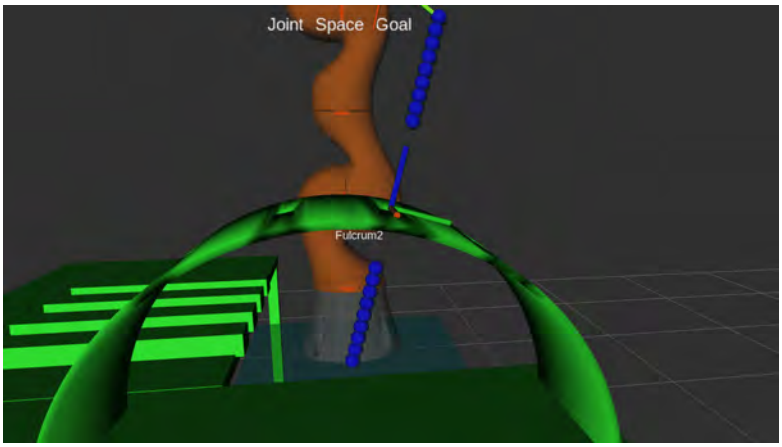

Figure 9.10: Simulation 3b: Line segment trajectory that is invariant under the fulcrum transformation, in terms of shape (the line remains a line).

Planner 3b	Approach and **line segment pivot** trajectories with **RRTConnect**			
	elbow-up preparatory path			
Simulation	Elbow-up Start pose planning time (sec)	Execution status	Elbow-up preparation path planning time (sec)	Execution status
1	0.221091	1	0.074075	1
2	0.19466	1	0.194823	1
3	0.268828	1	0.073928	1
4	0.154783	1	0.049908	1
5	0.110496	1	0.104181	1
6	0.155316	1	0.061127	1
7	0.140189	1	0.048735	1
8	0.166412	1	0.261208	1
9	0.121243	1	0.247453	1
10	0.209206	1	0.054966	1
Average	0.174222	1	0.117040	1
Standard deviation	0.049002	-	0.084238	-
	Approach & Insertion			
Simulation	Approach fulcrum 2 path planning time (sec)	Execution status	Insertion path planning time (sec)	Execution status
1	0.067982	1	0.30244	1
2	0.345301	1	0.325638	1
3	0.112407	1	0.173074	1
4	0.054112	1	0.194001	1
5	0.117048	1	0.23512	1
6	0.055864	1	0.284625	1

7	0.07501	1	0.231642	1
8	0.170249	1	0.105259	1
9	0.102762	1	0.374426	1
10	0.064243	1	0.269333	1
Average	0.116498	1	0.249556	1
Standard deviation	0.088078	-	0.078941	-
	Line segment pivot trajectories			
Simulation	Line segment path planning time (sec)	Execution status	Reverse line segment path planning time (sec)	Execution status
1	5.326128	1	5.486623	1
2	0.479823	1	5.279854	1
3	0.288405	1	-	0
4	0.295343	1	-	0
5	0.225066	1	5.445483	1
6	5.132968	1	-	0
7	0.300322	1	5.285311	1
8	5.484813	1	5.403696	1
9	0.297406	1	5.309383	1
10	0.259739	1	5.285902	1
Average	1.809001	1	5.356607	0.7
Standard deviation	2.421448	-	0.086818	-

Table 9.6: Time results for RP 1 using the RRTConnect path planner algorithm. Planning time is the sum of solution time and path simplification time.

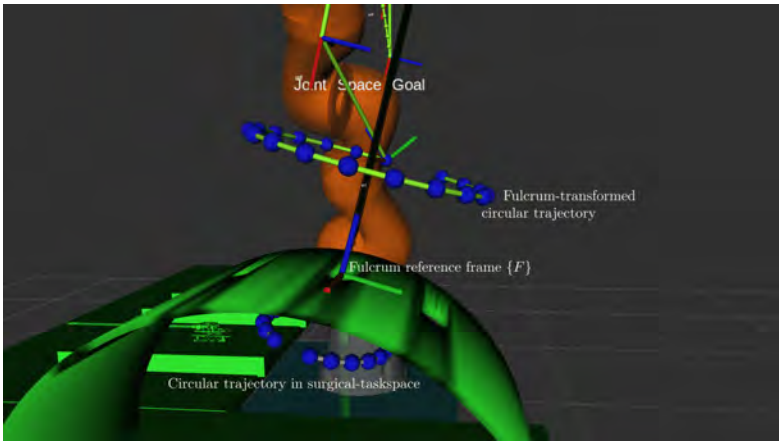

Figure 9.11: Simulation 3a: Create circular trajectory inside the surgical site (below the green mounting dock) and transform it via the fulcrum transformation to a trajectory for the robot's TCP.

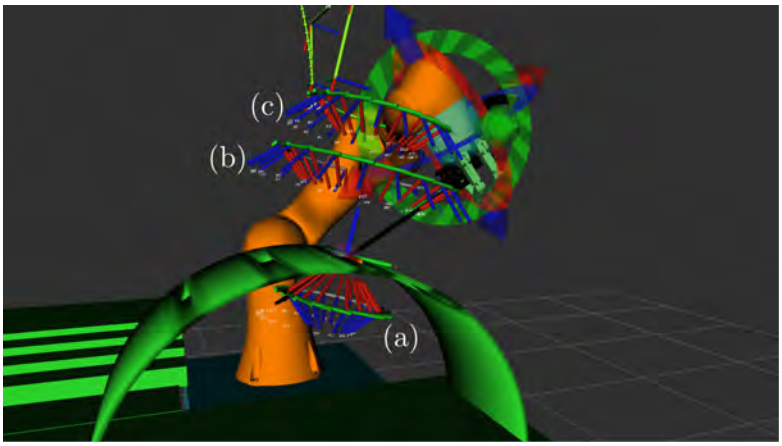

Figure 9.12: Simulation 3a: A more detailed view of the circular trajectories. (a) the original circular trajectory designed inside the surgical taskspace, (b) the transformed trajectory that the base of the surgical tool will follow, (c) the actual transformed trajectory that the robot's end-effector will follow

Note that in this simulation (see screenshot 9.11), the trivial case is examined where the circle is parallel to the xy plane of the fulcrum's coordinate system. That is because the parametric definition if the circle, can be easily expressed when in a standard plane of the coordinate system i.e. xy, xz or yz planes and not in an arbitrary orientation.

9.3.3 Cubic Spline trajectories in task space

The goal of this simulation is to generate a cubic spline trajectory inside the surgical taskspace which will then be transformed via the fulcrum transformation to a trajectory that the robotic arm can execute. Cubic spline equations are presented in more detail at 6.2.5 and the generated trajectories of this simulation can be seen in Figure 9.13.

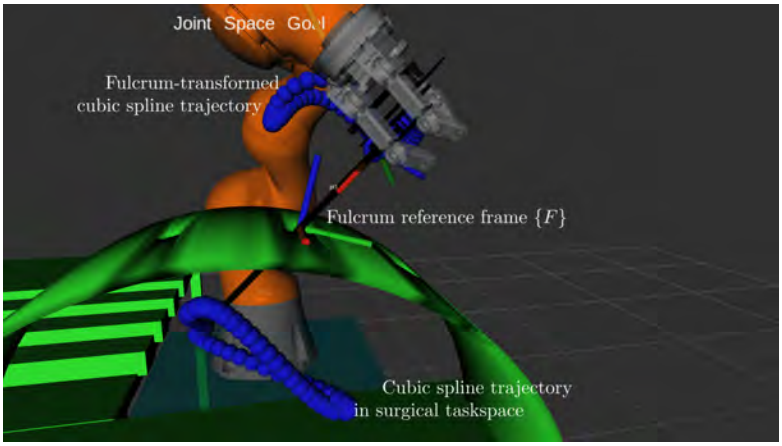

Figure 9.13: Simulation 3c: Create the cubic spline trajectory inside the surgical site (below the green mounting dock) and transform it via the fulcrum transformation to a trajectory for the robot's TCP.

As seen on the screenshot 9.13, for the surgical task there are 4 four points that were selected to create 3 cubic splines. The points that define the trajectory were:

$$\mathbf{x}_1 = [-0.1, -0.1, -0.2]$$
$$\mathbf{x}_2 = [0.1, -0.1, -0.1]$$
$$\mathbf{x}_3 = [0.1, 0.1, -0.2]$$
$$\mathbf{x}_4 = [-0.1, -0.1, -0.2] = \mathbf{x}_1$$

and a constant derivative for all points of $\mathbf{x}_d = [0.1, 0.1, 0.1]$. Note that this derivative describes the shape and smoothness at the points that connect 2 consecutive splines. These derivatives may not necessarily express a velocity in the cartesian space. That is because this derivative is calculated with respect to the path variable s. The time derivative of the trajectory would be calculated using the chain rule:

$$\dot{\mathbf{x}} = \frac{d\mathbf{x}}{ds} \cdot \frac{ds}{dt} = \mathbf{x}_d \cdot \frac{ds}{dt} \tag{9.3.3.1}$$

9.3.4 B-Spline trajectories in task space

The goal of this simulation is to generate a B-Spline trajectory inside the surgical taskspace which will then be transformed via the fulcrum transformation to a trajectory that the robotic arm can execute. B-spline equations are presented in more detail at 6.2.6 and the generated trajectories of this simulation can be seen in figure 9.14.

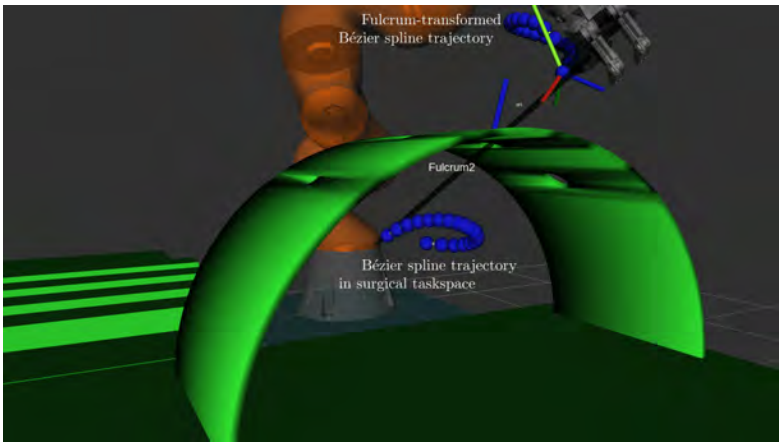

Figure 9.14: Simulation 3d: Create the B-spline trajectory inside the surgical site (below the green mounting dock) and transform it via the fulcrum transformation to a trajectory for the robot's TCP.

9.3.5 Polynomial trajectories in joint space

The goal of this simulation is to generate a trajectory in joint space using a polynomial of 5th degree (see 6.3.1). To generate this trajectory, at least 2 points are needed in the cartesian space. For each of these points the inverse kinematics problem is solved. The joint values of the first point are used as the start points of the trajectory and the joint values of the second point are used as the end points. The velocities and accelerations

are chosen to be zero for both start and end positions. The poses of the 2 points that were used as input to the 2 IK problems are

$$T_A = \begin{bmatrix} & \mathbf{R_{rpy}} & & 0.1 \\ & & & 0.1 \\ & & & 1.95 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \; T_B = \begin{bmatrix} & \mathbf{R_{rpy}} & & 0.4 \\ & & & 0.1 \\ & & & 1.95 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{where} \mathbf{R_{rpy}} = \mathbf{Rot}(\hat{\mathbf{z}}, 2.952052) \cdot \mathbf{Rot}(\hat{\mathbf{y}}, 1.311528) \cdot \mathbf{Rot}(\hat{\mathbf{x}}, -1.750799)$$

and the solutions chosen for each point are

$$\mathbf{q_A} = \begin{bmatrix} 1.148534 \\ -0.583084 \\ -0.212395 \\ -1.430756 \\ 0.609015 \\ 1.168518 \\ -0.523555 \end{bmatrix} \quad \text{and} \quad \mathbf{q_B} = \begin{bmatrix} 2.293053 \\ -0.277892 \\ -1.795364 \\ -0.941923 \\ 0.876039 \\ 1.451593 \\ -0.916277 \end{bmatrix}$$

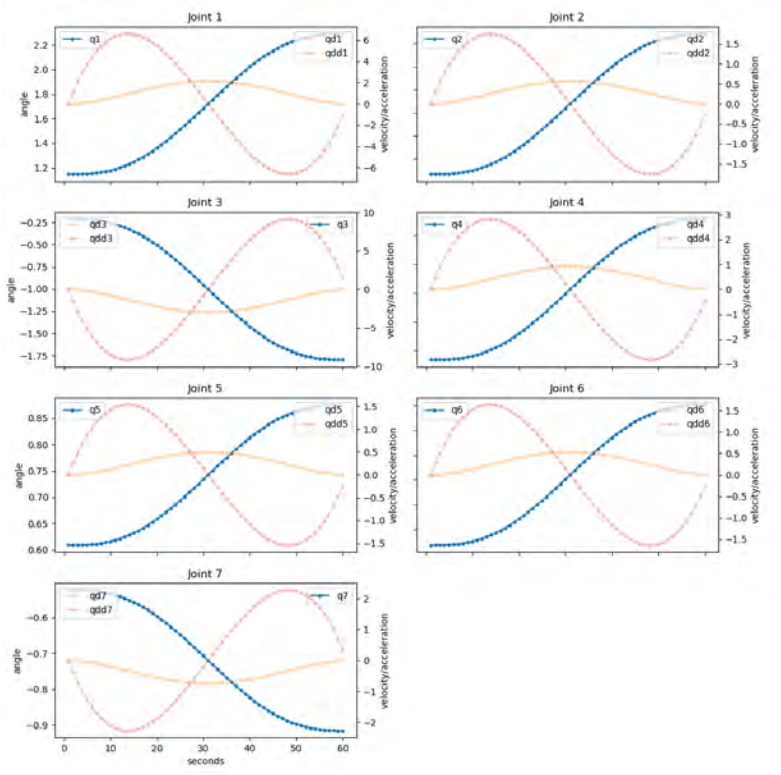

Figure 9.15: Simulation 3e: Polynomial trajectories of 5th degree for each of the 7 joints.

9.3.6 Trajectories in joint space with trapezoidal velocity profile

The goal of this simulation is to generate a trajectory in joint space using a trapezoidal velocity profile. The start and end points as well as the inverse kinematics solutions for

these points are the same as those in 9.3.5.

9.3.7 Trajectories in joint space with s-curve velocity profile

The goal of this simulation is to generate a trajectory in joint space using a smooth s-curve velocity profile. The start and end points as well as the inverse kinematics solutions for these points are the same as those in 9.3.5.

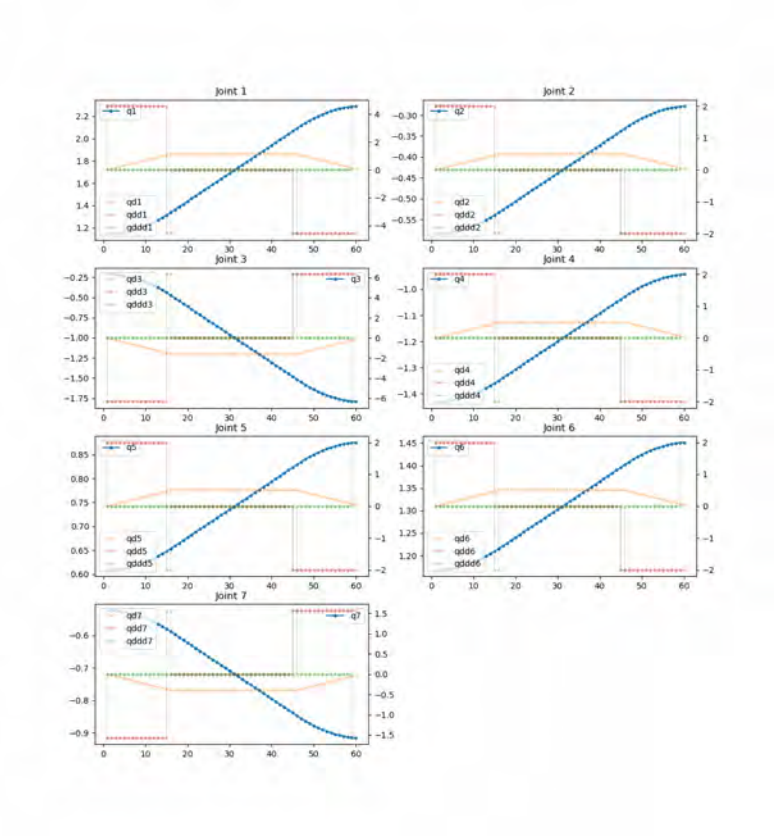

Figure 9.16: Simulation 3f: Trajectory using a trapezoid velocity profile ($\tau = 15$ seconds).

9.3.8 Helical trajectories in task space

The goal of this simulation is to generate a helical trajectory inside the surgical taskspace which will then be transformed via the fulcrum transformation to a trajectory that the robotic arm can execute. To define a helical trajectory in the taskspace, two parameters are required: the (x, y, z) coordinates of the helix center, its radius r, as well as the number of cycles τ of the helix and slope parameter β.

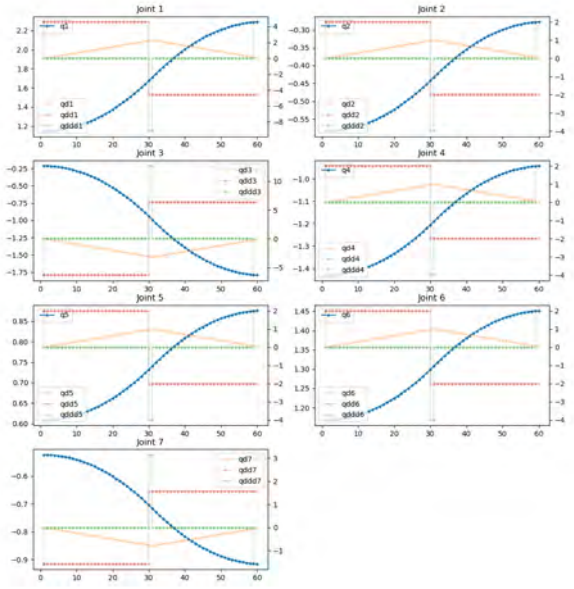

Figure 9.17: Simulation 3f: Trajectory using a trapezoid velocity profile ($\tau = 30$ seconds)

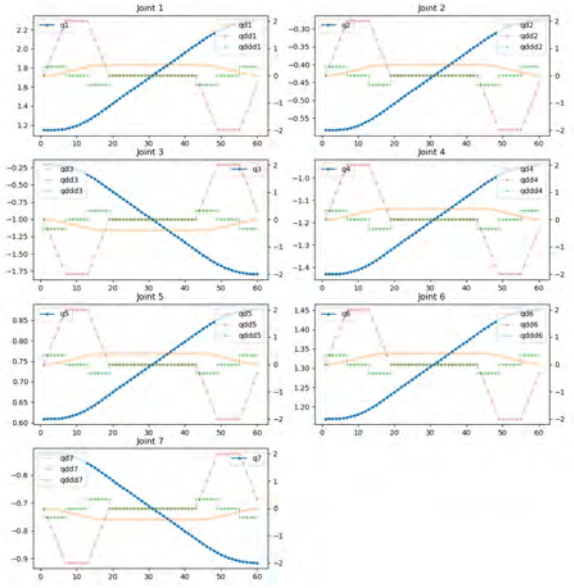

Figure 9.18: Simulation 3g: Trajectory using a s-curve velocity profile ($\tau_1 = \tau_2 = 6$)

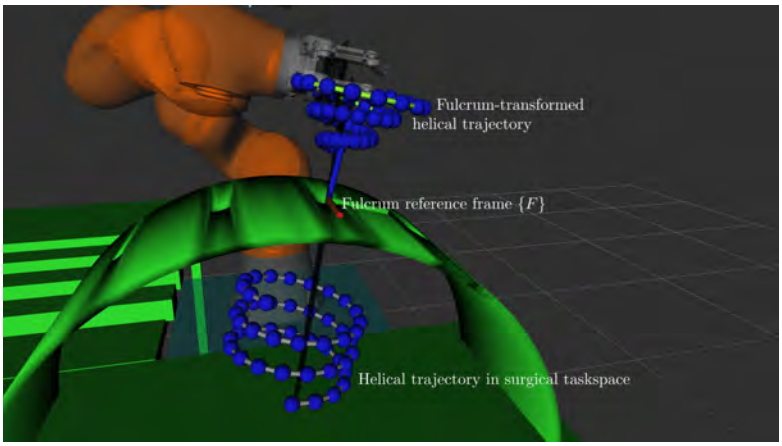

Figure 9.19: Simulation 3h: Helical trajectory creation inside the surgical site (below the green mounting dock) and subsequent transformation via the fulcrum transformation to a trajectory for the robot's TCP.

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

In this simulation 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 simulation 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.

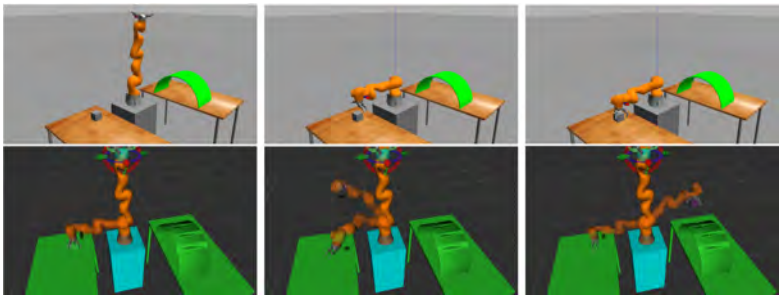

Figure 9.20: Simulation 4: Simple pick-and-place simulation of a cube

Observing the results from table 9.7, the time duration for the pick pipeline of this simulation using the RRTConnect path planning algorithm, ranges from approximately 0.01 to 0.02 seconds, whereas in the Place pipeline we observe mainly 2 distinct groups of time ranges. One group has values around 0.02 seconds and the other has bigger values around 0.14 seconds. In the case with the bigger time durations, the solution made the robot go around the obstacle whereas in the other cases the path planning solution

avoided the obstacle, which means that an easier path was found, it was solved quicker and the path solution was much simpler (in terms of kinematic constraints) which led to smaller path simplification time durations. It is also possible, that due to the obstacle and the initial conditions, the robot may not find a solution.

Planner 4	**RRTConnect**			
	Pick Pipeline			
Simulation	Status	Solution Time	Path Simplification Time	Planning Attempts
1	1	0.026189	0.065173	1
2	1	0.016835	0.043333	1
3	1	0.025439	0.023178	1
4	1	0.029292	0.066913	1
5	1	0.024873	0.01747	1
6	1	0.017615	0.024568	1
7	1	0.014263	0.028095	1
8	1	0.015479	0.027035	1
9	1	0.027803	0.057064	1
10	1	0.026057	0.033179	1
Average	1	0.02196178	0.0386008	1
	Place Pipeline			
Simulation	Status	Solution Time	Path Simplification Time	Planning Attempts
1	1	0.014644	0.020947	1
2	1	0.131334	0.938948	1
3	1	0.021425	0.08429	1
4	1	0.01675	0.01675	1
5	1	0.165795	1.661019	1
6	1	0.119819	0.394493	1
7	1	0.140645	0.476835	1
8	1	0.133917	0.673233	1
9	1	0.017644	0.038319	1
10	1	0.020025	0.074847	1
Average	1	0.0781998	0.4379681	1

Table 9.7: Time results for RP 4 using the RRTConnect path planner algorithm

The main observation from the results of table 9.8 is that RRT* time durations are larger than those from RRTConnect, but the RRT* algorithm has the advantage of finding better, more accurate solutions (approximately the same solutions at most attempts) and with less collisions. Although these time durations are not optimal for real-time applications, they could be useful in pick-and-place pipelines if the solutions are pre-computed and saved for later use.

Planner 4	**RRT***			
	Pick Pipeline			
Simulation	Status	Solution Time	Path Simplification Time	Planning Attempts
1	1	44.97476	0.030918	1

2	1	45.006897	0.073009	1
3	1	44.989085	0.053440	1
4	1	44.998414	0.034571	1
5	1	44.982524	0.041732	1
6	1	44.997148	0.017660	1
7	1	45.00115	0.022290	1
8	1	44.991409	0.046728	1
Average	1	44.99267	0.0400435	1
	Place Pipeline			
Simulation	Status	Solution Time	Path Simplification Time	Planning Attempts
1	1	45.064568	0.884821	1
2	1	44.965931	0.054692	1
3	1	44.976697	0.012784	1
4	1	44.968431	0.020354	1
5	1	45.068239	0.551800	1
6	1	45.016043	1.089529	1
7	1	45.048851	0.680121	1
8	1	45.023364	0.000004	1
Average	1	45.0165155	0.411763	1

Table 9.8: Time results for RP 4 using the RRT* path planner algorithm

Planner 4	**PRM***			
	Pick Pipeline			
Simulation	Status	Solution Time	Path Simplification Time	Planning Attempts
1	1	44.987182	0.000028	1
2	1	44.992757	0.000005	1
3	1	45.00983	0.000004	1
4	1	45.004154	0.105568	1
5	1	44.99367	0.000002	1
6	1	45.015169	0.000002	1
7	1	45.02914	0.000002	1
Average	1	45.00456	0.015088	1
	Place Pipeline			
Simulation	Status	Solution Time	Path Simplification Time	Planning Attempts
1	1	44.992207	0.000004	1
2	1	45.512132	0.000001	1
3	1	45.950533	0.000002	1
4	1	44.99777	0.000003	1
5	1	45.691226	0.000002	1
6	1	45.023251	0.013478	1
7	1	45.782305	0.000003	1
Average	1	45.421346	0.0019276	1

Table 9.9: Time results for RP 4 using the PRM* path planner algorithm

9.5 Robot Planner 5: Visual servoing

The goal of this fifth simulation is to control the KUKA robot using the camera via the visual servoing technique. In the first part of the simulation the robotic arm moves to an initial known position 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 simulation, the robot follows a similar algorithm. It starts with a known position, 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.

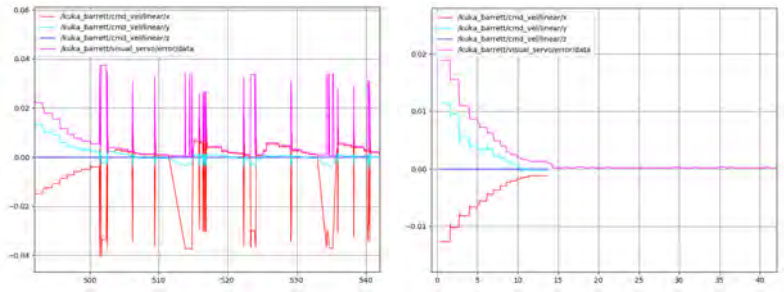

Figure 9.21: Visual servo controller error diagrams. On the left image in the error graphs appear some spikes. These spikes occur from the sudden temporary detection of a nearby surgical tool. On the right image, these spikes are filtered out, and only the error graphs of the visual servoing of one tool are shown. The controller parameters are $K_p = 0.9, K_d = 0.2$

9.6 Robot Planner 6: RCM alignment error in insertion and retraction

The goal of this simulation is to measure the alignment error of the long axis of the surgical tool with the fulcrum point. This error tracks whether the robot pose satisfies the RCM constraint, in which the axis of the tool must always pass through the fulcrum point and is explained in more detail in 7.1. To simplify the Simulation, constant values are assumed for both position coordinates and orientation angles and a constant arbitrary

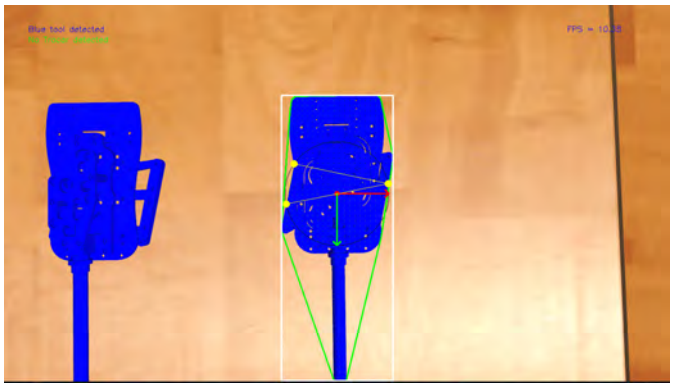

Figure 9.22: Image based visual servoing and calculation of grasp points. The yellow points are the grasp points and the thin black circumscribed circle is the growing circle that was used to calculate them.

diagonal covariance matrix.

$$\begin{bmatrix} x \\ y \\ z \\ \psi \\ \theta \\ \varphi \end{bmatrix} = \begin{bmatrix} 0.529996 \\ 0.059271 \\ 1.398114 \\ -0.271542 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{C} = diag(0.001, \cdots, 0.001) \tag{9.6.0.1}$$

where $\mathbf{p} = [x, y, z]$ and the Euler angles ψ, θ, φ are used to calculate the $\mathbf{q}$-quaternion .

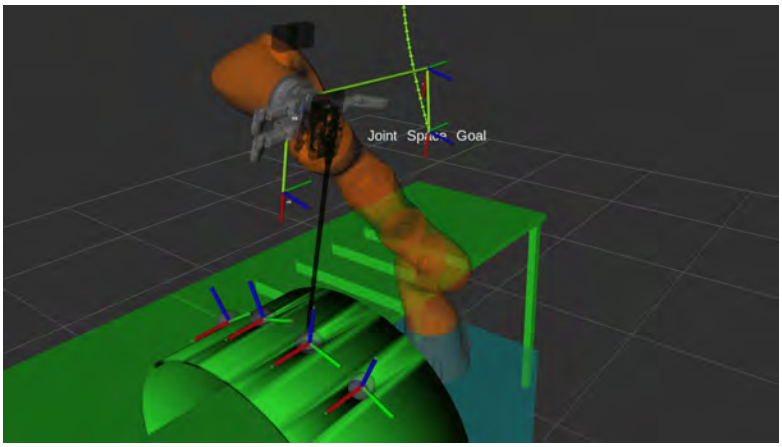

Figure 9.23: Robot planner 6: Path from home, to point above Fulcrum point and path of insertion/retraction

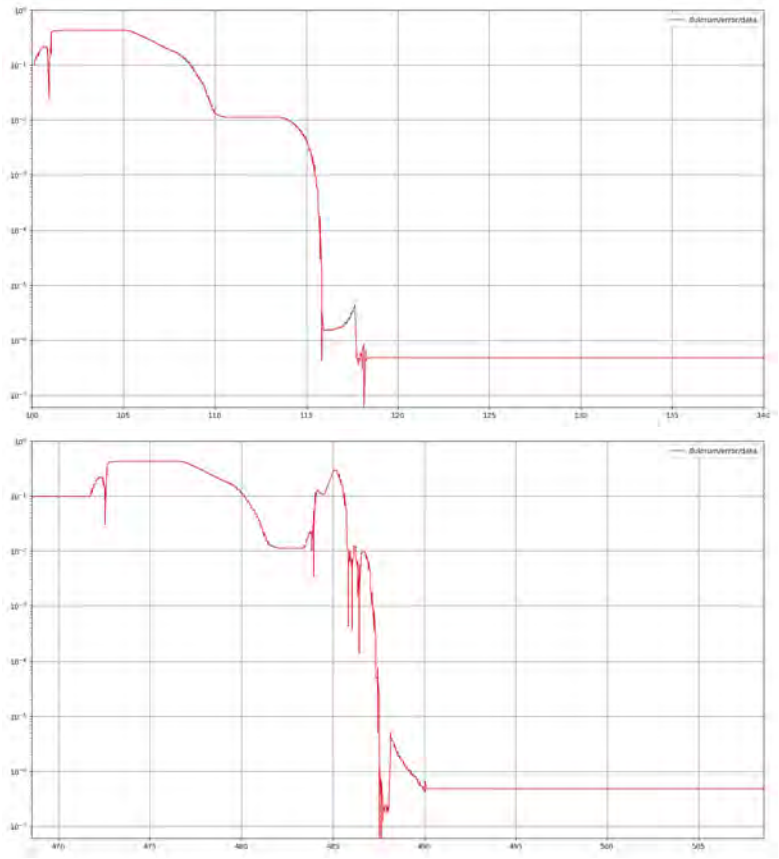

Figure 9.24: RCM alignment error, diagrams from 2 executions of the same simulation. The x-axis is measured in seconds (ROS time) and the y-axis is the distance error measured in meters in logarithmic scale.

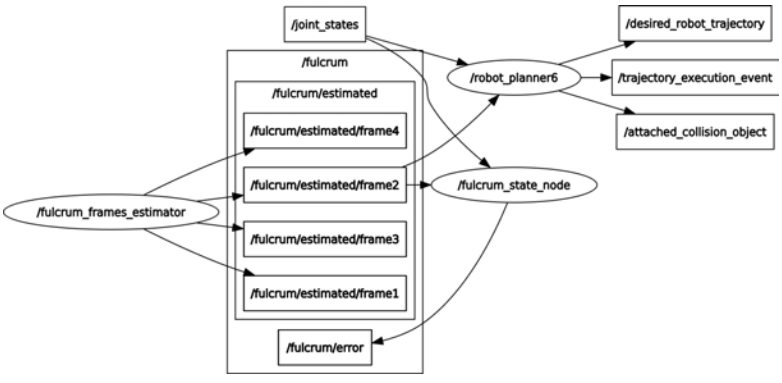

Figure 9.25: ROS nodes and topics used for the robot-planner6 simulation. In this simulation the trajectories and RCM fulcrum errors are with respect to the 2nd fulcrum reference frame only.

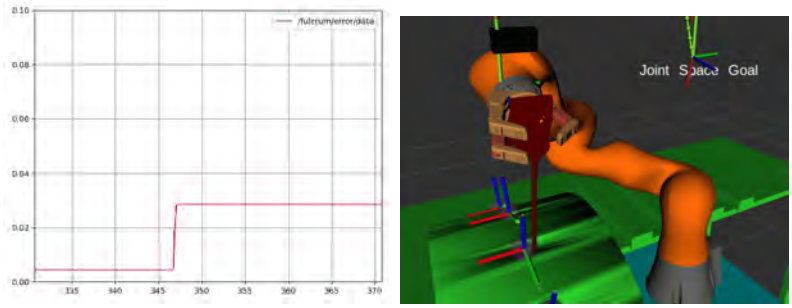

Figure 9.26: Surgical tool slides from an aligned, RCM pose to a pose misaligned from the fulcrum point, no RCM motion and surgical tool is exerting pressure to the abdominal wall. The diagram on the left shows the error starting from a correct position to a problematic position. The right image shows the tool in collision with the mounting dock (abdominal wall) and there is an obvious distance from the fulcrum reference frame.

9.7 Robot Planner 7: State machine - End-to-end running

The goal of this simulation is to run all stages of this thesis together to study the end-to-end result. Each stage is run by a different node and the sequence in which every task is executed is dictated by the status of the state machine as shown in Figure 9.27.

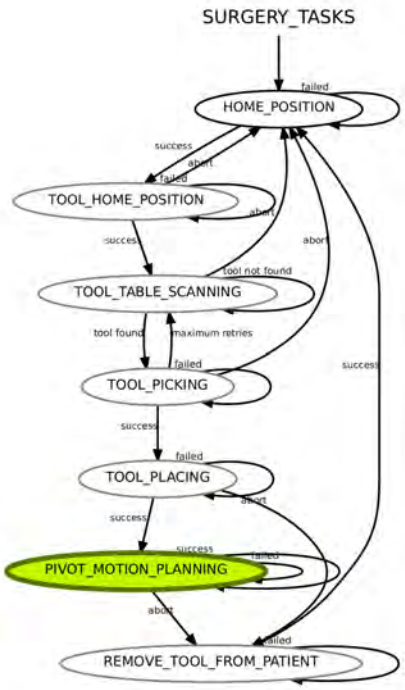

Figure 9.27: State machine status, shown in smach viewer

Chapter 10

Conclusions and Future Work

10.1 Conclusions

Robotic MIS has a large impact in improving surgical operations while being demanding and complex in its implementation. Due to the nature of the tasks the surgical robots must execute, that is to operate on patients, the implemented solutions must have very strict requirements in accuracy and quality assurance. The critical requirements that were studied in this thesis are discussed in the sequel.

Positional accuracy: The trajectories that the robot executes (especially the pivot trajectories inside the surgical site) must be executed with micrometer accuracy. This positional accuracy is required for both the end-effector as well as for the tip of the surgical tool. It is important to note, that due to the fulcrum effect a small error in the end-effector can cause a larger error in the position of the tool tip and vice-versa. Errors larger than several micrometers can cause serious surgical side-effects. The experiments of this thesis were conducted with the requirement of bounding the position error of the end-effector to $\pm 5\mu m$. The positional accuracy in trajectories outside of the patient body is not as strict and was increased to $0.5mm$.

Orientation accuracy: Orientation accuracy is also very crucial for both the end-effector and the tip of the surgical tool, and it affects the deviation error from the fulcrum point. Orientation errors can also have negative side-effects in the surgical operation and small errors in the end-effector can cause bigger errors in the tool tip and vice-versa. The experiments of this thesis were conducted with the requirement of bounding the orientation error of the end-effector to $\pm 5 \cdot 10^{-6}$degrees. The orientation accuracy in trajectories outside of the patient body, was increased to $\pm 5 \cdot 10^{-4}$degrees.

Time optimization: Time optimal trajectories are concerned with finding the paths, avoiding collisions and respecting the constraints as well as calculating the trajectories in the shortest amount of time as possible. This is very important, so that the robot can be responsive in real-time to the surgeon's commands and to be able to quickly and safely execute a given task. In all experiments a maximum of 5 seconds was specified for each path planning request. Although this time duration is significantly large, many experiment path planning requests were completed in less than a second (see measurements from 9.1, 9.2, 9.3.1).

RCM constraint: To satisfy the RCM constraint, all trajectories were designed using spherical coordinates with respect to the fulcrum reference frame. To make sure that the surgical tool satisfies this constraint an RCM error metric was used to measure the distance of the line of the long axis of the tool from the fulcrum point. This error was mostly measured for the insertion, retraction and line segment trajectories with very satisfactory results with the errors being in the magnitude of micrometers (see Figure 9.7). It is important to keep this error at a minimum to avoid exerting pressure to the patient's abdominal wall.

Interpolation accuracy: This requirement states that the number of interpolated waypoints of a path/trajectory should be enough such that the geometry of the path is not significantly distorted. In this thesis the interpolation is implemented in two stages; the first stage is the one related to creating distortionless trajectories and the second is a lower level interpolation made by MoveIt path planner when executing Cartesian line paths. For this second interpolation, a step distance is defined such that the actual executed path is as close to the line as possible (see Figure 6.2). In the simulations, the geometric interpolation was made using 20 or 50 sample points and the second interpolation was made using a step size of 1mm. For both parameters, a higher choice is recommended for more precise movements but with an extra cost of heavier computational requirements.

Collision avoidance: Path planning algorithms must take into consideration the environment obstacles and execute trajectories around them without colliding with them. In most cases this requirement was automatically satisfied from the MoveIt framework, where it was specified that each kinematics solution should be collision-aware. The second case that was not handled by MoveIt and was separately handled in the experiments was the elbow-up constraint. This constraint (see also 2.2.3.3) was purposefully imposed so that the robot arm's elbow would not collide with the mounting dock (the object with the trocars that simulates the patient) when executing pivot motions.

Suitable path planning algorithms: The MoveIt motion planning framework for ROS provides a wide range of path planning algorithms to choose from. The requirements for choosing a path planning algorithm were to find a collision-aware solution in the shortest time possible, respecting all constraints, without any replanning (if possible) and to provide under the same conditions the same results. In the first simulation study the two algorithms that were benchmarked were the RRTConnect and the RRT* algorithms and all other experiments were conducted using the RRTConnect algorithm, which provided satisfactory time results.

Repeatability & deterministic results: When studying robotic applications another is the robot's repeatability. The repeatability measures how "close" the measurements are with each other. A good repeatability is achieved when the measurements' standard deviation is as small as possible. In this thesis, all measurements were generated from running each simulation at least 10 times, in order to calculate the average and the standard deviation. Deterministic results in the measurements, in the context of this thesis, means that the measurements are not random and are predicted to be close to the average measurement and within a range of two or three standard deviations.

Environment layout and robot base position: Selecting the robot's position in the

operating room and setting up the objects' layout is quite important. The surgical tools' positions that the robot must approach, and the surgical site, where the robot must operate at, must be fully inside the robot's workspace and be well reachable with ease and flexibility. In the first 2 simulations, a layout was used that made it difficult for the robot to reach all surgical tools and trocar points. In the remaining simulations a different layout was used, where the robot was much closer to the tools and the surgical site and could execute the pivot trajectories with more flexibility.

Trajectory reachability: The designed trajectories, especially those of the pivot motions, must be in such a way so that the robot can fully follow them with ease and flexibility. The "flexibility" term means, to operate at points away from singularity points, away from the workspace boundary and without poses that make the robot to "self-fold" or even worse to self-collide.

Singularity avoidance: When in a singularity point, the robot loses one or more kinematic DoF and moving it would require large effort from the joints. This can result in surgical side-effects, as well as cause problems to the robot and should always be avoided. Singularity points are avoided from the MoveIt framework when calculating the paths, and are also avoided in the lower-level controllers of the robot's hardware. Singularity point approaching was also tracked using the robot's Jacobian and the manipulability index (see Figure 5.2).

Jump threshold: This parameter reduces the large unpredictable motions of redundant joints that could be a safety issue (in this case both for the patient and the operating staff). These undesirable motions can occur when the arm approaches singularities and needs to reorient the arm in order to continue following the trajectory. In simulation experiments this parameter was disabled to better study the path planning in all possible cases.

The aforementioned conclusions were drawn given the following **assumptions** that were taken into consideration to simplify, without loss of generality, certain simulation studies.

- The **fulcrum position and orientation** were assumed known and constant. All trajectories from the simulations were calculated using constant fulcrum reference frames, which according to the bibliography in more realistic scenarios is not the case, and the pose of a fulcrum point is usually estimated.

- The surgical tool appears fixed on the end-effector of the robot. In this thesis a higher level overview of **surgical tool grasping** was studied in order to better study the pivot trajectories separately without being affected by grasping forces and fulcrum-tool torques.

- The simulation environment features were assumed known and constant, which means that the pose of the surgical tools and the trocar were assumed known and their values were not extracted from the **Computer Vision node** that calculated these poses using a stereoscopic camera.

The goal of this thesis was to study in an **end-to-end approach**, all the stages involved in the perception, control and manipulation of laparoscopic tools, but **more emphasis was given to the pivot trajectories and the RCM constrained motion planning**.

Since this approach is very complex and involves many different algorithms, mathematics, techniques and technologies, it was very useful that each stage was studied separately and self-isolated to ensure that each stage performs well on its own (unit testing), so that later all stages can be integrated and studied together (integration testing). A higher level end-to-end pipeline that can be used for integration testing is proposed in the state machine of 9.7.

10.2 Future Work

Simulation and interaction with deformable bodies. In this thesis the laparoscopic tool is inserted through holes (trocars) in a mounting dock that simulates the patient. A more realistic approach would be instead of using such a rigid mounting dock, to use a deformable phantom object to study the interaction between the laparoscopic tool, the trocar and the patient's abdominal wall. Such a simulation is not feasible using ROS Gazebo simulation environment because it does not support collisions and physics with deformable objects. A framework that could be used with the help of a suitable ROS plugin to simulate deformable objects is the SOFA Framework.

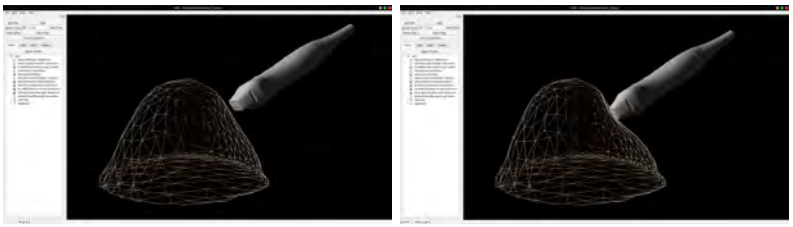

Figure 10.1: Deformable tissue/organ medical simulation - Simulation of breast probing using the SOFA Framework. Screenshots from running the repository at https://gitlab.com/altairLab/probe-tissue-simulation

Advanced visualization and Haptics feedback In this thesis, no medical imaging is used relevant to laparoscopic operations. It would be greatly beneficial to study pivot motions based on medical imaging or haptic feedback from endoscopic instruments. CHAI3D is an open source, platform agnostic framework for computer haptics, visualization and interactive real-time simulation which supports haptic devices.

Other ideas for future work that could improve and further explore the outcomes of this thesis are:

- Using the trajectories from Chapter 6 as building blocks for more complicated trajectories like suturing or cauterization
- Design pivot trajectories without assuming that the Fulcrum point is known and constant and use estimation methods (like a Kalman filter) to estimate its pose
- Multiple robot arm collaboration: designing pivot trajectories for two or more robot arms that collaborate for a common surgical task, for example one robot controls an endoscopic camera and the other robot controls the surgical tools

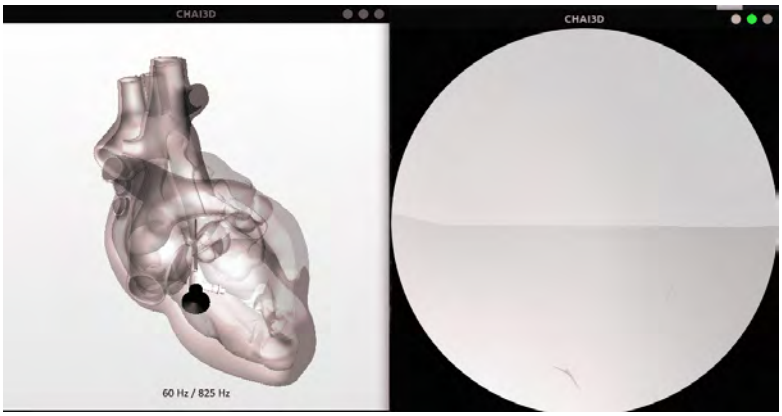

Figure 10.2: Heart endoscopy medical simulation using the CHAI3D framework. Screenshots from running the repository at https://github.com/chai3d/chai3d

- Applications of Machine Learning in Computer Vision: detection of laparoscopic tools

Appendices

.1 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/**

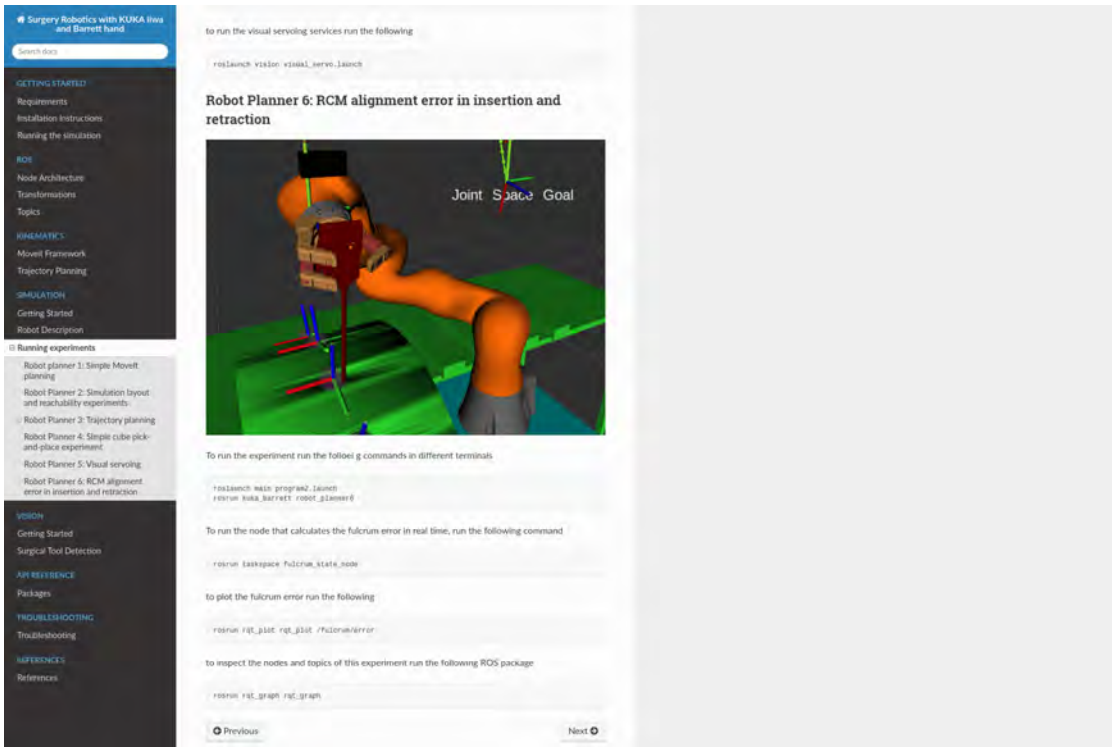

Figure 3: Documentation written with Sphinx and rosdoc_lite

.2 Mathematics

.2.1 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

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

.2.2 Cartesian to spherical coordinates

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

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

$$\varphi = atan2(y, x)$$

.2.3 Spherical to cartesian coordinates

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

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

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

Nomenclature

$\dddot{q}_i$	Angle jerk of joint i
$\ddot{q}_i$	Angle acceleration of joint i
$\dot{q}_i$	Angle velocity of joint i
$\hat{\mathbf{r}}, \hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\phi}}$	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
$\mathbb{H}$	Quaternions group. A four-dimenional space with basis $\mathbf{1}, \mathbf{i}, \mathbf{j}, \mathbf{k}$ that includes all quaternions. In the context of this thesis quaternions are used for 3D rotations instead of matrices from $SO(3)$
$\mathbb{J}$	Joint taskspace
$\mathbb{P}$	Pivot paths 3D taskspace, $\mathbb{P} \subset \mathbb{T}$
$\mathbb{R}^n$	Vector space of n-dimensional real-valued vectors
$\mathbb{R}^{n \times m}$	Vector space of all real-valued $n \times m$ matrices
$\mathbb{S}$	Surgical 3D taskspace
$\mathbb{T}$	Robot's Euclidean 3D taskspace
$\mathbf{C}$	Covariance matrix
$\mathbf{q}$	Angle positions vector of all joints
$\mathcal{C}$	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
$\tilde{O}_{Fi}$	Estimated origin point of fulcrum reference frame $\{i\}$
${}^{i-1}\mathbf{p}_{iO}$	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

O_F, O_{Fi} A generic origin point of fulcrum reference frame and origin point of fulcrum reference frame $\{i\}$ and estimated origin point of fulcrum reference frame $\{i\}$ respectively

q_i Angle position of joint i

s_i Shorthand notation for $sin\theta_i$

$SE(3)$ Special Euclidean Group. Vector space of all 3D transformations

$SO(3)$ Special Orthogonal Group. Vector space of all 3D rotations

τ Time constant, represents a specific time duration in a trajectory velocity profile or in a system controller

RCM Remote Center of Motion

TCP Tool's Center Point

List of Figures

List of Listings

List of Algorithms

List of Tables

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] Enrique Bauzano et al. “Control Methodologies for Endoscope Navigation in Robotized Laparoscopic Surgery”. In: *Eurobot Conference*. 2009.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] G. Cowley. “Introducing "Robodoc". A robot finds his calling–in the operating room”. In: *Newsweek* 120.21 (Nov. 1992), p. 86.

[13] Bassem Dahroug, B. Tamadazte, and N. Andreff. "3D Path Following with Remote Center of Motion Constraints". In: *ICINCO*. 2016.

[14] Bassem Dahroug, B. Tamadazte, and N. Andreff. "Task Controller for Performing Remote Centre of Motion". In: *ICINCO*. 2016.

[15] 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.

[16] 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.

[17] 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.

[18] Lin Dong and Guillaume Morel. "Robust trocar detection and localization during robot-assisted endoscopic surgery". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)* (2016), pp. 4109–4114.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] J. Funda et al. "Constrained Cartesian motion control for teleoperated surgical robots". In: *IEEE Trans. Robotics Autom.* 12 (1996), pp. 453–465.

[24] 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.

[25] 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.

[26] 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.

[27] 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).

[28] Caspar Gruijthuijsen et al. “Leveraging the Fulcrum Point in Robotic Minimally Invasive Surgery”. In: *IEEE Robotics and Automation Letters* 3 (2018), pp. 2071–2078.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] Corke P. I. *Visual Control of Robots, high-performance visual servoing*. John Wiley& Sons Inc., 1996.

[35] Craig J. *Εισαγωγή στη ρομποτική, Μηχανική και Αυτόματος Έλεγχος*. Τζιόλα, 2009. ISBN: 978-960-418-160-5.

[36] 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.

[37] S. Karaman and Emilio Frazzoli. “Sampling-based algorithms for optimal motion planning”. In: *The International Journal of Robotics Research* 30 (2011), pp. 846–894.

[38] 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.

[39] Parham Kebria et al. “Kinematic and dynamic modelling of UR5 manipulator”. In: Oct. 2016, pp. 004229–004234. DOI: 10.1109/SMC.2016.7844896.

[40] Zachary K. Kingston, Mark Moll, and Lydia E. Kavraki. "Sampling-Based Methods for Motion Planning with Constraints". In: *Annual Review of Control, Robotics, and Autonomous Systems* (2018).

[41] 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.

[42] 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.

[43] Chin-Hsing Kuo and Jian Dai. "Robotics for Minimally Invasive Surgery: A Historical Review from the Perspective of Kinematics". In: Jan. 2009, pp. 337–354. ISBN: 978-1-4020-9484-2. DOI: 10.1007/978-1-4020-9485-9_24.

[44] 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.

[45] 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.

[46] Sally K. Longmore, Ganesh R. Naik, and Gaetano Dario Gargiulo. "Laparoscopic Robotic Surgery: Current Perspective and Future Directions". In: *Robotics* 9 (2020), p. 42.

[47] Kevin M Lynch and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control*. English (US). Cambridge Univeristy Press, 2017. ISBN: 978-1107156302.

[48] 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.

[49] 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.

[50] 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.

[51] 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.

[52] 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.

[53] Kim-Doang Nguyen, Teck Ng, and I-Ming Chen. "On Algorithms for Planning S-Curve Motion Profiles". In: *International Journal of Advanced Robotic Systems* 5 (Mar. 2008). DOI: 10.5772/5652.

[54] 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.

[55] 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.

[56] 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.

[57] 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.

[58] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In: *ICRA Workshop on Open Source Software*. 2009.

[59] R. Woods R. Gonzalez. *Ψηφιακή Επεξεργασία Εικόνας*. Εκδόσεις ΤΖΙΟΛΑΣ, 2018.

[60] 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.

[61] *ROS/Concepts - ROS Wiki*. URL: https://wiki.ros.org/ROS/Concepts (visited on 09/19/2020).

[62] 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.

[63] 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.

[64] H. Shao et al. "A New CT-Aided Robotic Stereotaxis System". In: 1985.

[65] Bruno Siciliano et al. *Robotics: Modelling, Planning and Control*. 1st. Springer Publishing Company, Incorporated, 2008. ISBN: 1846286417, 9781846286414.

[66] Ioan Alexandru Sucan and Sachin Chitta. "Motion planning with constraints using configuration space approximations". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2012), pp. 1904–1910.

[67] I. A. Şucan and S. Chitta. *Moveit!* URL: https://moveit.ros.org/ (visited on 04/28/2020).

[68] V. Vitiello et al. "Emerging Robotic Platforms for Minimally Invasive Surgery". In: *IEEE Reviews in Biomedical Engineering* 6 (2013), pp. 111–126.

[69] 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.

[70] Haiwang Xu et al. "Laparoscopic Robot Design and Kinematic Validation". In: Jan. 2007, pp. 1426–1431. DOI: 10.1109/ROBIO.2006.340138.

[71] 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.

[72] 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.

[73] Xiaoqin Zhou et al. “New remote centre of motion mechanism for robot-assisted minimally invasive surgery”. In: *BioMedical Engineering OnLine* 17 (2018).

[74] Νίκος Ασπράγκαθος. *Ρομποτική*. Τμήμα Μηχανολόγων & Αεροναυπηγών Μηχανικών, Πανεπιστήμιο Πατρών, 2018.

[75] Ζωή Δουλγέρη. *Ρομποτική*. Κριτική, 2007. ISBN: 978-960-218-502-5.

[76] Κωνσταντίνος Μουστάκας. *Υπολογιστική Γεωμετρία και Εφαρμογές 3Δ Μοντελοποίησης*. Ανακτήθηκε την Κυριακή, 19 Ιουλίου 2020 από https://eclass.upatras.gr/courses/EE844/.