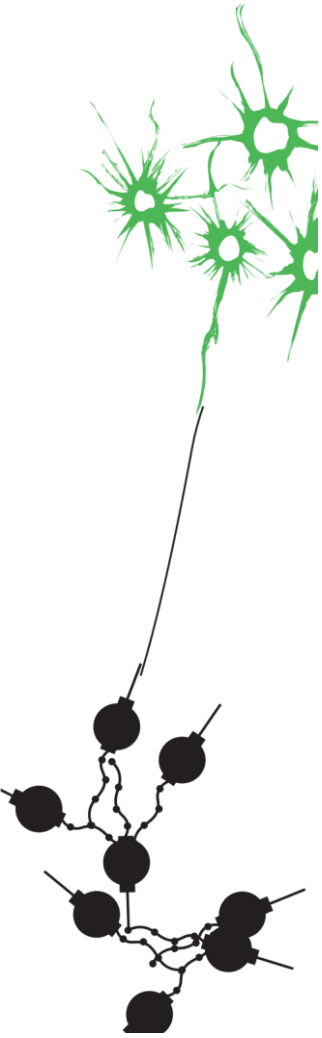


ROBOTICS
AND
MECHATRONICS

POSITIONING AND ORIENTATION CONTROL OF A NEEDLE-INSERTION MRI COMPATIBLE MEDICAL ROBOT BASED ON ROS USING VISUAL FEEDBACK

P. (Pamela) Shametaj

MSC ASSIGNMENT

Committee:
dr.ir. F. van der Heijden
dr. V. Kalpathy Venkiteswaran
dr.ir.M.Abayazid
G. Wardhana, MSc

April, 2021

Robotics and Mechatronics
EEMCS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

UNIVERSITY OF TWENTE. | TECHMED CENTRE UNIVERSITY OF TWENTE. | DIGITAL SOCIETY INSTITUTE

Abstract

Within the discipline of surgical interventions, there has been a focus on percutaneous needle insertions. The procedure usually involves the insertion of a needle or probe deep inside the patient's skin. These minimally invasive procedures are usually used for ablation, abscess drainages, biopsies or preventive purposes. The robot used in this project has been designed to be used as a patient mounted robot for irreversible electroporation (IRE) targeting pancreatic adenocarcinoma tumor cells. The project's scope is creating a functional first prototype that could be used for the insertion of electrodes during the IRE procedure, with a focus on creating a software framework that can be adapted to future changes in the physical robot design.

This thesis will skew from a traditional approach of a pure research or design thesis project and be a combination of two phases, Implementation and Design. This is due to the initial state of the project, where only the physical structure of the robot was available. In the first phase, an initial testing prototype upon which research could be performed was developed. With the use of ROS, a modular framework that integrated all the hardware components, calculated the robot kinematics and handled the communication within the software was designed. In the second phase, the project's focus changed to researching a specific design decision, that of the robot feedback. After analysing the available options, the decision of using non-MRI visual feedback was made. In this stage, the accuracy and reliability that vision feedback offers in terms of control were investigated. A calibration was proposed, and its ability to be performed with minimal errors through visual feedback was examined. The type of hardware that could provide the best visual feedback, as well as the tracking and detection algorithms that could provide the highest accuracy, were explored. Both research and experimentation were carried out in order to address these statements.

The Intel RealSense Depth Camera (D435i), a stereo solution, was implemented to provide visual feedback. Four different trackers, CSRT, MOSSE, KCF and Medianflow, were implemented and compared to one another through testing. The tracker with the most satisfying results KCF was then chosen to perform a robot calibration. In the end, the testing showed that in the current implementation, the use of visual feedback provided unsatisfactory results. However, visual feedback could still prove feasible when implementing the necessary updates, such as a combination of IR and Aruco markers for higher accuracy and a combination of KCF and Kalman Filtering for better tracking results.

Contents

1 Introduction

1.1 Context

Robotics in the field of surgical interventions is not a new discipline. The first recorded medical application can be dated as far back as 1985 where it was used for a brain biopsy. Recently, various medical robots have been developed for use in rehabilitation, telesurgery and needle guidance operations. (Monfaredi et al., 2015) The necessity for automation is because of the high accuracy needed for such procedures. (Bekku et al., 2014) One could also understand the importance of technology involved in such procedures by considering how repetitive, lengthy and accurate medical processes are. Despite requiring human supervision such tasks seem to be highly suited for robotic integration.

Furthermore, within the discipline of surgical interventions, there has been a focus on percutaneous needle insertions. The procedure usually involves the insertion of a needle or probe deep inside the patient's skin. These minimally invasive procedures are usually used for ablation, abscess drainages, biopsies or preventive purposes.(Patel et al., 2018; Bekku et al., 2014; Monfaredi et al., 2015; Dehghani et al., 2018) The current process involves doctors inserting needles manually, a process which is highly dependent on the physician's skill. When using medical imaging such as Xray, MRI and CT, the doctors themselves need to imagine the trajectory of insertion at the moment of extraction or ablation. Usually, such processes are lengthy and tough on both the patient and the doctors since it might require re-insertion if the needle or probe does not reach the target. With a lack of real-time feedback, this scenario is not uncommon. Recent technological advancements have allowed for the development of multiple specialised robotic systems used for needle guidance and insertion. (Patel et al., 2018; Bekku et al., 2014; Monfaredi et al., 2015; Dehghani et al., 2018) Using the specialised robotic systems brings many benefits in needle insertion procedures. Many robotic systems can calculate the trajectory of insertion needed to reach the tumor, thus reducing the errors that come from free-handing the procedure. Even a partially automatic needle insertion, where the needle is oriented and positioned by the robot but inserted by the doctor, can provide a more stable needle insertion and reduce the doctor's workload. Such improvements can lower the number of times a patient gets punctured making the process less painful.

The robot used in this project has been designed to be used as a patient mounted robot for irreversible electroporation (IRE) targeting pancreatic adenocarcinoma tumor cells. The novelty of the IRE technique sets this project apart from the already available technology. During the procedure an electrode is inserted in soft tissue. Pulsed electrical fields are then delivered to the cells, creating lethal nanopores in the plasma membrane to induce cell death. Different electrodes need to be inserted with distances between 10 and 20mm, depending on the tumour size. This method is considered a non-thermal ablation method. IRE uses a series of square high voltage pulses to target the cells. Rebersek et al. (2014) Although the electrodes used for clinical practice and in vivo experiments are either needles or plates, stainless steel or titanium electrodes will be used in this experiment. Usually, they are covered with an insulating material, except for a certain tip length. The exposed tip length of the electrode is described as the "active tip length" which delivers the pulses to the soft tissue.

Although the IRE method requires the accommodation of multiple needles, the first robot prototype will initially carry only one needle. In the future, new prototype designs that include multiple needles and specialised multiple needle holders will be explored. In this project, the scope will be creating a software framework that accommodates changes in design and positions and orients the singular needle to an external target. The main focus, goals and requirements will be discussed in detail in the coming sections.

1.1.1 Project Scope

In this section, the project's scope, and the general requirements, will be discussed. A review of the project's current state will also be carried out in order to obtain a clearer understanding of the areas that can be improved upon. This thesis will skew from a traditional approach of a pure research or design thesis project. The main reason behind this decision is the starting state of the project. In the initial state, the physical robot design had already been finalized and assembled. The actuation method, pneumatic actuation, was printed and integrated into the physical robot. However, the robot lacked the low level and high level control implementation to be a finished prototype. Initial implementation decisions needed to be made for both the hardware and software components. The decisions to be made were regarding the implementation and programming of a micro-controller, implementation and calculation of kinematics, as well as a software platform that facilitates the necessary communication between all the software components.

The project's scope is creating a functional first prototype that could be used for the insertion of electrodes during the IRE procedure, with a focus on creating a software framework that can be adapted to future changes in the physical robot design. The software can then be improved and optimized in the future phases of the project. Thus a necessary preliminary step for this thesis was the development of an initial testing prototype upon which research could be performed. In the second phase of this thesis, the project's focus will change to researching a specific design decision. The two project phases will be noted as the Implementation phase and the Design phase.

Possible directions were explored for the design phase, including; further optimisation of the software, adaptations to the physical design, comparison with a different actuation method, sensory feedback and target imagining. The decision was made based on the following questions; How does this research benefit the current state of the project, will it aid in the fulfilment of a functional prototype, and can it be built upon after the conclusion of this thesis. Other important aspects to keep in mind are the time and physical constraints. Can this research be concluded in the planned time frame, and is it feasible considering the current resources. By evaluating all the prospective directions and their feasibility, the chosen research direction was on designing the robot's sensory feedback.

1.2 First Prototype

This section will focus on introducing the first prototype, as well as looking at the current state of robot design and what can be improved further. An initial robot design was already present at the start of this project. The decisions behind the design and the designs capabilities will be introduced in the first subsection. In the initial prototype, no external sensors were present, thus in the second subsection, the sensor choices and the motivation behind each choice will be explored and explained.

1.2.1 Robot Design

The needle insertion robot was created with the goal of targeting pancreatic tumor cells. To be able to reach the pancreas, the robot should be positioned on top of the patient's body. Simultaneously, part of the procedure will be performed inside the MRI, such that the needle position and its distance from the tumor can be checked in real-time. Taking these situations in consideration, the following requirements were established. Firstly, the robot needs to be lightweight since it will be mounted on top of the patient. Secondly, in order to be used inside the MRI, the robot should be made with MRI compatible materials. Lastly, the space inside the MRI should also be taken into account. Even with a wide bore MRI the space inside is 70 cm wide with only up to 75 cm head-space, meaning the robot cannot take significant space and needs to be relatively small.

To achieve both the lightweight and MRI compatible requirements, all the robot links and joints are 3D printed using tough Polylactic Acid (PLA) material. The material is fully MRI compatible and light-weight, weighing only 1.24 grams per CM^3. (Herrmann et al., 2014) The first prototype, a parallelogram robot, is shown in figure 1.1. The robot has 4 DOF; translation in the x-axis, y-axis, rotation in the z-axis and y-axis. The rotation in y is a coupling between two x-translations. The robot dimensions are 10.8 cm in height and 26.1 and 20.8 cm in length and width. The robot is actuated with pneumatic stepper motors developed at the University of Twente. The motors are 3D printed and use the same materials as the rest of the robot, making them completely MRI compatible. This is very practical for prototyping while also being a low-cost solution. Initially, the robot was not equipped with any external sensors. Only the robot's physical structure was available at the start of the project, making this the first prototype to be explored. A more thorough description of the robot joints, links and limits will be explained in chapter 2.

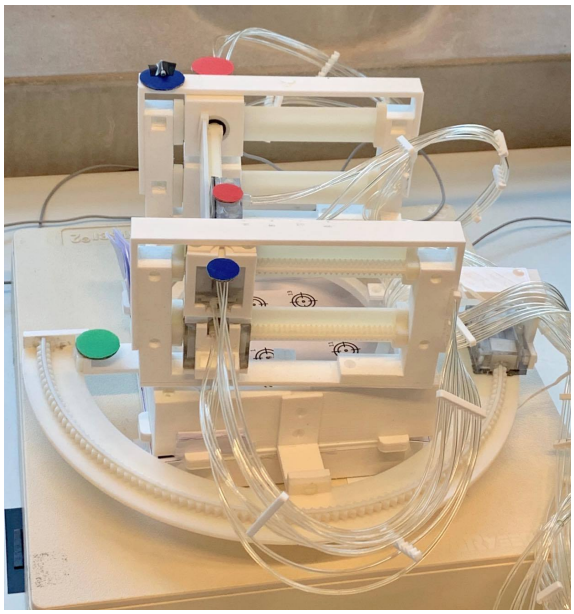


Figure 1.1: The robot prototype assembled.

1.2.2 Robot Feedback

A critical part of the project is the usage of external sensors for providing feedback for both the motors and the kinematic calculation. When choosing the type of feedback for such an application, the options are limited by the weight, accuracy, real-time implementation and material requirements. Although this is the first prototype, the feedback choice must adhere to the previously mentioned requirements. The current robot design and interface still have to be validated inside and outside of the MRI. Previous works on similar applications indicate that the two main options for feedback are vision and sensors. Within the sensor domain, two other options are presented; using MRI compatible sensors and using non-MRI compatible sensors. Non-MRI compatible sensors should then fulfill the requirements of not being bulky, not interfering with the current design, and be removable in order to test the robot design inside the MRI machine. On the other hand, the other option was to create an MRI compatible sensor as the ones available on the market are significantly expensive and exclusive. One such option for creating MRI compatible sensors is using optical fiber to create a fully MRI compatible encoder. This would be the optimal solution; however, it would shift the thesis's focus on just implementing an MRI compatible sensor for the current motors. It would also be quite challenging to create an optical fiber sensor that is small enough for the current design and does

not increase the number of cables. Because the motors used are pneumatic, the number of cables is already significant and disruptive to the movement. Adding more cables would increase the difficulty of building and testing new prototype and complicate the operation of the robot.

If using a sensor is not an option, it still leaves us with two vision feedback options; using a non-MRI compatible camera and using the MRI itself for feedback. Although using the MRI as feedback would fulfill all the requirements, which the previously mentioned options could not, it still cannot provide fast enough feedback. It is also an impractical method for testing multiple prototypes. On the other hand, using a non-MRI compatible camera could be a good way of providing a fast way to test the current design and interface of the robot. By providing real-time feedback, this implementation allows for a fast way to test ever-changing prototypes and may also be used for calibration outside of the MRI machine. Since the robot's motors can be operated in feed-forward mode at a safe frequency, this outside calibration method is possible. In conclusion, considering the positive and negative aspects, the usage of a non-MRI compatible vision setup is more viable in the current application than other methods.

1.3 Research Objective

The goal of this project is to create a functional prototype of the needle positioning robot. Since this is the first prototype of many to come, it means that a framework must be designed with the goal of it being capable of easily adapting to changes in the physical model. Thus a way to approach the trajectory planning and modelling of a hybrid serial-parallel needle insertion robot should be explored thoroughly. During this exploration, the proposed solution must fulfill requirements such as modularity and accuracy while keeping the implementation as simple as possible for the benefit of future research.

On the other hand, the accuracy and reliability that vision feedback offers in terms of control will also be investigated. Due to the nature of the motors, vision feedback should be sufficient. This comes as a consequence of the fact that the implemented stepper pneumatic motors provide no backlash. However, the motors move relative to their initial position and require a proper zero position calibration. Simultaneously, supervision is also required in case of error build-up or hardware malfunction. A literature review, prototype development, and experimentation were carried out to investigate those directions, calibration and supervision.

As mentioned previously, in order to position and orient the needle accordingly, a good calibration needs to be achieved. The possibility of keeping the design of the prototype independent from its sensory visual feedback is explored. However, specific questions need to be answered to finalize the implementation of such a feedback method. Firstly, how reliable and accurate is the usage of visual feedback? For example, can an accurate calibration with minimal errors be performed with just visual feedback? What hardware or camera could provide the best accuracy for such a problem? What tracking and detection algorithms provide the highest accuracy? Both research and experimentation will be carried out in order to address these questions. In a later phase, the whole implementation will be tested alongside the performed calibration.

In conclusion, the assignment's primary goal is to be able to control and position the actual prototype of the Needle-Insertion robot correctly, and conduct an analysis on the accuracy of the designed implementation in ROS by using visual feedback.

2 Current State

Several needle insertion robots were studied to understand technology's current state better and provide inspiration for this project. Consequently, this review focused on aspects such as the actuation of the robot, the method of feedback, the structure of software implementation, and the robot's kinematics implementation.

For this project, extensive research was already conducted in regards to the IRE technique. The previous research focused on determining the number, type and size of the electrodes necessary for the IRE procedure. However, this project's scope is having a functioning prototype that positions and orientations electrodes to the target position rather than the IRE procedure. In this section, the project's current state , which was already introduced in chapter 1 will be looked at further. The robot's design will be expanded on in terms of DOF and kinematic structure, along with the choice of actuation and choices made regarding hardware implementation.

2.1 Background

Available needle insertion robots were studied during a literature review, and a variety of technical and structural aspects of these robots were analysed. Since no examples of robots created for the purpose of IRE were identified, examples of robots created for the purpose of kidney of cryoablation Tokuda et al. (2018), prostate brachytherapy Orlando and Joseph (2017), cryotherapy of renal cancer Hata et al. (2016), for both kidney and liver cryoablation Watkins et al. (2016) and lastly, an example of robots not being constrained to a specific organ Hungr et al. (2016) were examined instead. However, none of these robots specifically targeted the pancreas, making it harder to take inspiration regarding the robot's workspace requirements and design.

Despite the fact that none of the robots were built for the same purpose, some were designed for the upper body and therefore have similar designs. One robot that was looked at for inspiration was a multimodal patient-mounted interventional radiology robot, evaluated under MRI guidance.(Ghelfi et al., 2018) The prototype is a light puncture robot which utilises kinematics with two serial kinematic chains connected in parallel. This approach is unique in that most other available needle insertion robots, such as Bekku et al. (2014); Patel et al. (2018); Hata et al. (2016), provide solely either fully serial or fully parallel robot solutions. In this ways it shares similarities with the first prototype design in this project. To overcome the complexity of such a robot, a combination of transformation matrices and geometric solutions was used. For the serial chains, just like the robot mentioned in the study Hata et al. (2016), the Denavit and Hartenberg (DH) convention is used. A similar approach could solve the kinematics of the current design by considering the two parallel chains as serial chains and equalizing the end effectors.

Another feature that some of the robots in the reviewed studies shared with this project was their MRI compatibility. By looking at the MRI compatible robots, two elements were looked at; the actuation method and the implemented sensors. The MRI compatible robot in the study of (Ghelfi et al., 2018) utilises a combination of pneumatic and piezoelectric actuation. The authors justify this decision by the claim that piezoelectric actuators are considered to be more accurate than available pneumatic motors. On the other hand, the pneumatic actuator is used to gripping the needle due to its high reliability. The claim that pneumatic motors' usage is not highly common despite being a fully MRI compatible solution can also be seen in the other reviewed studies. Hungr et al. (2016); Watkins et al. (2016) Another interesting direction to explore is the usage of sensors in an MRI environment, and for this reason, the study Gassert et al. (2008) was analysed. The authors Gassert et al. (2008) argue that if placed sufficiently away from the imaging region and equipped with adequate shielding and filtering, conven-

tional electrically powered sensors may be used. This includes sensors such as camera-based measurement systems, strain gauges, accelerometers, Hall effect sensors, force/torque sensors, and optical encoders. The paper also provides particular examples of MRI sensors used in previous projects, such as a hydrostatic water pressure transducer to infer grip force, a modified sphygmomanometer bladder connected to a mercury column, and fiber optic sensors. The latter ones are said to guarantee high MRI compatibility, a simple and flexible installation, and negligible transmission delay and signal loss even over long distances. The works from Hungr et al. (2016) also shows some examples of MRI compatible sensing. Here the optical tracker (Certus, NDI, Waterloo, ON, Canada) was used together with fiducial markers.

The final aspect that was researched was how the studies approached the organization of the software framework. The first study that was closely looked at was the one of Hungr et al. (2016). In this research, the robot was programmed to insert a needle through the patient's skin anywhere in the upper body, without being restricted to a specific organ. The research details the particular interfaces that are used in the general implementation: QT used for the GUI, Visualization Toolkit (VTK) used for visualisation of tumor, Insight toolkit (ITK) used for image analysis, and Computer Assisted Medical Intervention Tool Kit (CamiTK) used for communication. The software was used respectively for the following tasks: 1) robot controller, 2) robot/scanner registration software, 3) user interface software, and 4) a communication channel to transfer MRI images and control signals between the scanner room and the control room. This implementation shows a similar structure to that of ROS but specialised for medical applications. The other study that was analysed was a continuation of the work from Hungr et al. (2016), by Patel et al. (2018). Here a GUI based application providing both joint spaces as well as task-space control was developed and added to the previous work. The GUI is designed such that the clinician can observe all the desired robot status information, such as the current robot pose versus its desired pose. The study by Tokuda et al. (2018) also concentrated on the overall software framework design. The designed system consists of navigation software and a needle guide device controller running in parallel. The controller is in charge of controlling the actuators and monitoring the encoders and sensors. It receives control commands and parameters (e.g. registration transform and target coordinates) and sends hardware status (e.g. current orientation of the needle guide and hardware error information) to and from the navigation software. The target coordinates received are translated to individual actuators' displacements by computing the inverse kinematics and passing to the PID control to control individual actuators. The encoder readings are converted to the needle guide's orientation guide and sent back to the navigation software.

During this review it was noted that the actuation most commonly used in MRI compatible robots was piezoelectric actuation. All the papers prioritized the high accuracy offered by piezoelectric motors over the high MRI compatibility of the pneumatic motors. Therefore, a pneumatic motor needs to provide high accuracy to be chosen over a piezoelectric motor. It also became clear that to provide fully MRI compatible feedback; one can either develop an MRI compatible sensor or use MRI imaging as feedback. The MRI compatible sensors used in the papers which are available for purchase are low in number and highly costly. Imaging, on the other hand, was used often and was shown to be successful but was only tested on the tracking the needle tip. Additionally, such an implementation requires the constant use of an MRI machine during the experimentation phase, which is infeasible for the current project. All the statements in this review reinforce the analysis done in the previous chapter, where it was decided for an non MRI compatible camera. The general framework of the projects and their implementation structure was another aspect that was paid significant interest to. In terms of the general software framework, the use of standalone programs or especially designed software was preferred, while available robot frameworks such as Robot Operating System (ROS) were not used at all. When designing or choosing the software, no particular notice was paid to issues such as modularity. On a more structural level of implementation, it can be seen from

the papers that the actual needle control software framework was divided into three steps: position, orientation, and manual or automatic insertion of the needle. However, multi-needle control was not mentioned in any of the papers.

After presenting the current state of the available needle insertion robots, the robot's current state used during this project will be explained. The decisions behind its design and implementation will be backed by the knowledge gained from this section.

2.2 Robot

2.2.1 Design

This section will focus on familiarizing the reader with the design of the first prototype and elaborate on the design choices made. Initially, the kinematic structure - including the robot joints, their limits, links and DOF - will be introduced together with the chosen needle insertion method. Besides textual explanations, diagrams and models will be utilized to help with visualization of the overall robot structure.

The developed robot has 4DOF, two of which are reserved for translations in the x- and y-axes and the other two for end-effector rotations in the z- and y-axes. The rotation in z is achieved by the bottom joint and is used to transport the entire top platform. The joint's position can be seen more clearly in figure 2.1. The rotation in y is performed by coupling the two x translations of the top frame and the bottom frame as seen in figure 2.2. By using this method, more support is given to the needle for insertion. However, this means that the translation of the needle in z is also coupled with this x orientation. Although this does not pose an issue, as the insertion itself is entirely manual, and the needle is not transported while the robot is moving, this coupling of axes should still be taken into account. This method allows for a broader workspace since the distance from the upper and bottom frame does not become a limitation. Each joint has upper and lower limits either for rotation or translation. The bottom joint, which can be seen in figure 2.1, can rotate from -1.34 radians to 1.34 radians. The upper and bottom side prismatic joints translate from -35 mm to 35 mm. The middle prismatic joint translates from -31.25 mm to 31.25 mm.

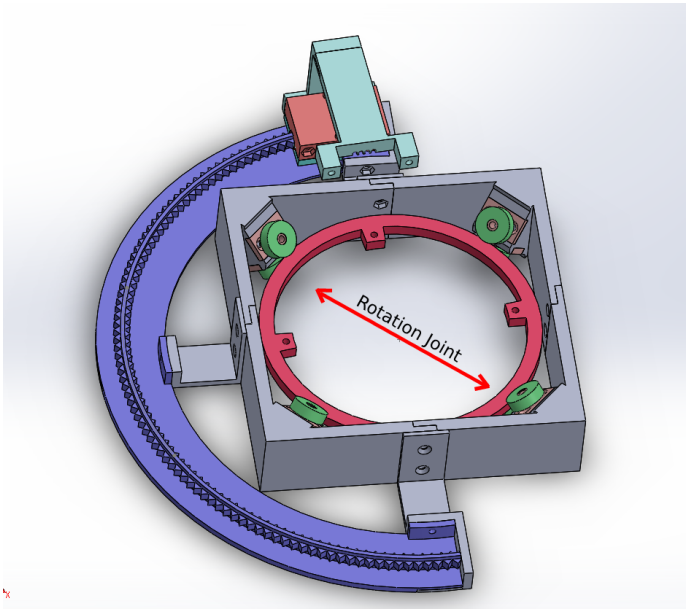


Figure 2.1: The rotation joint found at the bottom base of the robot.

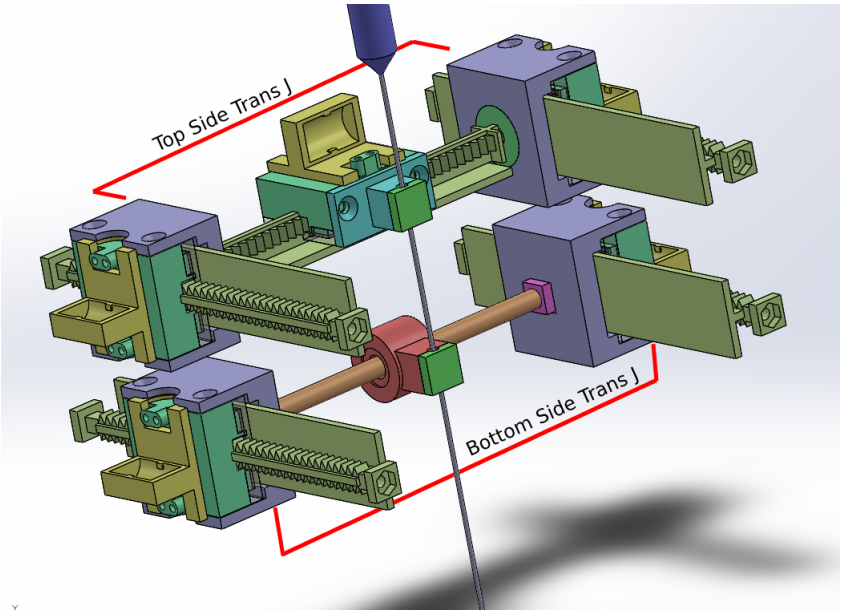


Figure 2.2: Two translation joints that provide orientation of the needle. As can be seen in the figure the translation joint is in fact two coupled translation joints instead. (J stands for Joint)

To accommodate both needle's insertion and orientation, passive rotational joints were used in the upper and bottom frames as seen in figure 2.3. It needs to be noted that the bottom joint is both a passive rotation joint and a passive translation joint. In total, there are eleven joints, four of which are passive while the other seven are actuated. From the seven actuated joints, four of them are coupled and are mirroring each other. An example of this can also be seen in figure 2.2.

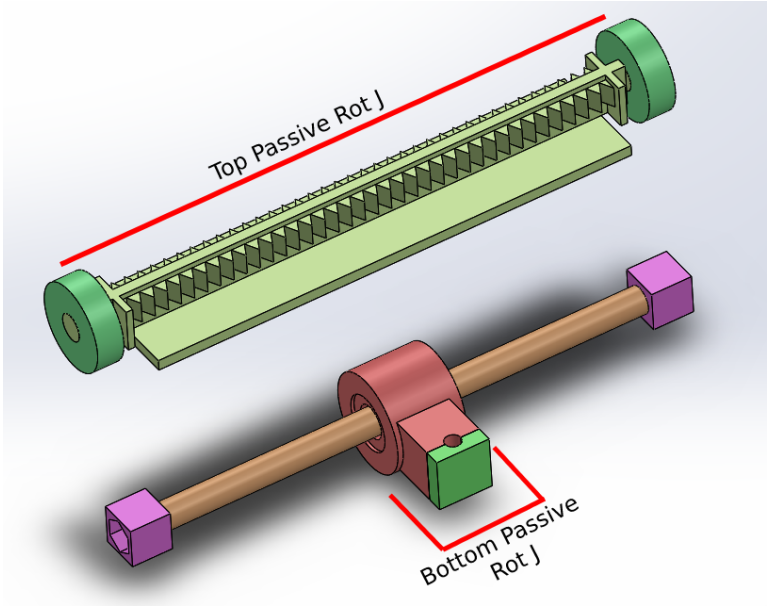


Figure 2.3: Passive joints of the robot needed for orientation of the needle. (J stands for Joint)

The links have significantly complicated shapes as can be seen in figures 2.1,2.2,2.4 and are integrated into the motors. For example, most of the upper links are either the linear guide of the motor or the motor itself with extensions. All the links are very lightweight and fully

MRI compatible, as they are 3D printed using plastic material. As mentioned beforehand, the needle is inserted after links and joints' final position has been reached. The needle is taken as an end-effector simply for calculating the final kinematic position. Dynamically, the upper translation joint movement and the bottom translation joints are not effected by the needle. This manual insertion of the needle has been chosen for two reasons; a proper needle holder needs to be designed but is beyond the scope of this project, and a higher orientation can be achieved by manually inserting the needle. There is no finalized needle holder design in the current prototype, but simply a 3 mm diameter opening in the extensions of the links on the top and bottom as seen in figure 2.5. This means that only electrodes of slightly smaller than 3mm diameter can be accommodated in the current design.

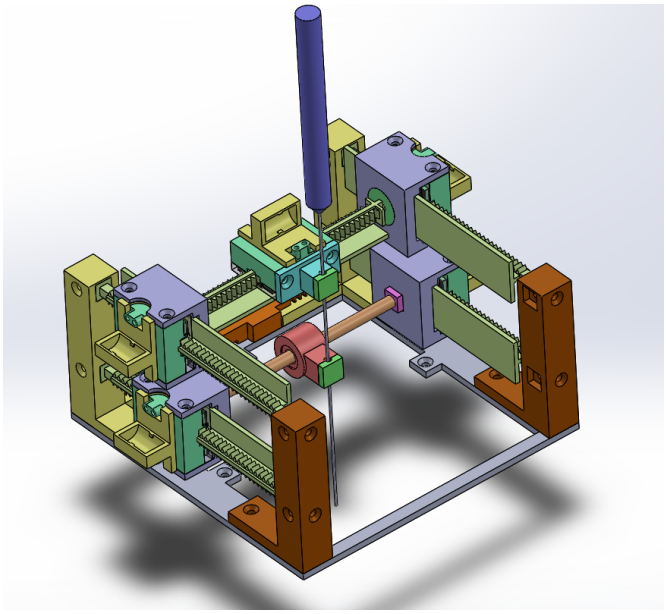


Figure 2.4: All links and joints of the robot excluding the rotation joint and bottom link.

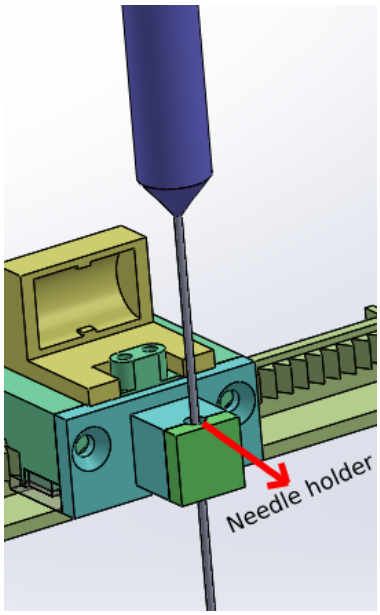


Figure 2.5: Needle inside the needle holder.

2.2.2 Actuation

In the current implementation, pneumatic motors are used to actuate the robot. The actuation method was mainly chosen because of the MRI compatibility of the motors. The decision to use these motors was further validated through the analysis of other available MRI actuation methods. The authors of the studies examined stated that piezoelectric motors were usually favoured over pneumatic motors.Hungr et al. (2016); Watkins et al. (2016) The reasoning behind this claim was stated to be the lack of accuracy in the available pneumatic motors. However, the motors chosen for this application overcome that limitation by providing high accuracy, especially when driven at lower frequencies.

The chosen pneumatic motors were produced at the University of Twente by Groenhuis et al. (2018) with the goal of being used for applications that require MRI compatibility. The motors are 3D printed linear stepper motors with the name T-26. A 3D model of the motor can be seen in figure 2.6. T-26 is composed of two pistons surrounded by a 3D printed housing that pushes against a rack. As a result of the motors comprising the link's physical structure, the length of the wedges is dependent on the required size of the links. The rack is a wedge mechanism comprised of tiny teeth of 2.5mm on both sides. The tooth size of the wedge mechanism is what determines the step resolution. The teeth provide piston's grip to progress into different states, which then move the motor forward or backwards. The pistons have four housing chambers. Two chambers are needed for each piston, with one residing on the top and one on the bottom, as shown in figure 2.7. The housings of the motor are pressurised in an orderly manner by having each piston move back and forth in the double-acting cylinder. This is done to create the stepping movement that moves the robot forward and backwards. Only one piston, the one which was pressurized formerly, can be fully engaged to the rack. During movement, the most recently pressurized piston temporarily releases its grip, allowing the other piston to fully engage to the rack .(Groenhuis et al., 2018) Out of this motion, five consecutive motor states exist in the linear stepper, displayed in figure 2.8.

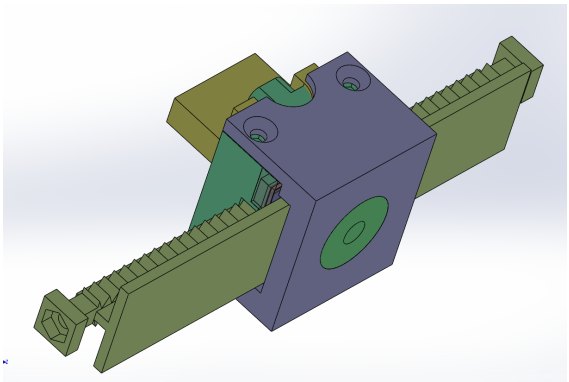


Figure 2.6: 3D model of the T-26 motor taken from the AutoCad model.

Four pneumatic tubes and two pneumatic valves are needed for each motor o pressurize the chambers of the pistons. The valve serves as the mechanism that opens and closes the pneumatic connections. In the current hardware implementation, only eight valves are used. This is because four motors were coupled in order to limit the number of pneumatic tubes, as seen in figure 2.9. The valves used can be seen in figure B.5 in appendix C. The sharing of valves is done to reduce the amount of actuation and number of pneumatic tubes. As can be seen from figure B.5, there is already an overwhelming amount of tubes going into the robot. Such a construction can limit robot movement, increase overall weight while making it harder to troubleshoot and fix leaks. To control all the valves - and consequently, the motors - a 2600

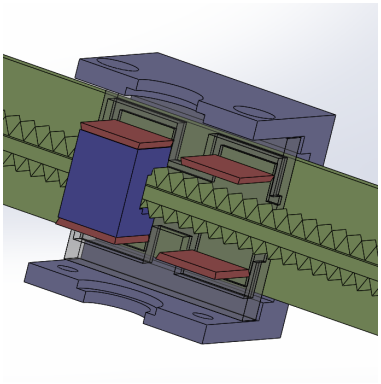


Figure 2.7: T-26 motor cross section taken from the 3D AutoCad model.

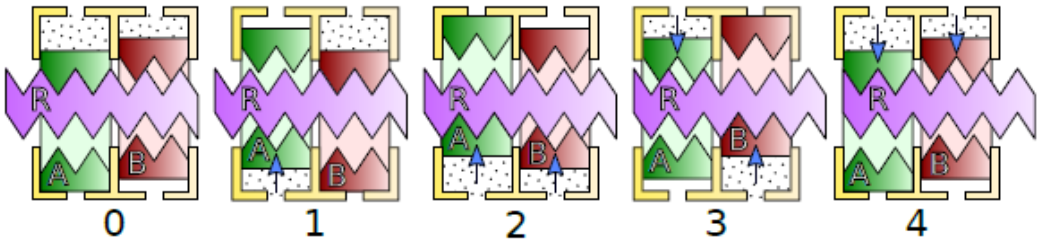


Figure 2.8: T-26 motor states taken from paper Groenhuis et al. (2018).

mega Arduino micro-controller is used. To connect the valve with the digital Arduino pins, an additional circuit is constructed as seen in figure B.4 in appendix C.

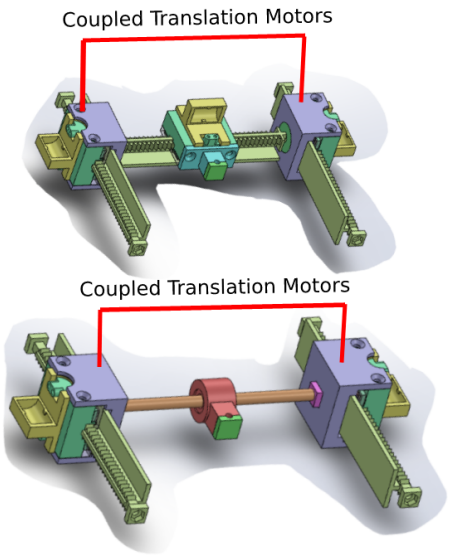


Figure 2.9: The coupled motors utilising only one valve.

During this chapter, the physical structure of the robot in its initial state was addressed. The existing state of needle insertion robots was also investigated. The following chapter will go through a more in-depth review of the thesis's objectives as well as the decisions taken during the Implementation and Design phases.

3 Analysis

3.1 Implementation Work

This section will concentrate on the first stage of the thesis, Implementation. The project's goals and requirements, as well as the necessary practical implementation will be discussed in detail. The available options for each decision will be noted, and each choice's reasoning will be argued.

3.1.1 Software framework

The main goal of the Implementation phase is to create the first functioning prototype. A low-level micro-controller needs to be programmed accordingly and software calculating the forward and inverse kinematics has to be developed. Lastly, all the software components need to be integrated seamlessly under one communication software structure. When attempting to achieve the Implementation phase's objectives, specific requirements must be noted as well. The first and most critical requirement is the development of a modular software framework. The current prototype will be the first of several, implying that the robot architecture will undergo numerous changes during its development cycles. In the future, the robot will include more than one needle; thus, the ability to plan for multiple end-effectors needs to be available. New hardware and software can be introduced in future prototypes. As a result, a software framework capable of integrating a wide range of external software and hardware must be built. The final requirement is the availability of seamless communication between the software units. In the following sections, these requirements will be addressed.

Two options are commonly available when creating software for robotics applications; using one's own implementation or using the Robot Operating System (ROS) platform. Creating a specialised communication platform is a more straightforward option. Just as the name suggests, this option would give the creator the option to choose its own preferred scripting language as well as their preferred communication protocols, making the platform prototype specific. Nevertheless, implementing a personalised framework is a time-consuming task when dealing with a great number of software and external devices. The communication between the different software needs to be developed and optimized, creating difficulty when implementing or testing new hardware. Since modularity and updating of the prototype are important aspects of this project, this communication aspect is problematic.

On the other hand, ROS has become a common choice when writing software for robots. The framework provides tools and libraries created by specialists worldwide and is used to simplify the task of creating complex and robust software for robots. Its main goal is to provide fast prototyping and efficient communication. As a result, it can be used to write general-purpose software, which can be continuously built upon in future stages. Such a solution is ideal for this project because it can provide a stable foundation for a constantly evolving robot. Simultaneously, through ROS, one can integrate hardware, third-party software and visualisation tools quickly.

After analysing the two available options for creating robotics software, ROS was chosen as the prototype's preferred framework. This implementation provides easy integration with visualization tools such as Rviz and Gazebo, using hardware such as Arduino and a multitude of external sensors. The framework takes care of all the communication through ROS messages and allows for both C++ and Python scripting. Most importantly, ROS is an open-source framework that is constantly tested and optimized.

The second aspect to address in terms of the modularity requirements is the kinematic implementation. Two popular options are available in the robotics field, the usage of Matlab and

ROS-MoveIt. The two platforms provide a variety of solvers, mainly regarding inverse kinematics.

Matlab is programming and numeric computing platform generally used to analyze data, develop algorithms, and create models. When implementing robotics solutions, a combination of Matlab, Simulink and Toolboxes is used. Simulink is a model-based programming tool that provides a simulation environment to textual Matlab programming. The software packages used in combination with Simulink are the Robotics system Toolbox , Symbolic Math Toolbox, Simscape Multibody and Fuzzy logic. The forward kinematics can either be performed by creating multibody models through Simscape or implementing one owns the mathematical solution. The inverse kinematics, on the other hand, can be solved through the four Toolboxes. The solution can be either analytical by using geometry or numerical by calculating the geometric Jacobian.

MoveIt is an open-source robotics manipulation platform for developing commercial applications, mostly used for planning kinematics and executing trajectories. The software offers its methods of collision checking, one for the robot with itself and one for the robot with its environment. The forward kinematics is done within MoveIt with the help of the transformation (TF) package. TF is actually utilized to figure out the transformations between the robot links. Together with the joint positions from the joint state publisher, these transformations are utilized by MoveIt to calculate the forward kinematics. To solve the inverse kinematics, three MoveIt solvers are available: KDL, IKFast and TrackIk. The KDL kinematics plugin wraps around the numerical inverse kinematics solver provided by the Orocos KDL package. IK Fast is a powerful inverse kinematics solver provided within ROSen Diankov's OpenRAVE motion planning software. It can analytically solve the kinematic equations of any complex chain and generate language-specific files (like C++) for later use. However, the IKFast plugin generator tool does not work with <6 degrees of freedom arms. TRAC-IK is an inverse kinematics solver developed by TRACLabs that combines two IK implementations via threading to achieve more reliable solutions than common available open-source IK solvers. TRAC-IK concurrently runs two IK implementations. One is a simple extension to KDL's Newton-based convergence algorithm that detects and mitigates local minima due to joint limits by random jumps. The second is an SQP (Sequential Quadratic Programming) nonlinear optimization approach which uses quasi-Newton methods that better handle joint limits.

Both options, Matlab and ROS, provide fast solutions and are implemented in a similar modular manner, both fulfilling the modularity requirement. It even is possible to implement a combination of ROS and Matlab by using an additional Matlab toolbox. This allows Matlab to be used alongside the ROS platform that had already been chosen for hardware and software integration. However, Matlab is not an open source software, requires an additional toolbox to communicate with ROS, and has a more complex collision checking method. To keep the software structure more reliable, open source and seamless, MoveIt was chosen for the overall implementation of the kinematics and TRAC-Ik was the chosen inverse kinematic solver.

3.2 Design Work

3.2.1 Vision Hardware

The chosen robot feedback, a non-MRI compatible camera, and the reasoning behind this decision were already discussed in this project's introduction chapter. A non-MRI compatible camera is capable of providing real-time feedback and a fast way to test the current and future designs. The camera also allows for calibration outside of the MRI machine. The latter is done by taking advantage of the fact that the robot at hand has the ability to be driven in feedforward when used at a safe frequency. The setup's ability to achieve high accuracy calibration and tracking feedback still need to be tested.

In the current implementation, the used joint feedback is actually a measure of distances. To be able to measure the distance effectively, the 3D coordinates of the objects of interest need to be available. The depth component can be used to get rid of false detections by removing the background information. To measure the z-coordinate, either a monocular camera or a stereo camera can be used. There have been multiple studies on achieving high accuracy depth and measurement approximation from monocular cameras. However, the papers focus on a non-real-time application. In addition, the monocular approximation methods need objects of reference and particular calibration methods, are dependent on learning the environment and require specialised equipment.Kang et al. (2015); Wei Liu et al. (2016); Mane and Yangandul (2016); Bui et al. (2018) On the other hand, the stereo camera can provide highly accurate results.Min-jeong Kang et al. (2008); Mustafah et al. (2012a,b); Meng et al. (2018); Bui et al. (2018); DANDIL and ÇEVİK (2019) The downsides of stereo cameras are their bulkiness, long processing time, the lack of accuracy because of improper camera alignment, and complicated calibration.Mane and Yangandul (2016); Bui et al. (2018) To be able to bypass the mentioned limitations of monocular implementations while prioritising high accuracy and real-time feedback, a stereo camera was chosen to be implemented.

After the choice of stereo versus mono camera setup was made, one other question remained. Will the camera be sourced from the range of already available stereo cameras, or will it be developed from two mono-cameras? Before the question is answered, the requirements need to be set. First, it needs to be mentioned that a high depth value is not within the scope of requirements. Instead, a close-range implementation would provide satisfactory results. Secondly, high accuracy is a requirement; thus, the camera must have a high resolution to be able to fulfill the accuracy requirement.

There are multiple advantages of building one's own stereo camera setup. Parameters such as baseline, resolution, and focal length can be chosen to achieve the appropriate depth levels. Simultaneously, using two mono-cameras can be cost-effective, considering that purchasable stereo cameras come with specialized software and can be highly costly. On the other hand, an arduous amount of time needs to be spent on perfecting the setup. From a hardware standpoint, the cameras must be correctly aligned and positioned, and the setup must be robust, since misalignment and hardware adjustments may cause major accuracy errors. Mane and Yangandul (2016); Bui et al. (2018) Besides the hardware aspect, specialized software needs to be built and optimized. A camera calibration method must be created, as well as feature extraction and stereo matching techniques. During these phases, it needs to be made sure both of the video feeds are aligned. On the other hand, available market cameras can provide a sturdy product that is already self-aligned and tested. A significant number of these cameras also come with their own on-chip camera calibration as well as optimized software. By looking at both implementations' positive and negative aspects, a decision was made to use a purchasable camera. In the current project the sturdiness, ease of implementation regarding hardware alignment, and code efficiency was prioritized over customizability. Recently, purchasable stereo cameras have also become more easily available at a lower cost and a higher resolution.

The next step is looking at available stereo cameras. A comparison of their specs can be seen in the table 3.1. Looking at the camera's specifications, the camera chosen for this implementation is Intel RealSense Depth Camera (D435i), United States. The Real Sense Camera D435 offers a higher field of view, a good minimum depth, high resolution for both depth and RBG feeds with up to 90 fps. After further research the depth accuracy error also stands at less then 2 percent up to 2m.

Cameras	Intel® RealSense™ D435	ZED Stereo Camera	Tara	Intel®RealSense™ D415
Depth Technology	Active IR	Embedded	Embedded	Active IR
RGB module	Yes	Yes	No	Yes
Depth Resolution	1280 × 720	2208 x 1242	752 x 480	1280 × 720
Dimensions(mm)	90×25×25	175x30x33	100x30x35	99×20×23
Min Depth(m)	0.28	1.5	0.5	0.45
Depth FOV	86° × 57°	96° x 54°	60° H	64° × 41°
Frame Rate(fps)	90	15	60	90
Price(dollar)	179	449	169	149

Table 3.1: A comparison table of the stereo cameras

3.2.2 Vision Software

In this subsection, the reasoning behind the software decisions will be discussed. The main goal in this phase of the project is to perform a calibration method for the motors and be able to track their movement as feedback for the kinematics. Thus in terms of vision, the goal is to create a method that is able to track the robot joints of the motor, determine their 3D position, and calculate their distance from the joint's initial(zero) position. The first requirement to be noted is the software's ability to track and identify multiple markers at the same time.

To be able to fulfil this requirement, as well as to achieve faster and more stable results, an algorithm combining both detection and tracking is utilized. The tracker takes over for the detector by providing faster results and searching on a smaller region of interest.(Pallapotu, 2019; Yao et al., 2016; Tannús, 2020; Lehtola et al., 2017) This is due to having previous knowledge of the targets and their previous position. The trackers also deal better with partial occlusions. (Pallapotu, 2019) Simultaneously, the detection is used to initiate the trackers and get rid of the accumulated error since tracking algorithms can accumulate a small amount of error through multiple frames. (Pallapotu, 2019)

To be able to make a choice on which tracker could provide the best solution, the current setup needs to be looked at. The detected objects are not expected to change in number from frame to frame, could be moving significantly fast, do not have a great change from the first frames in terms of size, and most probably will never be fully occluded. At the same time, the most crucial requirement will be to keep up with its movement and the ability to achieve accurate results. When considering these requirements, four trackers were selected. In regards to tracking discriminative method trackers were mainly looked at. These target tracking algorithms are known to provide real-time tracking by simplifying the time consumption calculation. Correlation Filters were one the most popular discriminative methods mentioned in literature. (Gong et al., 2020)

The first selected tracker is the Discriminative Correlation Filter with Channel and Spatial Reliability (CSRT). "The CSRT tracker joins the channel and spatial reliability concepts to discriminative correlation filters tracking. The spatial reliability map adjusts the filter support to the part of the object suitable for tracking." (Tannús, 2020) In such a way, one can enlarge the region of interest (ROI) and improve the tracking of non-rectangular objects.(Tannús, 2020; Ullah et al., 2019; Biswas et al., 2019). To localise the object reliability scores are used as feature weighting coefficients. These scores reflect the channel-wise quality of the learned filters.(Tannús, 2020) The CSRT tracker is known for its accuracy and the ability to track differently shaped objects. However, a drawback of the tracker is that it prioritizes accuracy over speed and does not do very well with occlusions.(Biswas et al., 2019)

The Kernelized Correlation Filters (KCF) algorithm is the second chosen tracker. It uses the shift-invariance property of Fourier transform (FFT) and Inverse Fourier Transform (IFT) to increases the speed of the tracker. Simultaneously, it does have a lightweight implementation by exploiting the large overlapping regions between various positive samples. (Raghava et al., 2020) The tracker is known for good reporting of tracking failures, as well as a satisfying compromise between speed and accuracy. This tracker's drawbacks are its inability to recover from full occlusions and lack of efficiency when dealing with objects that change in size. Overall the tracker is favoured by many papers as well as in comparison to the other selected trackers, which are mentioned in this chapter. (Charalampaki and Malamos, 2017; Ullah et al., 2019; Raghava et al., 2020; Rani et al., 2017; Lehtola et al., 2017)

Median flow is the third chosen tracker. In this algorithm, a region of interest (ROI), represented by a bounding box, is used as input. A grid is then generated from this bounding box, and its points are tracked from frame to frame. Then the sparse optical flow is used to estimate the movement of the object of interest. Here the object of interest is assumed as a combination of points moving synchronously. The quality of the predictions is estimated, and for each object point, an error is calculated. Half of the best predictions are then used to estimate the displacement of the object. (Raghava et al., 2020) Median flow is known for its speed and does a considerably good job at reporting failures; however, it has a hard time dealing with large jumps in motion, fast movements and abrupt appearance changes. (Charalampaki and Malamos, 2017; Ullah et al., 2019; Raghava et al., 2020; Rani et al., 2017; Lehtola et al., 2017)

Minimum Output Sum of Squared Error (MOSSE) is the final chosen tracker. MOSSE was initially created for solving the single-object tracking task. Initially, a ROI/tracking window is selected where the object of interest is centred. The object is then tracked by correlating the filter over a searching window in the next frame. The maximum value in the correlation output is looked at to estimate the object's position in the next frame. (Shen et al., 2018) The tracker is known for its high speed due to the computational efficiency of correlation filters. (Yao et al., 2016) MOSSE is mentioned many times in the literature as a good choice whenever one needs pure speed and is considered more robust to variations in deformation, scale and occlusion. However, many mention its high inaccuracy and failure to track when there is a more significant change in direction or pose. Overall in terms of accuracy, it seemed to perform worse than the CSRT and KCF trackers. (Ullah et al., 2019; Biswas et al., 2019)

3.2.3 Chapter conclusion

In this chapter, a detailed analysis of the thesis's two phases was given, the Implementation phase and the Design phase. The goals and requirements were noted for each phase, and the motivation behind various design decisions was explained. This chapter's subsections serve as an introduction to the practical work that will follow in the coming chapters. So far, the motivation behind the implementation phase was discussed, and the answer to one of the research sub-questions was provided. In the following chapters, the practical implementation of both phases will be elaborated on in detail. The Implementation phase chapters are the Kinematics chapter and Planning Framework, while the Design phase chapter is called Robot Feedback.

4 Kinematics

Kinematics in robotics uses geometry to study the movement of robots manipulators. The robot's links are modelled as rigid bodies while joints are separated traditionally into revolute and prismatic joints. The idea behind studying the kinematics of the robot is to be able to understand the relationship between the dimension of the joints and links and the placement of the end-effector in space, as well as the relationship between velocities in the joints with the help of the Jacobian matrix.

The goal of the kinematics in this project is reaching specific targets in the workspace. The focus of this project will be on positioning and orienting the needle to achieve this. The speed of the end-effector and the forces applied to it will calculated, since the needle's insertion is kept manual and is performed after the joints have moved to their respective values. The end-effector has 4-DOF in terms of position and orientation, as well as an additional DOF created by pushing the needle in the patient's body. The DOF coming from pushing the needle needs to be considered when planning for the kinematics, because one of the needle rotations is coupled with the z translation. Therefore, to have a precise understanding of the final position and orientation that can be reached by the needle, 5-DOF will be taken into account.

4.1 Kinematic Diagram

The first thing that will be studied at is how the robot is be expressed in terms of its kinematics. In the current case, there are three scenarios, each with their own benefits and limitations. An example diagram of the first scenario can be seen in figure 4.1. In this kinematic diagram, the bottom joint is connected in serial with the rest of the frames. Two serial chains comprising the upper chain and the bottom chain are then connected in parallel. The upper chain connected to the top of the needle and the bottom chain to the tip. This approach would be beneficial when considering a static needle, one that is inserted at the beginning of the implementation. However, that is not the case in the current design, as the insertion of the needle is performed at a latter stage to facilitate for a higher needle orientation. In the second scenario is to consider the robot is considered as a serial chain. Only the top joints are considered in the kinematic calculation, as seen in figure 4.2. The bottom y translation joint is considered as an actuator for the rotation. In order to account for the needle being pushed inside the body, an extra z translation joint is added. The problem behind this approach is that it makes it harder to check if the position is reachable, while also complicating the overall framework implementation. The third scenario considers the robot as a tree structure with two serial chains as seen in figure 4.3. Two end-effectors are considered in this scenario. The kinematics is then solved for both of the chains with the same end-effector configuration. It needs to be noted that a z translation is added for both of the chains, to account for the needle being pushed into the patients skin. The benefits of such an implementation are the reachability of a bigger workspace, the ability to plan for needle insertion, and assurance that the position is reachable. A negative aspect of such a method is that it requires a manual needle insertion. However, this does not pose a problem for the current implementation.

The third scenario of implementing each chain separately was chosen for this setup. This implementation can be seen in figure 4.4, which displays all the joints of the two chains and their respective links.

4.2 Traditional Implementation

There are two main ways of implementing kinematics in the traditional sense without using ROS. Denavit-Hartenberg (DH) and the use of twists. DH notation is more generally available for standard industrial robots and is used by most commercial robot simulation and program-

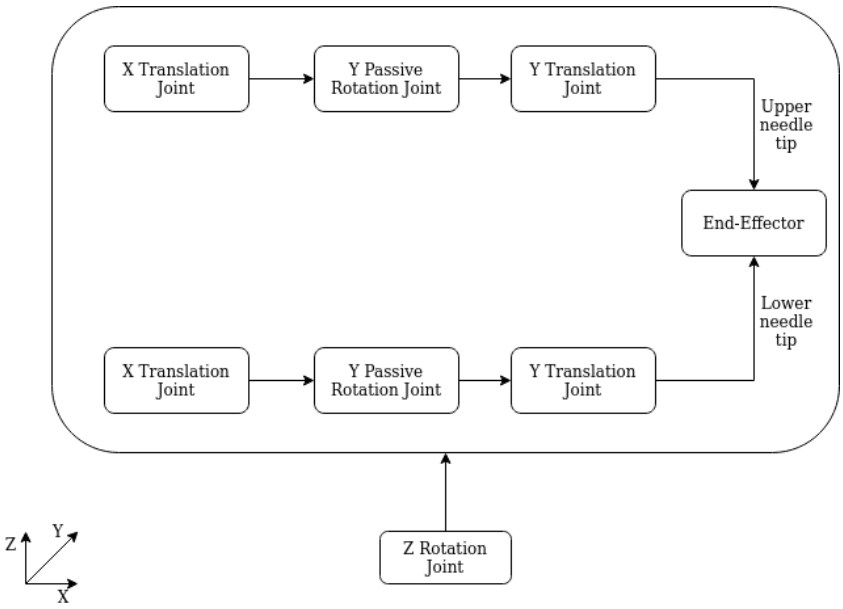


Figure 4.1: Scenario one of the kinematic structure.

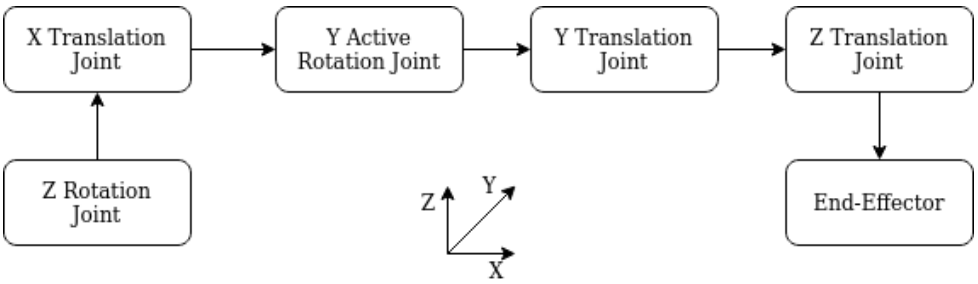


Figure 4.2: Scenario two of the kinematic structure.

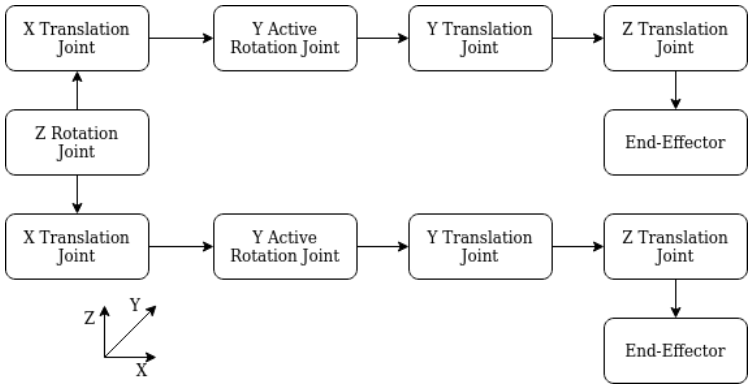


Figure 4.3: Scenario three of the kinematic structure.

ming systems. The DH method is achieved by setting a list of parameters for each link and then constructing the homogeneous transformations between frames using these parameters. These parameters are obtained by applying a set of rules in order to specify the position and orientation of frames attached to each link of the robot. For the traditional implementation of this project,the kinematics were solved using the product of exponentials and twist repre-

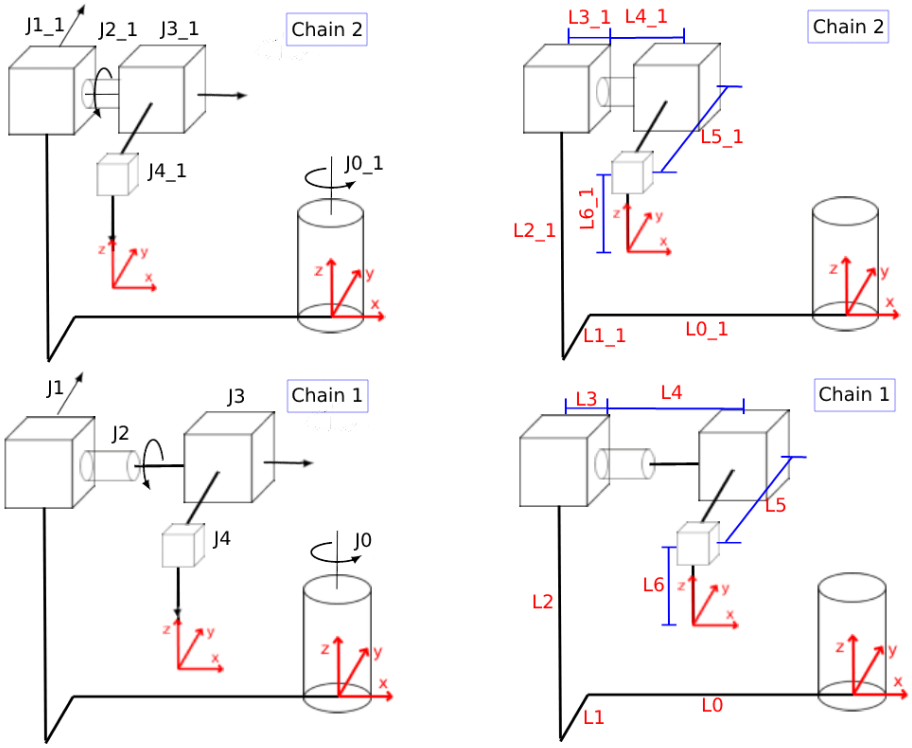


Figure 4.4: Kinematic diagrams of the two chosen kinematic chains. The diagrams show both the joint of the two chains and their respective links.

sentation instead of the DH notation. The advantage of using this method over DH is that one only needs two set of frames; that of the base and end-effector. The small number of frames combined with the geometric meaning of twists makes the chosen method superior in terms of simplicity and practicality. In the traditional implementation, the following limitations were taken in set to comply with the requirements of the set up. To simplify the problem at hand and be able to apply the product of exponentials, the robot kinematics were expressed in terms of an open chain with extra constraints in one of the joints. Thus, the chain's joints are the bottom rotation joint, translation in x, rotation in y and translation in y. The bottom actuated joint is then calculated through geometry.

4.2.1 Forward Kinematics

The first step in calculating the end-effector matrix is noting the homogeneous transformation from the base to the end-effector at zero configuration. The two frames are chosen as seen in the previous figure 4.4. The same matrix stands for both of the chains, while the only changing parameters are the velocity vector values. The respective vector values are further explained in equation 4.2.

$$H^0_{ee}(q=0) = \begin{bmatrix} 1 & 0 & 0 & -v1 \\ 0 & 1 & 0 & -v2 \\ 0 & 0 & 1 & -v3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

$$\begin{aligned} v1\_chain1 &= (L0 - (L3 + L4)) \\ v2\_chain1 &= L1 + L5 \\ v3\_chain1 &= L2 - L6 \\ v1\_chain2 &= (L0\_1 - (L3\_1 + L4\_1)) \\ v2\_chain2 &= L1\_1 + L5\_1 \\ v3\_chain2 &= L2\_1 - L6\_1 \end{aligned} \tag{4.2}$$

Next, the unit twists are calculated for all the joints. Only the translation unit vector is needed for the translational joints since the rotational unit twist is zero. The twist has a w rotational component and a v linear component. When dealing with a prismatic joint, the rotational component w is zero, and only the v component is needed. The general format can be seen in equation 4.3. The rotational joint has a rotational component w and a linear component v, which is a cross product between a chosen q vector and the rotational vector w. The q vector is chosen along the joint's axis to denote the distance from the world axis to the joint. The general format can be seen in equation 4.4.

$$T = \begin{bmatrix} 0 \\ \vec{v} \end{bmatrix} \tag{4.3}$$

$$T = \begin{bmatrix} \vec{w} \\ \vec{q}x\vec{w} \end{bmatrix} \tag{4.4}$$

The final unit twists are then calculated using the formats mentioned in equations 4.3 and 4.4 by looking at figure 4.4. The twists of the top chain (1) can be found in 4.5,4.6 while the twists of the bottom chain (2) can be found in equation 4.7.

$$T_1^{0,0} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} T_2^{0,1} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -L0 \\ -L1 \\ L2 \end{bmatrix} T_3^{0,2} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ L2 \\ L1 \end{bmatrix} \tag{4.5}$$

$$T_4^{0,3} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -(L0 - (L3 + L4) \\ -L1 \\ L2 \end{bmatrix} T_5^{0,4} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -(L0 - (L3 + L4) \\ -(L1 + L5) \\ -L2 \end{bmatrix} \tag{4.6}$$

$$T_1^{0,0\_1} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} T_{2\_1}^{0,1\_1} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -L0\_1 \\ -L1\_1 \\ L2\_1 \end{bmatrix} T_{3\_1}^{0,2\_1} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ L2\_1 \\ L1\_1 \end{bmatrix} \tag{4.7}$$

$$T_{4\_1}^{0,3\_1} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ L3\_1 + L4\_1 - L0\_1 \\ -L1\_1 \\ L2\_1 \end{bmatrix} T_{5\_1}^{0,4\_1} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ L3\_1 + L4\_1 - L0\_1 \\ -L1\_1 - L5\_1 \\ -L2\_1 \end{bmatrix} \tag{4.8}$$

In the end, the exponential formula is used to calculate the final relation between the end-effector and the joint values. The formula can be seen in equation 4.9. The same formula is used for both of the chains. The end-effector is expressed in the form of a rotational vector in combination with a linear vector as seen in equation 4.11. To find the values of the passive joints, the two chains are equalized.

$$H_{ee}^{0}(\theta) = e^{\tilde{T}_1^{0,0}\theta_1} e^{\tilde{T}_2^{0,1}\theta_2} e^{\tilde{T}_3^{0,2}\theta_3} e^{\tilde{T}_4^{0,3}\theta_3} e^{\tilde{T}_5^{0,4}\theta_4} H_{ee}^{0}(0) \tag{4.9}$$

$$H_{ee}^{0}(\theta) = e^{\tilde{T}_{1\_1}^{0,0\_1}\theta_1} e^{\tilde{T}_{2\_1}^{0,1\_1}\theta_2} e^{\tilde{T}_{3\_1}^{0,2\_1}\theta_3} e^{\tilde{T}_{4\_1}^{0,3\_1}\theta_3} e^{\tilde{T}_{5\_1}^{0,4\_1}\theta_4} H(0) \tag{4.10}$$

$$H_{ee}^{0}(\theta) = \begin{bmatrix} [R_{ee}^{0}] & \vec{v_{ee}^{0}} \\ 0x3 & 1 \end{bmatrix} \tag{4.11}$$

4.2.2 Inverse Kinematics

There are multiple ways to calculate the inverse kinematics for a robot. One of the common methods of resolving such a calculation is using the Jacobian matrix. The Jacobian is a representation of the velocities present in the manipulator. This matrix is then calculated for both of the chains. The geometric Jacobian is calculated by using figure 4.4. The Jacobian columns for chain one can be seen in 4.12, 4.13, while the columns for chain two can be seen in 4.15,4.16. Where the [exp1 exp2 exp3] vector is the cross product of w and q in equation 4.14 for chain one and [exp_1 exp_2 exp_3] vector is the cross product of w_1 and q_1 in equation 4.17.

$$J0 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} J1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -sin(J0) \\ cos(J0) \\ 0 \end{bmatrix} J2 = \begin{bmatrix} cos(J0) \\ sin(J0) \\ 0 \\ exp1 \\ exp2 \\ exp3 \end{bmatrix} \tag{4.12}$$

$$J3 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -sin(J0)cos(J2) \\ cos(J0)cos(J2) \\ sin(J2) \end{bmatrix} J4 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ -sin(J2) \\ cos(J2) \end{bmatrix} \tag{4.13}$$

$$\vec{w} = \begin{bmatrix} cos(J0) \\ sin(J0) \\ 0 \end{bmatrix} \vec{q} = \begin{bmatrix} cos(J0)[L3 - L0] - sin(J0)[J1 - L1] \\ sin(J0)[L3 - L0] + cos(J0)[J1 - L1] \\ L3 \end{bmatrix} \tag{4.14}$$

$$J0\_1 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} J1\_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -sin(J0\_1) \\ cos(J0\_1) \\ 0 \end{bmatrix} J2\_1 = \begin{bmatrix} cos(J0\_1) \\ sin(J0\_1) \\ 0 \\ exp1\_1 \\ exp2\_1 \\ exp3\_1 \end{bmatrix} \tag{4.15}$$

$$J3\_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -sin(J0\_1)cos(J2\_1) \\ cos(J0\_1)cos(J2\_1) \\ sin(J2\_1) \end{bmatrix} J4\_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ -sin(J2\_1) \\ cos(J2\_1) \end{bmatrix} \tag{4.16}$$

$$\vec{w}\_1 = \begin{bmatrix} cos(J0\_1) \\ sin(J0\_1) \\ 0 \end{bmatrix} \vec{q}\_1 = \begin{bmatrix} cos(J0\_1)[L3\_1 - L0\_1] - sin(J0\_1)[J1\_1 - L1\_1] \\ sin(J0\_1)[L3\_1 - L0\_1] + cos(J0\_1)[J1\_1 - L1\_1] \\ L3\_1 \end{bmatrix} \tag{4.17}$$

The next step would be calculating the pseudo-inverse Jacobian. With the help of the pseudo inverse, the joint changes can be calculated.

$$J^+ = (J^T J)^{-1} J^T \tag{4.18}$$

The kinematics calculation noted down in this chapter exists to get a better understanding of the robot's kinematics, as well as how the calculation can be implemented in the traditional manner. Despite the calculations of this chapter, the traditional implementation is not used in the software implementation of this project.

4.3 ROS Implementation

The first thing to be considered is that we will only solve our kinematics for positioning since the motors receive the position value directly. The speed of the motors at the moment will be kept at a particular safe frequency. Both forward and inverse kinematics will be done for the robot. At the beginning of chapter 3, three kinematic chain approaches were discussed. The one chosen for the ROS implementation is the third case, thus considering the robot as two separate serial chains. Taking advantage of the way the robot is built and its joint limitations, solving for the same position and orientation separately can still bring us an acceptable solution. The first chain will be that of the first revolute *Joint_0*, top translation *Joint_1_1* , top rotation *Joint_2_1*, upper translation *Joint_3_1* and needle connected through a translation *Joint_4_1*. The second chain is simply comprised of the bottom translation *Joint_1_2*, bottom rotation *Joint_2_2*, bottom translation *Joint_3_2* and the needle connected by a translation *Joint_4_2* as well. All the implementation can be seen in figure 4.5. The reason why the initial rotational joint was not included in the calculation of the second chain is because the bottom chain is also affected by the bottom rotation *Joint_0*. The kinematics are calculated for two end-effectors simultaneously; the end-effector of the upper chain and the end-effector of the bottom chain as can be seen in figure 4.6. Both of the needles are identical. The needle is included as an end-effector twice to allow for separate kinematic calculations for each of the chains. In this implementation, all the joints are considered active when planning the kinematics. However, the passive joints (*Joint_1_1*, *Joint_1_2*) are not sent to the hardware microcontrollers interface. The joint values passed to the hardware interface are the bottom rotation joint, the top and bottom x translation joints, and the middle y translation joints, as seen in figure 4.5.

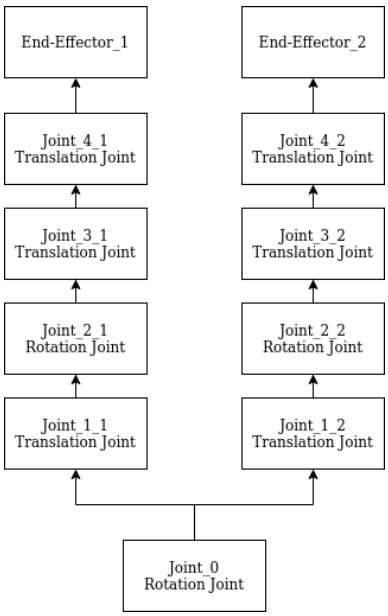


Figure 4.5: Diagram of the joints of the robot.

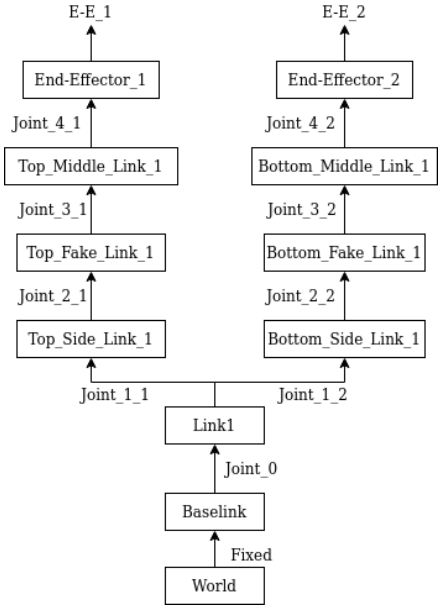


Figure 4.6: Diagram of the full implementation including both links and joints of the robot.

The first step in creating a proper kinematics implementation by using ROS is by creating a Unified Robot Description Format (URDF) model. URDF is a ROS specific XML format for representing a robot model. The file stores the kinematic structure of the robot, its appearance through geometric shapes and meshes, as well as other information such as joint limits, link masses and types of joints. The other format needed is the Semantic Robot Description Format (SRDF); this format saves the kinematic chains and uses them for solving kinematics. The URDF and SRDF store the mentioned parameters, which are then used by MoveIt. The forward kinematics is done within MoveIt with a combination of the joint state publisher and robot state publisher as well as the transformation (TF) package. TF is actually utilized to figure out

the transformations between the robot links. These transformations together with the joint positions from the joint state publisher, are utilized by MoveIt to calculate the forward kinematics.

The solver chosen in this project for solving the inverse kinematics is the Track_. The package is an inverse kinematics solver developed by TRACLabs that combines two IK implementations via threading to achieve more reliable solutions than common available open-source IK solvers. The package combines two different solvers. One is a simple extension to KDL's Newton-based convergence algorithm that detects and mitigates local minima due to joint limits by random jumps. The second is an SQP (Sequential Quadratic Programming) nonlinear optimization approach which uses quasi-Newton methods that better handle joint limits. By default, the IK search returns immediately when either of these algorithms converges to an answer.

This chapter introduces to what method is used for the implementation of the forward and inverse kinematics, as well as what external packages and software make the execution possible. In the coming chapter, the specifics of the implementation will be explained in detail as well as the overall framework.

5 Planning Framework

A major focus of this project is creating a general interface that connects the user input for the needle position and orientation to the feedback, robot simulation and actual hardware. From the figure 5.1 one can see the general overlay of how the robots interface works. The first step of this implementation is sending a setpoint. At the moment, the framework has been designed with two options; one where the user can input an endpoint with a position and orientation, and another where the user can use the GUI by guiding the needle to a physical marker in the robot space. The option to directly input a position and orientation has been made possible such that in the future, the whole process can be semi-automated. By doing so, the tumor information can be sent in the robot's coordinates, and the doctor can guide the needle towards the tumour using the desired orientation and position. In the future, the process can become entirely automated by directly sending the data from either from the MRI script to the planning script.

The second step is the actual planning, which is done through the MoveIt package. A script is used to call the planner, which solves the inverse kinematics for both of the end-effectors in each chain. Identical position and orientation end-effector values are sent to the interface in order to solve the inverse kinematics. If planning for both of the chains succeeds, then the value is sent through a trajectory controller. When using a simulation such as Gazebo, a controller from a ROS package called *ros_control* is used. The interface diagram when using Gazebo can be seen in figure 5.2. This package sends commands to the joints in the simulation and reads the joints' current value in the simulation to feed it back to the MoveIt package. On the other hand, if the connection directly to the robot is used, a separate script (hardware interface) transforms those commands into something that the hardware can understand. These values are sent through ROS, as ROS messages, to the Arduino interface. The Arduino interface then uses the motors' library to drive the motors to the right position using a preset velocity.

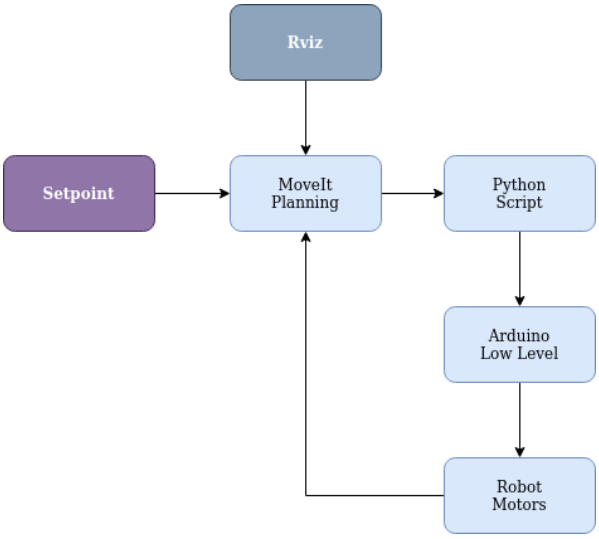


Figure 5.1: Diagram of the robot overall framework when connected to the real hardware.

In the next sections, the interface and its script components will be explained in detail.

5.1 Robot URDF

The creation of a URDF model is the first step to a proper MoveIt implementation. The URDF stores the kinematic structure of the robot, its appearance through geometric shapes and

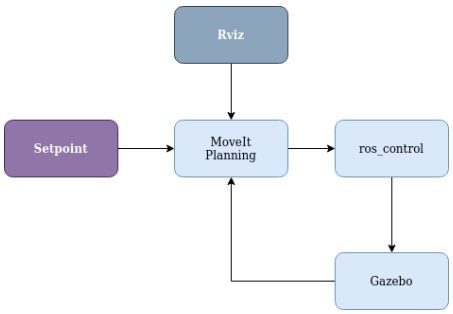


Figure 5.2: Diagram of the robot overall framework when using the Gazebo simulation.

meshes, as well as other information such as joint limits, link masses and types of joints. The object types within the URDF format are link and joint, equivalent to the traditional robotic concepts of link and joint. There are six types of joints within the URDF format:

- fixed with no motion
- revolute with rotation along a single axis
- continuous with rotation around a single axis with no limits
- prismatic with translation in one dimension
- planar translation in two dimension
- floating with unlimited motion

In this robot, only the basic revolute and prismatic joints are used. All the joints have upper and lower limits mirroring the real-life hardware and, thus move along only one axis. The unit of translation is in meters, while the unit for rotation is in radians. The robot expressed in URDF format can be seen in figure 5.3.

All the robot elements are created with respect to a world coordinate. The first link of the robot is the base link which is then connected to Link1 as seen in figure 5.3 through a revolute joint, *Joint_0*. In terms of rotation, the joint has an upper limit of -1.5 radians (-85 deg) and a lower limits of 1.5 radians (85 deg). From Link1, two translation joints can be seen; *Top_Side_Link_1* and *Bottom_Side_Link_1*. Two translation joints are then coming out from the top and the bottom of the first link as shown in figure 5.4, which shows the whole robot in RViz. The side links are, in fact, connected by two joints with Link1; but, the motors controlling those two joints are connected to the same valve. Because they cannot be controlled separately, they are considered as one joint. This goes for both the top translation link and the bottom translation link. In the future, the possibility of controlling them separately should be made possible. In this way, one can correct for small inaccuracies and discrepancies that can be found between the coupled joints. A combination of a rotational and prismatic joint is present in both the top chain and the bottom chain. Because such a joint does not exist in the URDF format, a fake empty link is created in between those joints, as seen in figure 5.3. In both the top and bottom chains, two translation links equipped with a needle holder extension needed for needle insertion are present. They are shown in figure 5.3 and are labelled as middle links. The top middle link is, in fact, just the motor body and its casing. To facilitate the kinematics and the actual position of the needle tip, two needles of the same size are connected through the needle extension by a prismatic joint, *Joint_4_1* and *Joint_4_2*. As mentioned before, the needle insertion in the real robot will be manual. Therefore, the needle cannot be considered rigidly connected in the kinematics. Thus an empty link is created at the tip of the needle to simulate the insertion. All

the mentioned links seen in figure 5.3 are created with their own transformation frames, which are attached to the centre of their 3D bodies.

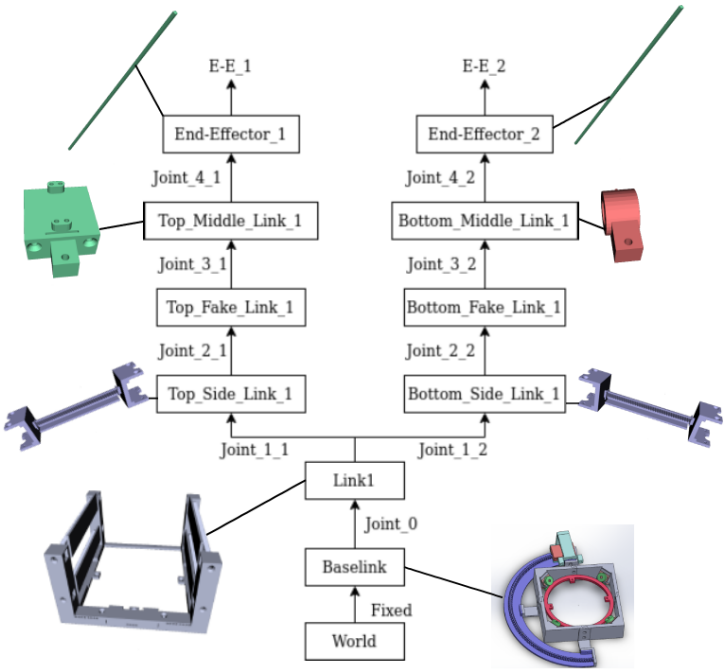


Figure 5.3: Diagram of all the robot links and their 3D models.

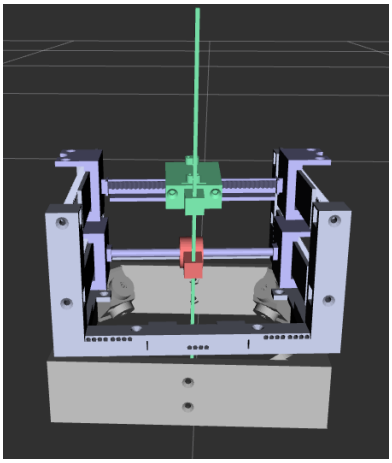


Figure 5.4: The full 3D model of the robot in RViz.

5.2 Gazebo

Robot simulation is an essential tool in every roboticist's toolbox. A well-designed simulator makes it possible to rapidly test algorithms, design robots, perform regression testing, and train AI systems using realistic scenarios. Gazebo offers the ability to accurately and efficiently simulate robots in complex indoor and outdoor environments. It is a robust physics engine with high-quality graphics, and convenient programmatic and graphical interfaces. What makes

Gazebo stand out is also its very involved community that continuously upgrades its capabilities as a toolbox.

Gazebo will be used to simulate the robot's movement. To make the simulation possible, either a URDF or a Simulation Description Format (SRDF) script is needed. Both URDF and SRDF are utilized as their description formats provide different benefits. Three simulation model versions of the robot were made. The reason behind these versions was to be able to simulate the robots real-life environment as accurately as possible. The first open loop version without the needle was made to ensure the inertias of the robot were correct. The closed-loop version is what should be aimed for to have a proper realistic implementation. However, it was found that in the current state of the software, it is not possible to have a fully functional closed-loop implementation. In future prototypes, a customised plugin should be written such that the motors move according to the controller in closed-loop form. The open-loop version with two needles is what the current implementation utilises.

In all three models, the main script is in the URDF format, while the SDF tags were used mainly in the closed-loop model. These tags are useful when one wants to add extra Gazebo specific functionalities or unsupported functionalities from the URDF. For links to be able to be displayed in Gazebo, they need to have a proper mass and inertia within the <inertia> and <mass> tags of every link. By adding the <gazebo> tags within the link, one can convert stl files to dae files for better textures. By adding a <joint> tag, one can set proper damping and control parameters. The latter was useful for making these joints actuated. All the scripts have an extra link in the current implementation that rigidly connects the robot to the world. Because of how the robot is built and its inability to have composite joints, fake links were also utilized within the URDF. To be able to display these joints within gazebo, low inertias were used instead.

Another benefit of using Gazebo is its easy connection with ROS. Through gazebo one is able to publish joint states and control the movement of the robot. To make the joints actuated, an external plugin has to be added to URDF. The main plugin used in the current project is that of *gazebo_ros_control*. In combination with this, plugin <transmission> tags are also added for each actuated joint. The tag specifies which joint is being actuated, the type of transmission and the type of hardware interface being used. There are three types of hardware interfaces; effort, velocity and position. In this implementation, the physical motors take the position as input, so the hardware interface chosen is that of position control as well.

After the joints and their transmission have been specified, the next step is setting up a ROS controller that can send commands. The real robot takes in position commands. These controllers will be used to connect the rest of the planning with the simulation and the simulated joints will then send their position back to the planning. Doing so allows one to have a fully working simulation. Two controllers must be set up, a *joint_state_controller* which publishes current simulation joints and *position_controller*, which sends joint commands taken from the planning directly to the simulation joints with the help of the <transmission> tag.

Two URDF scripts were written for the Gazebo simulation. The first script holds an open loop model while the second script a closed-loop one. The open loop model without the needle is shown in figure B.2 appendix. The model where the upper and lower loops both contain a full needle is shown in figure 5.5. Here all the joints are considered actuated. The script itself mainly contains URDF tags of parent and children links where one child does not have more than one parent. On the other hand, the closed-loop model uses the sdf tags to overcome the limitation that a child cannot have more than a single parent. The needle link is split into two pieces and connects in the middle with a fixed joint. The new fixed joint that creates the closed-loop is not a URDF tag joint but an sdf one. In this manner, we overcome the inability of URDF scripts of creating closed-loop models. The actual model can be seen in figure B.1 in the appendix C.

In the end, the chosen implementation was that of the open-loop since the decision to have the needle insertion done manually was made regardless of the gazebo model. Simultaneously, a big limitation in having a closed-loop model is the fixed size of the needle and the inability to have the needle slide for insertion. To simulate needle insertion, the points where the needle is connected to other links are considered prismatic joints. In the closed-loop, such an implementation creates too many problems, mainly because of the lack of rigidity when splitting the needle in two separate pieces. These reasons, together with how gazebo communicates with the planning scripts, are why the closed-loop implementation was not chosen. The closed-loop model could be useful in a future implementation in combination with a self-written plugin for control of the joints in closed form.

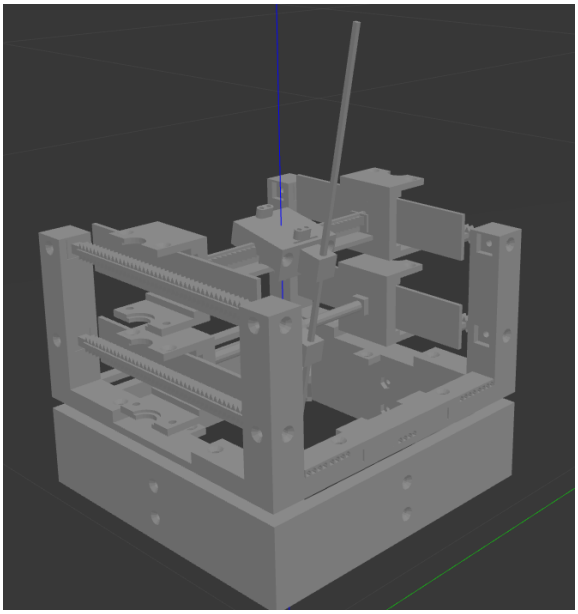


Figure 5.5: The open loop robot containing the needle in gazebo

5.3 MoveIt

MoveIt is a primary source of the functionality for joint manipulation in ROS. It builds on the ROS messaging and build systems and utilizes some of the common tools, ROS Visualizer (RViz) and the ROS robot format (URDF), to create a robot planning framework. The package takes parameters from the ROS server and uses them in combination with other external packages to create kinematic solutions. The general framework of the package is mainly based around the *move_group node*.

The MoveIt package has a central node called *move_group*. This node communicates with other topics to get the necessary joint information where topics are named buses over which nodes exchange messages. This communication provides the essential current joint position that is needed to plan and execute commands. By knowing the joints current value, the node can calculate how much it needs to move the joint for it to achieve its final position. The *joint_states* topic is published through the joint_state_publisher, which is aware of the information in the URDF configuration and combines it with information from external hardware values or simulation values. The Transformation (TF) package is utilized to figure out the transformations between the robot links. *Robot_state_publisher* uses the joint information received through both *joint_state_publisher* and URDF and publishes link transformations with the help of the TF package. In this way, one can get the Cartesian position of each link. These transformations

are then used to display the robot in RViz and calculate the forward and inverse kinematics of the robot. Better visualisation of this communication can be seen in figure 5.6.

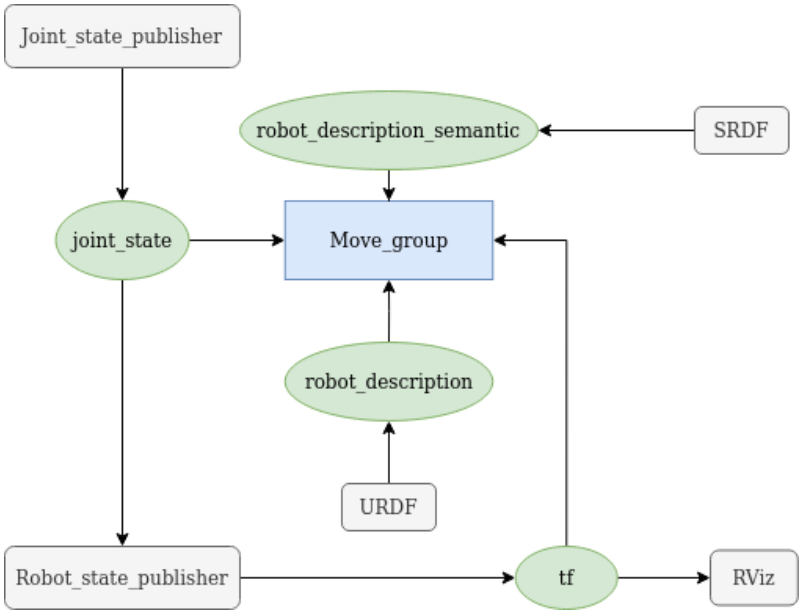


Figure 5.6: Diagram displaying the inputs of the *move_group*.

Different plugins can be used for the planning of inverse kinematics. The default planner is Kinematics and Dynamics Library (KDL), which uses the numerical Jacobian-based way to solve kinematics. Other external solvers can also be implemented through MoveIt if specifically needed. As mentioned in the kinematics chapter, the *Track_Ik* solver was chosen. Another important part of MoveIt and a major reason for it being the preferred choice for interface creation is the Collision plugin. This plugin is called during each planning session to check if the links collide with each other or the environment. The plugin facilitates the use of meshes as collision matrices, which is highly beneficial when the robot has a complicated build. After the planners have calculated the necessary joint values to achieve the requested Cartesian position, an extra package is needed to execute the commands, *ros_control*. Joint commands are sent through a ROS action interface and a joint trajectory controller. A server needs to service this action. For this to take place, a specific controller called *trajectory_controller* can be created independently from MoveIt by using the *ros_control* package. In our case, the trajectory controller package based on *ros_control* is created and is called *robot_test_control*.

It needs to be mentioned that to use the services provided by *move_group* node, three implementations are possible as seen in figure B.3 in appendix C. The first implementation is through C++, using the *move_group_interface* package. The second method, which is also one of the chosen implementations, is through Python by using the MoveIt_commander package. This python package is, in fact, just a wrapper of the original C++ package. The final type of implementation is through the GUI by using the Motion Planning plugin in RViz. The GUI implementation can also be beneficial from a medical perspective. This could give the doctor a better visual understanding of how the needle approaches the tumor. This is because this type of implementation allows for a visualization of the tumor in the robot's workspace by using RViz and an external transformation script. The python implementation was chosen in this project to keep the same scripting language throughout all the software framework.

5.4 Robot Interfaces

5.4.1 MoveIt planning interface

A python interface script was created to be able to use the functionalities of MoveIt. As mentioned in the MoveIt section, those functionalities can be reached with the help of the MoveIt_commander package. The script is written in python and called through ROS. It takes as input a vector with position and orientation, and it outputs joint positions. As mentioned in the previous chapter, the robot itself is separated into two kinematic chains. The top kinematic chain is called *move_group_1*. This chain contains the rotational *Joint_0*, top translational *Joint_1_1*, top rotational *Joint_2_1*, top middle prismatic *Joint_3 _1* and needle prismatic *Joint_4 _1*. The bottom kinematic chain is called *move_group _2*. This chain contains the bottom translational *Joint_1_2* joint, bottom rotational *Joint_2_2* joint, bottom middle prismatic *Joint_3_2* joint and needle prismatic *Joint_4_2* joint. *Track_Ik*.

A visualization of the script workflow can be seen in figure in the 5.7. Initially, an end-effector setpoint should be fed to the script. At the moment, the end-effector position is given within the script but it could also be sent through a ROS topic. For the final joint positions to be calculated and sent through ROS, the get_joint function is called. This function takes in the aforementioned end-effector setpoint and returns either true or false depending on whether a solution has been reached. Get_joint calls for the inverse kinematics through the external TracIk package, after a solution has been reached and collision checking is performed. Only after both of these calls return true are the joint values sent through a ROS command and the inverse kinematics is considered executed.

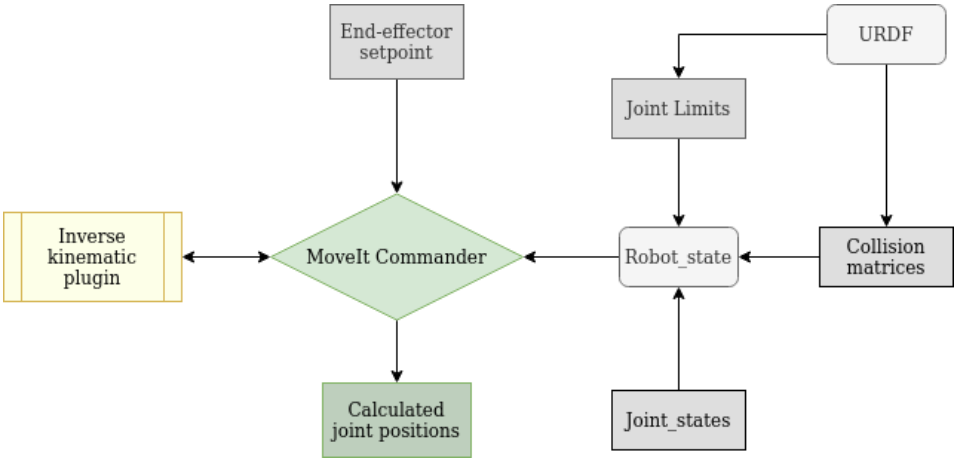


Figure 5.7: Diagram visualizing the workflow of the used MoveIt python interface with TrackIk package used for the inverse kinematics. Both the inverse kinematics and collision checking are called inside the MoveIt commander class. The output of those functionalities in the end are calculated joint position.

5.4.2 Python hardware interface

The python hardware interface is necessary for converting the joint information sent from the MoveIt interface to something that the motors and Arduino can understand. The communication between the two interfaces is done with the help of ROS messages. The joint information coming from the MoveIt interface is in meters and, it already takes into account the joint limits. Arduino, on the other hand, accepts steps as input. The step size differs depending on the motor, so the calculation is joint-specific. A diagram showing the input of the script is shown in figure 5.8.

Figure 5.8: Diagram showing the input and outputs of the Joint to Step python interface. This script simply does a conversion and uses ROS to send the appropriate message type containing the joint values in step unit to the arduino.

5.4.3 Arduino interface

The Arduino communicates with the python script via *ros_serial* package. The data it receives is already converted into a format that the motors can directly understand. To control the motors themselves, a package called pneumatic stepper is used. The package contains a motor class that is used to create a motor object. The functionalities that exist within this class are setting a setpoint, speed, update motor state and even specify the type of pneumatic motor used (number of valves). In the current implementation, the movement of the motors is relative to the initial positioning performed during calibration and keeps a static speed throughout all the implementation. The setpoint needed for the motors is received by subscribing to a ROS topic where a python script publishes only when a target has been specified. On the other hand, the Arduino script continuously publishes the joint's current position. The process of the motor update and the ROS subscription is done separately from each other in different threads.

6 Robot Feedback

6.1 Vision hardware

The camera used for this implementation is an Intel RealSense Depth Camera (D435i), United States. The camera can be seen in figure 6.1. The D435i is a stereo solution, offering both depth information and RGB feed for a variety of applications. It is a wide field camera with a high resolution. A better view of its internal components can be seen in figure 6.2. The camera has an on-chip calibration as well as its own Intel RealSense SDK 2.0 software package, which provides a variety of useful functionalities. The camera is placed on top of the robot allowing for a full top-down view of the robot. The RGB aspect of the camera is used for detection. The colors in combination with the different sizes make it easier to track a multitude of objects. It needs to be noted that in the current implementation, the chosen resolution is the 1280x720 pixel format with an fps of 30.

Figure 6.1: Realsense D435i camera (RealSense, 2020)

Besides the camera, the markers to be detected also needed to be chosen. Two types of objects must be detected by the software. One marker type was placed on the moving joints, while a reference type was placed on the frame itself. The reason behind this is to be able to do the calibration independently of how the robot is placed in front of the camera. Even if the robot has been placed in a rotated or displaced manner, the result of the calibration must still be the same. In total there are 4 joint markers; bottom rotation *Joint_0*, top side translation *Joint_1_1*, top middle translation *Joint_3_1*, bottom side joint *Joint_1_2* and three reference markers; bottom reference marker as *Ref_0*, top corner reference marker as *Ref_1*, middle reference marker as *Ref_2*. The markers placed on the robot can be seen in figure 6.3. One of the reference markers, *Ref_1*, will be used for two joints, top side translation *Joint_1_1* and *Joint_1_2*, to reduce the number of overall markers. Each marker has a small holder on the robot as to keep its position static in between calibrations. The markers themselves are relatively small but differ in color, size and placement. Red is the color of the *Joint_0*, *Joint_1_1*, *Joint_3_1* while yellow that of *Joint_1_2*. The reference markers are blue for the two top translation joints and green for the bottom rotation joint. All the markers are circles to achieve rotation in-variance, and matte

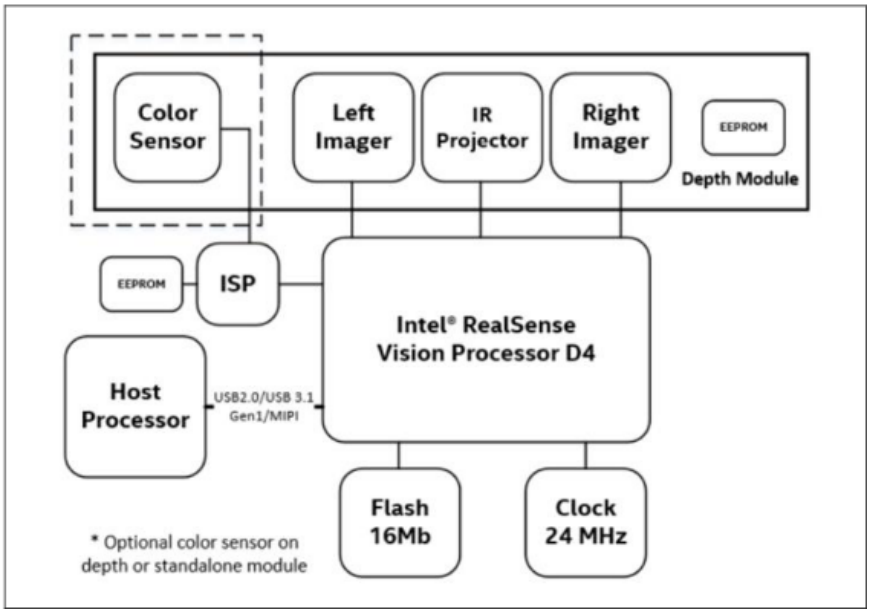


Figure 6.2: Block diagram of the D435i internal components. (RealSense, 2020)

so they don't change significantly under lighting conditions. The reason behind keeping the markers flat is to have a better depth result.

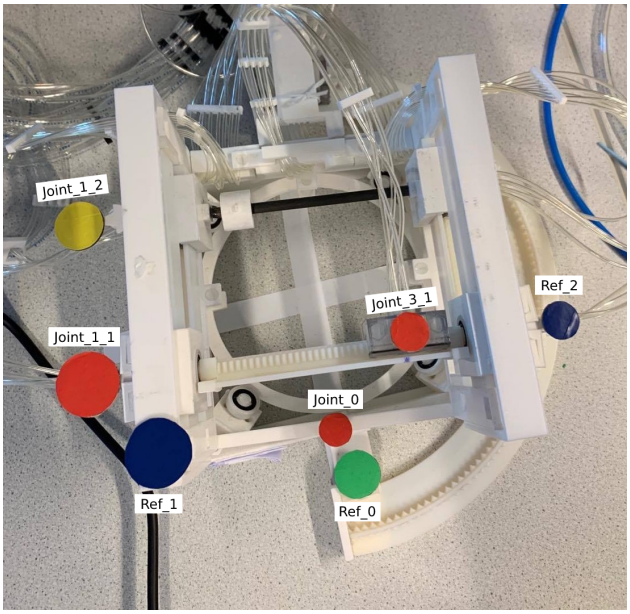


Figure 6.3: Joint markers and reference markers on the robot

6.2 Vision software

When developing the vision software, a combination of OpenCv and Intel RealSense SDK 2.0 was used. To send the resulting values from the detection, ROS messages were utilized. All the packages were implemented within one python script to increase the practicality of the software integration.

6.2.1 SDK and Depth implementation

The video feed and depth information were all retrieved using the RealSense SDK 2.0 software, created within the RealSense brand. In this project, the SDK software is utilized to retrieve the depth information. This is done because the software has direct information on the intrinsic and extrinsic parameters, which are needed for the pixel projection and de-projection. Pixel projection takes a point from a stream's 3D coordinate space and maps it to a 2D pixel location on that stream's images. De-projection takes a 2D pixel location on a stream of images, as well as a depth specified in meters, and maps it to a 3D point location within the stream's associated 3D coordinate space.

By using the depth information provided by the 3D camera, one can differentiate between objects that are of importance from those that are present in the background. The depth value varies depending on the camera's distance; thus, a threshold can be specified to classify background/foreground objects properly. This classification is helpful when dealing with busy backgrounds, and the usual features were chosen, such as colour or size are not sufficient for the differentiation. On the other hand, the distances measured are of a higher accuracy since the distance is calculated in 3D, and SDK is capable of taking in account camera distortions. Another benefit of using SDK is the possibility of retrieving the raw feeds, colour and depth. This is done with the reasoning that both the depth feed and the RGB feed come pre-synced. However, an alignment of the frames still needs to be done in order to get accurate depth reading. This is because the depth and RGB feeds have different maximum resolutions and viewing angles. The chosen resolution for both of the feeds was 1280x720 pixels with 30fps.

The first step in 3D distance measurement is retrieving the 2-dimensional X and Y values of the object's centroid from the RGB feed. Z depth of the same object's centroid is then stored from the aligned depth frame. The three coordinate values are then transformed to real-world 3D coordinates using the already known intrinsic parameters. Finally, a 3D euclidean distance is measured between the two objects of interest by using their respective world coordinates. The measurement pseudocode can be seen in 6.1.

```
distance_measurement ( trackers, references)
{
    #this value is taken from the SDK parameters
    cam_intrinsic;

    #sort according to size and depth
    #this is done to match tracker to reference
    trackers=sort(trackers)
    references=sort(references)

    #deprojection calculation
    for tracker in trackers :
        3D_pos= calculate3D(centroid,depth,cam_intrinsic)
        trackers_position.append(3D_pos)

    for references in trackers :
        3D_pos= calculate3D(centroid,depth,cam_intrinsic)
        trackers_position.append(3D_pos)

#distance calculation
for i=len(trackers)
3D_distance=euclidian(tracker,reference)
joint_value.append(3D_distance)
```

```
}
```

Listing 6.1: Measurement pseudo code

Now that the motivation behind the usage of SDK software has been stated and the technical implementation of the 3D measurement has been explained, the next section will focus on a different aspect of the framework. In the coming section the detection and tracking method will be clarified.

6.2.2 OpenCv Implementation

OpenCv is an open-source, fully documented library with tools and optimization methods for several tracking algorithms. The platform is used for pre-processing, detection and tracking. The workflow diagram of the software can be seen in more detail in figure 6.4. The detection and tracking are, in fact, done in the RGB feed. The first step in detection is transforming the feed into HSV colorspace. This is done because one of the main features of the tracked markers is colour. Because similar colours can be present in the background and the lighting in the environment may vary, morphological operations such as opening and closing are done to the scene. After the noise has been dealt with, the actual detection takes place. Contour detection is used within OpenCv for the markers. The detected contours are then filtered depending on size, shape, and depth. The pseudocode of the detection algorithm can be seen in 6.2. The resulting objects of interest are then used to initialize the tracker class. The tracker class is reinitialized each time the tracker fails, or after 200 frames have passed such that the accumulated error can be cleared. This class provides information such as the X, Y, Z, coordinates of the object's centroid in the cameras world frame. This position is then used to find the distance between the joint markers and the reference markers. This distance will be the information utilized for calibration and as well as feedback.

```
detection ( aligned rbg_feed, aligned depth_feed)
{
    #The xy coordinates of the rgb feed match those of the
    #depth feed
    aligned rbg_feed;
    aligned depth_feed;

    hsv_tracker= hsvfortracker(aligned rbg_feed)
    hsv_reference= hsvfortracker(aligned rbg_feed)

    hsv_tracker=open(hsv_tracker)
    hsv_tracker=close(hsv_tracker)
    hsv_reference=open(hsv_reference)
    hsv_reference=close(hsv_reference)

    contours_tracker=findCountour(hsv_tracker)
    contours_reference=findCountour(hsv_reference)

    for cnt in contours_tracker:
        area=moments(cnt)
        points=approxPolyDP(cnt)

        if threshold_1>contours>threshold_2 and points>8
            find centroid;
            find w,l
```

```
                find depth(centroid, depth_feed);

                if distance>depth>depth
                    track_det.append(nr,centroid,depth,w,l)

        for cnt in contours_reference:
            area=moments(cnt)
            points=approxPolyDP(cnt)

            if threshold_1>contours>threshold_2 and points>8
                find centroid;
                find w,l
                find depth(centroid, depth_feed);

                if distance>depth>0
                    ref_det.append(nr,centroid,depth,w,l)
    }
```

Listing 6.2: Detection pseudo code

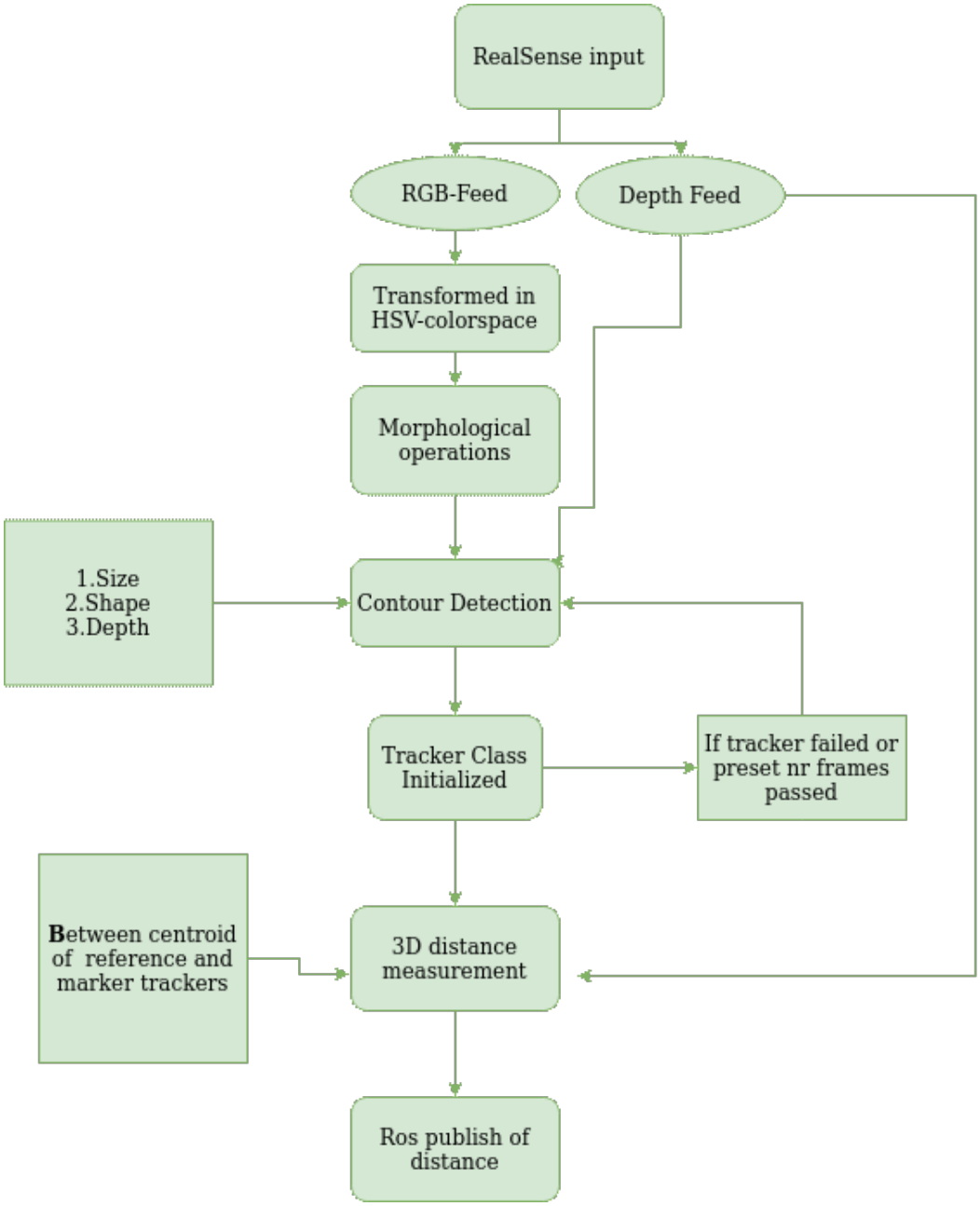


Figure 6.4: Diagram displaying the workflow of the OpenCv vision feedback software.

7 Testing Method

7.1 Workspace Test

The workspace test will give a good indication on the achievable positions of the needle in the robots workspace. A comparison between the simulation test versus the real workspace can pinpoint physical problems with the actual robot.

7.1.1 Simulation Test

To check the robot workspace, a simulation workspace test is done. A script is written to iterate through the joint combinations and use them to calculate the forward kinematics. The script calls for Moveit to execute the kinematics. Consequently, combinations where a collision is happening are not taken into account. The end-effector in this test is set at the tip of the needle and is saved in an external file in each iteration. To reduce simulation time, not every single joint combination was taken in account. Instead, joint limits were taken into account and testing was performed for joint positions in between these limits.

Since the bottom kinematic chain is solved separately, the way the test was performed was by solving for forwarding kinematics on the top chain, taking into account the joint combinations, and then using the top chain's end-effector position to solve inverse kinematics for the bottom chain. In this way, it is made sure that the joint positions can take us to an achievable needle position. The test was done for a needle of 20 cm and needle insertions for up to 4 cm from the initial position.

The final end-effector positions were recorded and saved to an external file with the help of a Python script and then used as input for a plotting script.

7.1.2 EM tracker Test

The real-life workspace test was performed with the help of a physical sensor as a ground truth. For this test, the medical Aurora 3D Guidance electromagnetic sensor manufactured by Northern Digital Inc was used. The sensor works by using a field generator that emits a low-intensity, varying electromagnetic field and establishes the tracking volume position. In the current setup, a tabletop field generator is used. The robot is then placed on top of the tabletop generator and lifted slightly with non-conductive platforms. The actual setup can be seen in figure 7.1. The sensor is a 6DOF Aurora EM tracker and is placed in the needle as seen in figure 7.2.

The reason for not placing the sensor at the tip of the needle is because the needle is inserted manually. With the current robot, it is not possible to keep the needle fixed to the robot while moving, nor is it possible to place the sensor at the tip of the needle. This sensing method is expected to introduce some inaccuracies in the measurements. The whole physical setup can be seen in the diagram in figure 7.3.

Besides the physical aspect of the EM tracker, the ability to record and save the data depends on its software setup as well. The overall idea of the software can be seen in figure 7.4. An NDI server is established with the help of Matlab and the Aurora software package. This server sends the EM tracker data in the form of a customizable message containing the x,y,z position in millimeters and rx, ry, rz, rw orientation information in the form of quaternions. The laptop client is then connected to this server with the help of an external ROS package called ROS-IGTL-Bridge. To be able to analyze the results, this incoming information is then saved in a ROSbag. Using a Python script and the ROS-IGTL-Bridge package, the data is saved onto a text file containing only positions and orientations.

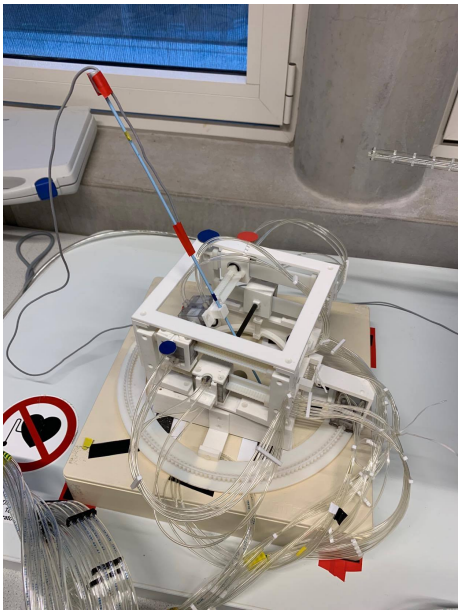


Figure 7.1: EM tracker physical setup picture

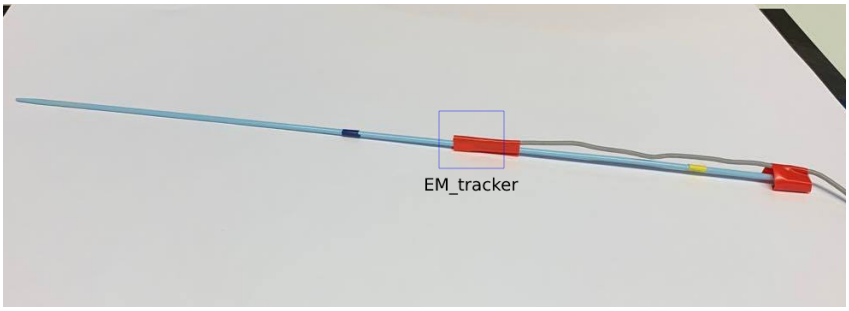


Figure 7.2: EM tracker sensor put on the needle

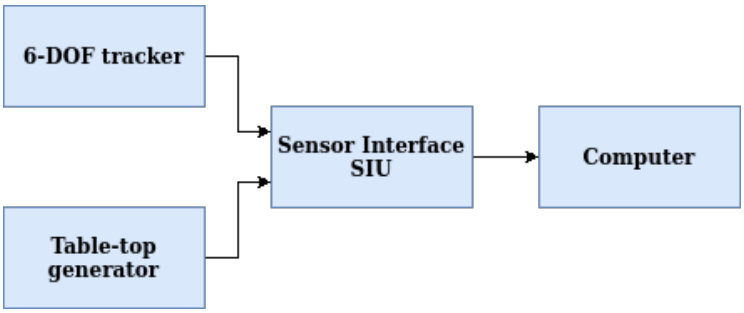


Figure 7.3: EM tracker physical setup components

After recording all the needle data and before performing analysis and comparisons, a transformation needs to be done from the robot in the EM tracker workspace to the robot workspace. The EM tracker sensor is placed in multiple known positions in the robot geometry, and these points are recorded with the help of ROS in the form of ROSbags. After the necessary points have been recorded in a similar manner, their position in the robot simulation is recorded as well. By knowing the positions in the EM tracker workspace and the robot workspace, a rigid transformation can be performed. In this example, seven points were used for the transfor-

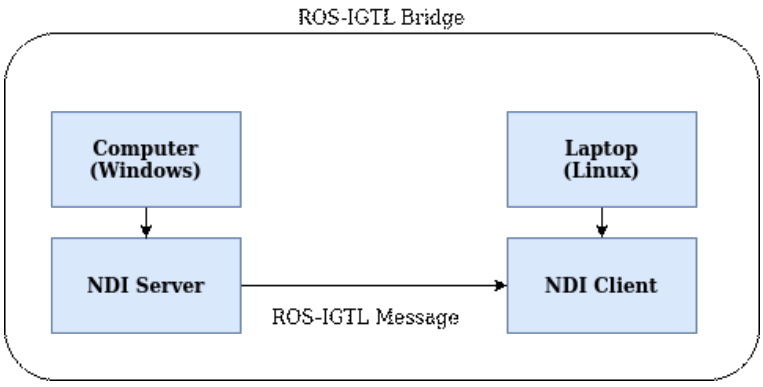


Figure 7.4: EM tracker software setup

mation. All of the points were located on the surface of the robot. The chosen method for this transformation was done by using a 3D affine transformation and solving using singular value decomposition (SVD). The pseudo-code of this transformation can be seen in 7.1. This method provides the optimal rotation and translation between corresponding 3D points. From this test, one can see how the needle position is in the real world workspace compared to the needle position in the simulation.

```
int transform( simulation_points, EM_tracker_points)
{
    #The points/point clouds from the two different
    #workspaces are used as input
        s_points=simulation_points;
        t_points=EM_tracker_points;

        #The centroid of the two "point clouds" are calculated
        s_centroid=mean(s_points)
        t_centroid=mean(t_points)

        #The origin of the two "point clouds" is calculated
        s_Origin=(s_points-s_centroid)
        t_Origin=(t_points-t_centroid)

        #The translation component is removed from the clouds
        H=s_origin x transpose(t_Origin)

        #The SVD method is used to find the optimal rotation
        U,S,V=svd(H)

        #The rotation matrix is calculated
        R= transpose(V) x transpose(T)

        #The translation component is calculated
        T=-R x s_centroid + t_centroid

        #rotation and translation are returned
        return R,T;
}
```

Listing 7.1: SVD method pseudo code

7.2 Vision tests

These tests will provide the answer to the core questions of this thesis, regarding whether visual feedback is reliable and what methods provide the best result. Two main tests will be performed in this chapter. The first test will be a comparison between the selected trackers, and the second test will be a motor calibration test. From the first test one can see if continuous feedback is accurate enough and what algorithms can be used, while the second test will showcase the results of the calibration of the motors. When doing the calibration, continuous feedback matters less than the final distance measurement for reaching the 0-joint position. It needs to be noted that both vision tests were performed on different days, with the base of the robot being placed at various initial rotations.

7.2.1 Tracker Comparison Test

The tracker comparison test is performed to help understand which trackers bring the best results when using vision for feedback. As mentioned in the vision paragraph, the general structure of the vision application uses a combination of detection and object tracking. In this test, two joints will be followed; the upper translation joint and the middle translation joint. Besides the actual moving joints, the reference markers will be followed as well. In total, four markers will be tracked, and two sets of multi-trackers will be initiated. The four trackers are the KCF, CSRT, MOSSE and MedianFlow. After all the markers have been tracked, a euclidean distance is calculated between the joint markers and the reference marker. For all the trackers, the distance is measured using the 3D x,y,z data. For each tracker, seven tests were done, and the robot itself was placed at different positions under the camera.

To facilitate a comparison between the ground truth and the test result, another script inside the Arduino containing the known step value is sent through ROS. The previous paragraph mentioned that the motors move relative to an initial position, and as long as the test environment is stable and the motor keeps a safe speed, the actual ground step position is recorded and sent through a ROS topic. Both the value of the distance from the tracker and the Arduino step value are sent to a ROS node, which translates the step value into a distance using the size step and saving it to an external file. The values are then imported to a Python file where the error is calculated and plotted.

7.2.2 Calibration Test

In the calibration test, one of the trackers is chosen and all the joints are tracked. During calibration, the joints are moved one by one to their respective zero position. The order of calibration goes as follows; bottom rotation joint calibration, upper side translation joint and middle joint calibration, bottom translation joint calibration.

First, detection of the rotation joint and the rotation reference marker is performed. If that is not achieved, a move message is sent to the Arduino through ROS. When the rotation joint marker and the reference marker are detected, the tracker takes over. While this is happening, the distance between the reference marker and the rotation joint marker is measured. When the required distance to reach the zero position is achieved, a stop message is sent by ROS to the motors. After the rotation calibration is performed, the calibration of the other joints is performed. The same procedure is repeated, the distance from the top translation joint and middle translation joint with their respective reference markers is calculated, and the motors are moved until the zero position distance has been reached. Lastly, the same logic is applied for the bottom joint. When all the joints have moved to their respective zero positions, the vision code moves to continuous feedback mode, and all the trackers are refreshed and initialised

simultaneously. To check if the calibration was performed correctly, the motors are moved extra steps closer to the zero position. The steps left for reaching the true zero position are then counted as the error.

7.3 Target tests

The final experiment is the target test, in which the whole setup will be tested. The robot will be calibrated with the help of visual feedback. Targets are then placed in the real world in the form of a target paper, as shown in figure 7.5. This is done to test the accuracy and repeatability of the full robot setup. On the bottom frame, a target paper is placed with positions relative to the robot already known. Those points are then sent to the framework for performing inverse kinematics and moving the robot to the necessary position. The needle is then inserted manually and the paper is poked. In total, there are fourteen targets per test. The test was repeated ten times. The error is then measured from center of the target to the position of the needle insertion.

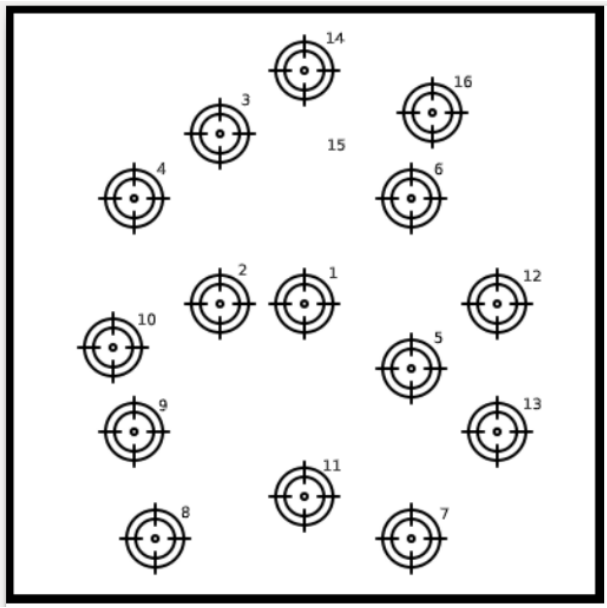


Figure 7.5: Target paper used during the target tests.

8 Results

8.1 Workspace Test

8.1.1 Simulation Test

The simulation test was done to see all the achievable points of the robots needle. During this test, the robot iterated through all the possible joint positions for the upper and bottom chain and saved the tip needle position in an external file. The figure with just the points but not the surface approximation can be seen in figure 8.1. Then a convex hull surface approximation was made to get a better visualization of the overall surface. The result of the surface can be seen in figure 8.2.

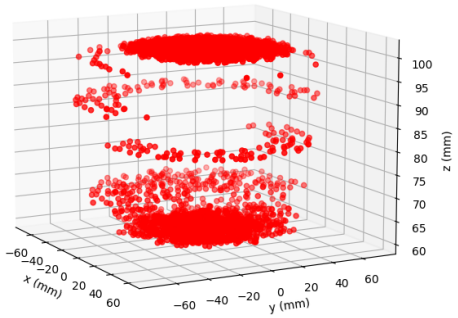


Figure 8.1: Simulation workspace of the needle in points.

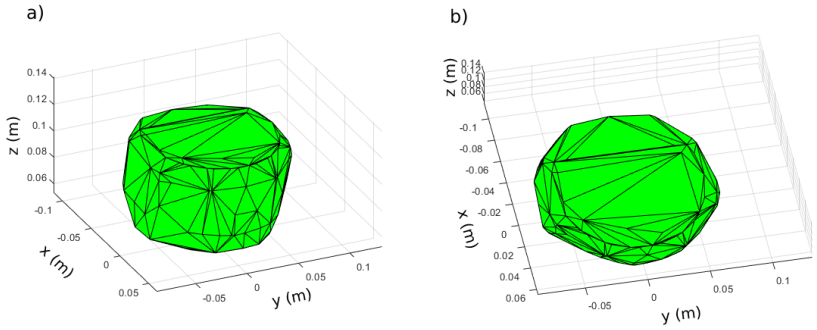


Figure 8.2: Simulation workspace of the needle by using surface approximation.

8.1.2 EM tracker Test

Before the EM tracker comparison tests are performed, the current motor ranges are checked. This allows for a better understanding on where the errors might come from. In figure 8.3, the robots positions in the EM tracker space can be seen. In this figure, the motor range is also visible in the green, red, blue and yellow plotted lines. Both the color blue and yellow denote a right side motor. The colors were chosen such that one can differentiate between the two tests. The tests were performed with the EM tracker placed in exactly the same position each time. All three motors, two side and one middle motor, were tested for their range.

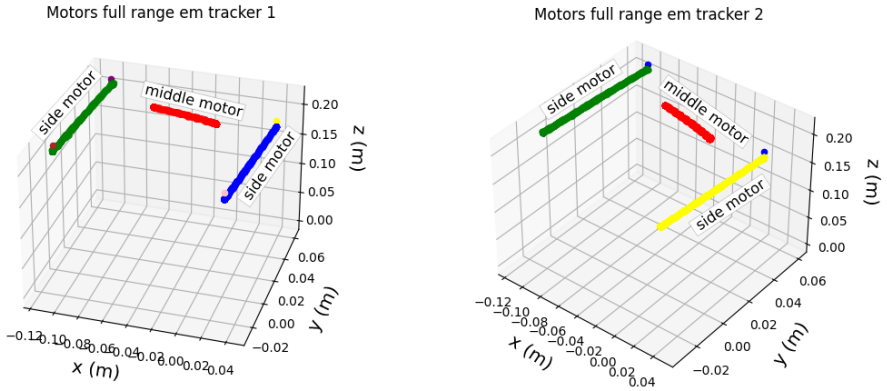


Figure 8.3: Motors range from the EM tracker measurEMent test 1 on the left test 2 on the right

The expected range values for the motors from the simulation are 70mm for the side motors and 62.5 mm for the middle motor. The ranges were calculated from the EM tracker test, seen on the left side in figure 8.3. From the calculations, it is seen that the motor range in the side motors is, in fact, achieved with the left side motor being 69.49 mm and the right side motor being 70.58 mm. From here, a small discrepancy of 1 mm can be seen between the two motors. This discrepancy could bring slight errors for the needle in the future. These two motors are coupled and move the needle in the y position. If there is a slight change between the two motors, it could cause a y-position error in the needle. The middle motor does not reach its full range, instead reaching only 47.2 mm. In the middle motor this is a significant error of 15.3 mm. This joint moves the needle in the x-direction meaning such an error on the motor can cause a needle error in the x-direction. The ranges of the second EM tracker test can be seen to the right in figure 8.3. The calculations showed that the achieved range of side motors was 70.58 and 68.73 mm. Once again, a 1.85 mm discrepancy between the side motors that could bring errors in the future in regards to the needle position. The middle motor once again does not reach its full range with only 49.17 with a significant error of 13.3 mm. These range errors could manifest in even bigger errors in future tests. The main reason for this error could be inaccurate calibration.

In the next test, a comparison was performed between the EM tracker needle position and the simulation needle position. The first step, as explained in the previous chapter, is to transform the EM tracker workspace to the simulation workspace. For this transformation, certain points were chosen on the real robot with the help of the EM tracker, and their respective points were then recorded in the simulation. Those points were then used to transform the EM tracker workspace to the robot simulation workspace by using a homogeneous transformation. In total, seven points were recorded in both of the tests. The chosen points can be seen as black dots in figure 8.4.

After the EM tracker points have been transformed, they are plotted next to the corresponding simulation points. The visualisation of these transformations can be seen in figures 8.6 and 8.7. Figure 8.5 shows how the robot considers the transformed positions.

Table 8.1 shows the error for the two tests of transformed points. Here the x, y and z distance error between the simulation needle position versus the simulation needle position was measured. The needle itself was put at different joint positions displayed on the left column of the table. The difference between the position of the simulation needle and the physical needle denotes the error in x,y and z. As can be seen, the leftmost column of the table denotes the joint positions of the robot. The first value denotes the first joint, which is the rotation joint with a

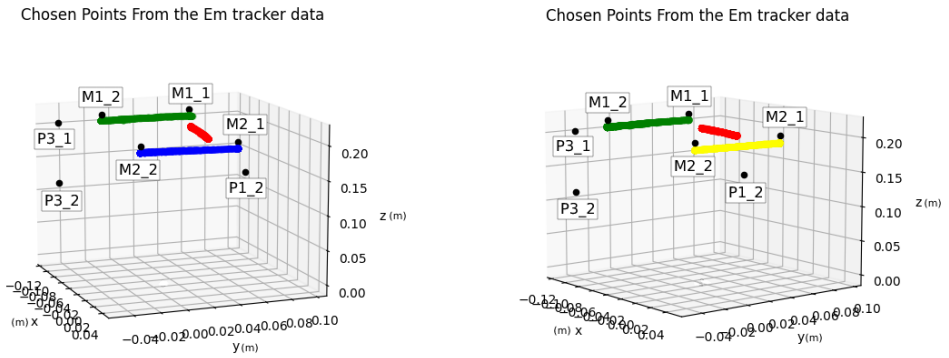


Figure 8.4: Chosen points used in the real robot used for the homogeneous affine transformation to transform to the simulation workspace. Points chose from test 1 on the left and points chosen from test 2 on the right.

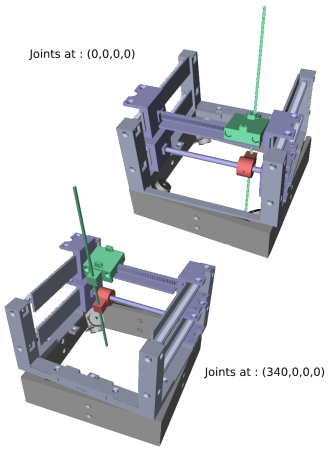


Figure 8.5: The simulation robot with joints at 0,0,0,0 and the simulation robot with joints at 340,0,0,0.

range value of 0 to 340 steps, the second value denotes the top middle translation joint with a range value of 0 to 90 steps, the third value denotes the bottom side translation joint which has a range value of 0 to 112 steps, and the final value is the top side translation joint with a range value of 0 to 112 steps as well. The second column shows the averaged error in the x-direction of both EM tracker tests, the third column the averaged error in the y-direction, the fourth column the averaged error in the z -direction, and the final column the total 3-dimensional error between the simulation needle and the EM tracker average.

The results were corrected for the z position since all results have a similar z-direction error. After normalizing for the z-distance error, it can still be seen that there is a significant error, especially at joint positions 340,0,112,112. This shows that there are discrepancies between the robot in real life and the one in the simulation. These discrepancies are expected to show up in the target tests as well. Initially, such discrepancies emerge because the motors have not reached the full range, mainly the middle motor. The other issues could be miscalculations in the rotation of the needle itself, meaning the needle does not reach the 0-rotation we expect it to have. However, errors in the transformation should also be taken into account. These transformation errors can come from the chosen points. The values of the chosen points in the EM tracker workspace might not relate 100 percent with their respective values in the simulation.

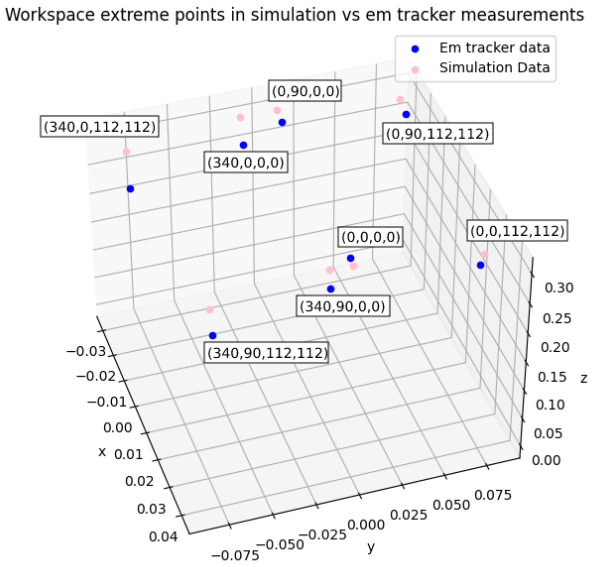


Figure 8.6: Transformed points vs the simulation points test 1.The unit of the x, y and z values is in m.

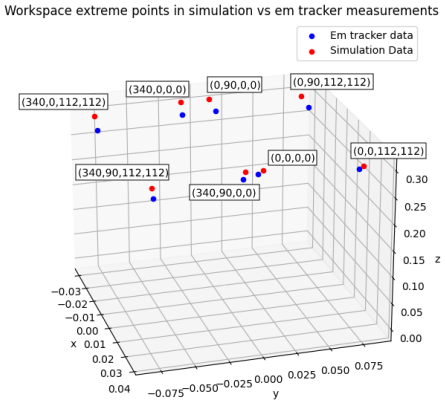


Figure 8.7: Transformed points vs the simulation points test 2. The unit of the x, y and z values is in m.

Error table in mm				
Joint positions	X-dst (mm)	Y-dst (mm)	Z-dst (mm)	Total-dst(mm)
0,0,0,0	5.03	5.23	1.42	7.42
0,90,0,0	2.32	6.26	0.43	6.74
0,0,112,112	2.38	2.51	2.63	4.66
0,90,112,112	2.88	5.94	0.79	6.65
340,0,112,112	8.10	6.32	0.85	10.34
340,90,112,112	2.99	4.51	0.60	5.62
340,90,0,0	2.14	4.63	1.47	5.36
340,0,0,0	6.05	6.44	1.83	9.05

Table 8.1: Distance between the tracker transformed and simulation needle

One final reason behind this error could also be the placement of the EM tracker. In the future, a proper holder should be designed for the placement of the sensor.

8.2 Vision tests

Two joints were focused on to make a comparison of the trackers. For each of the joints, four trackers were tested seven times. For this purpose, a ground truth value of the joint was saved. The error was then calculated between the ground truth joint distance and the tracker value joint distance. The averaged error for all the joint positions is then plotted to check and compare the four trackers. The standard deviation value is also plotted to see the repeatability of these trackers.

8.2.1 Tracker Comparison Test

First, the results for the side joint will be reviewed. Initially, the accuracy plot in figure 8.8 will be commented on. The trackers, represented as a line, were plotted for all the joint positions in order to make a proper visual comparison. All tests performed on CSRT and KCF trackers were successful. No tests needed to be repeated as the tracker did not entirely lose track of the object of interest. On the other hand, for Median flow and MOSSE, some of the tests needed to be repeated as seen in tables 8.4 and 8.5. Two tests failed and needed to be repeated when testing MOSSE, and four failed and needed to be repeated with MedianFlow. The tests were repeated such that it would be possible to have seven tests from all trackers. From the plot, it is clear that there is not a significant difference between the trackers. The only one that shows notably less accuracy is the Medianflow tracker. The other trackers on, the other hand, mostly overlap with each other throughout all the testing positions. In fact, such a high overlap between the trackers was not expected, higher accuracy was expected from KCF and CSRT.

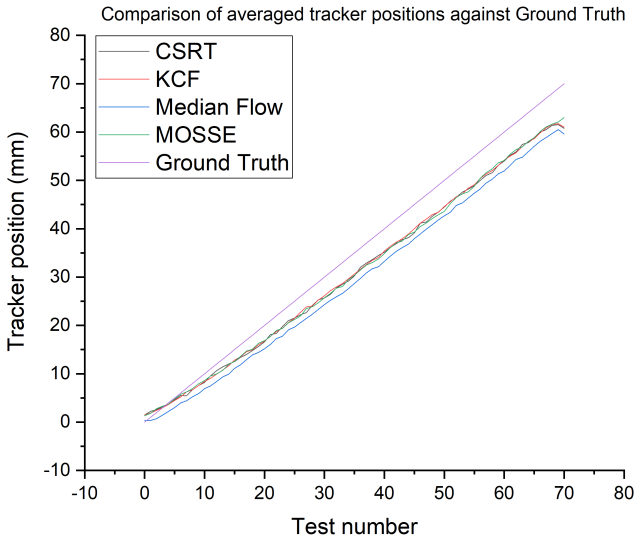


Figure 8.8: Comparison of average position values of the four trackers with the ground truth for the top side motor.

Another crucial aspect to study is the repeatability of these trackers, which can be seen in figure 8.9. Here, however, one can see that in terms of repeatability, MOSSE does significantly better while Medianflow and KCF follow behind with CSRT performing the worst. This could be a result of the speed MOSSE provides and its ability to keep up with the real time feedback of the ground truth.

Next, the results for the middle joint will be reviewed. The accuracy plot in figure 8.10 will be commented on first. The trackers represented once again as a line, were plotted here as well for all the joint positions. However, the difference in terms of accuracy between the four trackers

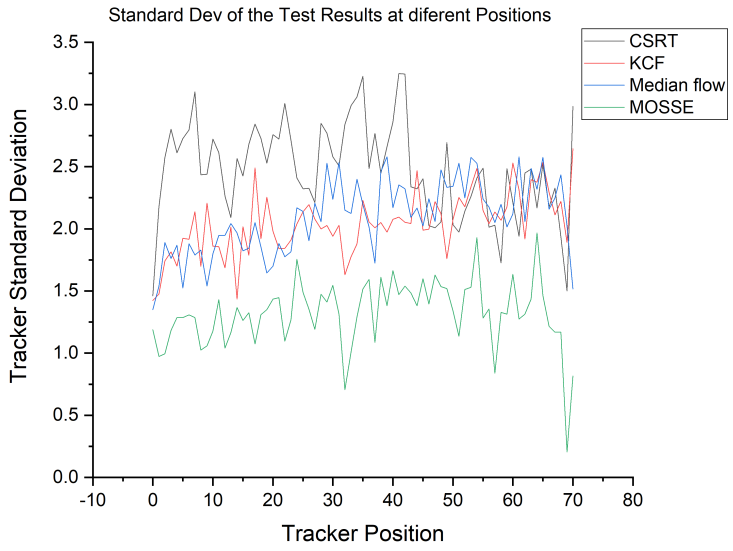


Figure 8.9: Comparison of the standard deviation for position values of the four trackers for the top side motor from all the tests.

is notably smaller. In spite of this, one can see that for the middle motor, the tracker generally performs better than for the side motor, with the error getting comparable only after 40 mm of joint movement.

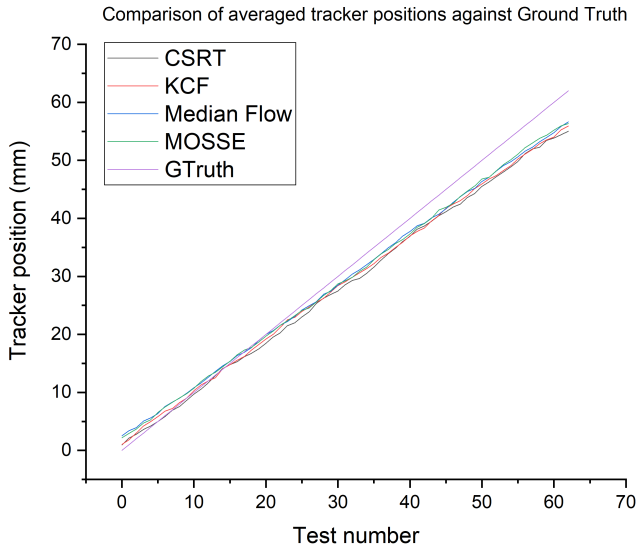


Figure 8.10: Comparison of average position values of the trackers with the ground truth for the middle motor from all the different tests.

In terms of repeatability, shown in figure 8.11, the results follow those of the previous joint. It can be seen that the CSRT tracker performs significantly worse in terms of repeatability, while MOSSE and Medianflow perform much better. KCF falls right in the middle in terms of repeatability performance.

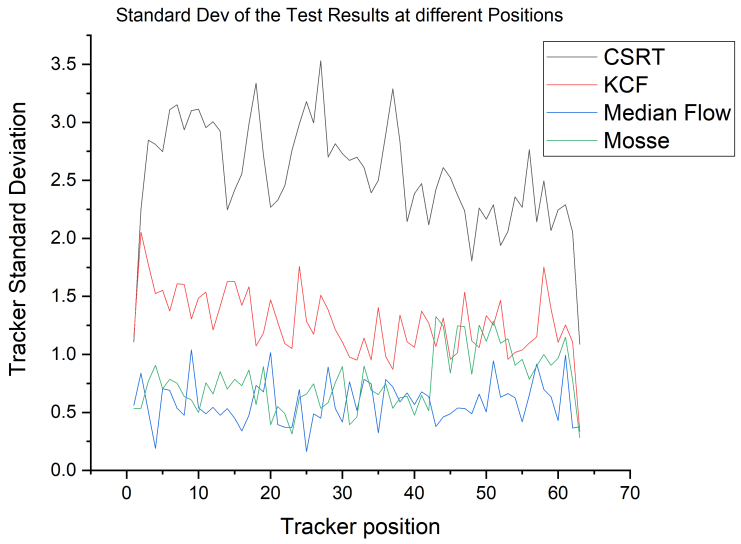


Figure 8.11: Comparison of the standard deviation for position values of the joint trackers for the middle motor.

To study the difference of the results, the actual repeatability error was noted in table 8.2 and the average error was noted in table 8.3. A more thorough visualisation of the results can also be seen in the appendix A. From the tables, one can see in more detail that MOSSE does, in fact, perform best in terms of repeatability and that Medianflow and KCF show a more significant difference in terms of repeatability when it comes to the middle motor. It can also be seen that the performance is, on average better for the middle motor than the side motor. On the other hand, in terms of accuracy, MOSSE once again performs well together with CSRT and KCF. Even in this closer look, the difference does not seem to be very significant between the trackers. Overall, in the performed tests MOSSE provides generally better results, with KCF following right after when considering both accuracy and repeatability. The reason behind such a result might be the speed required to keep up with the real time movement of the joints. A tracker like CSRT considered to be more accurate in literature suffers when dealing with faster moving objects. However, it must be kept in mind that MOSSE needed to be restarted twice to perform the test correctly. At the same time based in literature MOSSE is not known to be a highly accurate tracker with its performance mostly based on pure speed. When MOSSE was compared against trackers like KCF, it was never the preferred choice. In terms of accuracy KCF, does not fall behind significantly compared to MOSSE, having an accuracy difference of 0.7-0.1 mm. With a basis on literature as well as the fact that it does not fall significantly behind in terms of accuracy, KCF will be chosen for the calibration stage. The reason behind not considering MedianFlow in this discussion is the significant number of failed tests as seen in .

Repeatability Error		
Trackers	Middle motor	Side motor
CSRT	2.011	2.256
KCF	1.169	1.964
Medianflow	0.546	1.868
MOSSE	0.742	1.278

Table 8.2: Table denoting the reapeatability of MOSSE , MedianFlow, KCF and CSRT.

Accuracy Error in mm		
Trackers	Middle motor	Side motor
CSRT	3.574	3.772
KCF	3.181	4.195
Medianflow	2.754	5.965
MOSSE	2.476	4.093

Table 8.3: Table denoting the Accuracy of of MOSSE , MedianFlow, KCF and CSRT.

Testing Performance				
Test number	CSRT	KCF	MOSSE	MedianFlow
Test 1	Success	Success	Success	Success
Test 2	Success	Success	Success	Success
Test 3	Success	Success	Success	Success
Test 4	Success	Success	Success	Repeated
Test 5	Success	Success	Repeated	Repeated
Test 6	Success	Success	Success	Success
Test 7	Success	Success	Success	Success

Table 8.4: Table denoting the test performance of MOSSE , MedianFlow, KCF and CSRT for the middle joint.

Testing Performance				
Test number	CSRT	KCF	MOSSE	MedianFlow
Test 1	Success	Success	Success	Success
Test 2	Success	Success	Success	Success
Test 3	Success	Success	Repeated	Repeated
Test 4	Success	Success	Success	Success
Test 5	Success	Success	Success	Success
Test 6	Success	Success	Success	Repeated
Test 7	Success	Success	Success	Success

Table 8.5: Table denoting the test performance of MOSSE , MedianFlow, KCF and CSRT for the side joint.

8.2.2 Calibration Test

After comparing the four trackers, only one was chosen for the calibration. The decision was made based on both literature and the test results. Despite MOSSE performing much better overall, its shortcomings were mentioned in the literature. This tracker would definitely be a final choice if the motors would be moving at a faster speed and used for continuous feedback. However, the more reliable tracker in literature and overall testing while still displaying comparable results was chosen for calibration (KCF). The results of this test can be seen in figure 8.12. From the plot, it can be seen that the calibration error ranges from 1 to 1.5 mm and the std value stays under 1. The closer the value is to 0, the less variance the test data has from calibration to calibration, meaning the better it performs in terms of repeatability. From this plot one can clearly see that in the current state, the calibration through vision is not fully reliable. These errors will, in the future, add up to the physical errors of the robot and give suboptimal results.

8.3 Target tests

The final test performed was that of a target test, as explained in the previous chapter. Both the standard deviation and mean of the error between the reached position versus the aimed

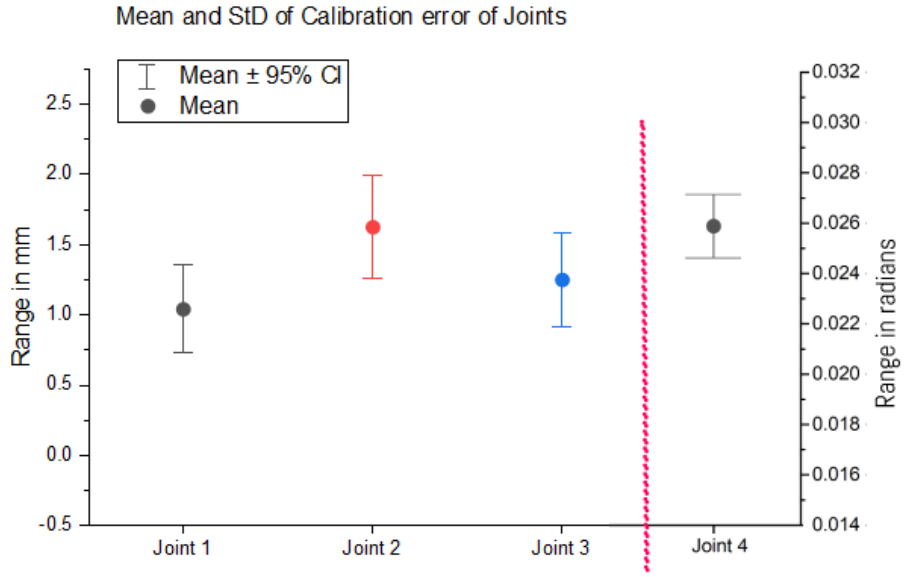


Figure 8.12: Standard deviation and mean of the calibration error of the translation joints and of the rotation joint

target was plotted in the figure 8.13. From the plot, one can see that the average error per target averages between 1 to 3 mm.

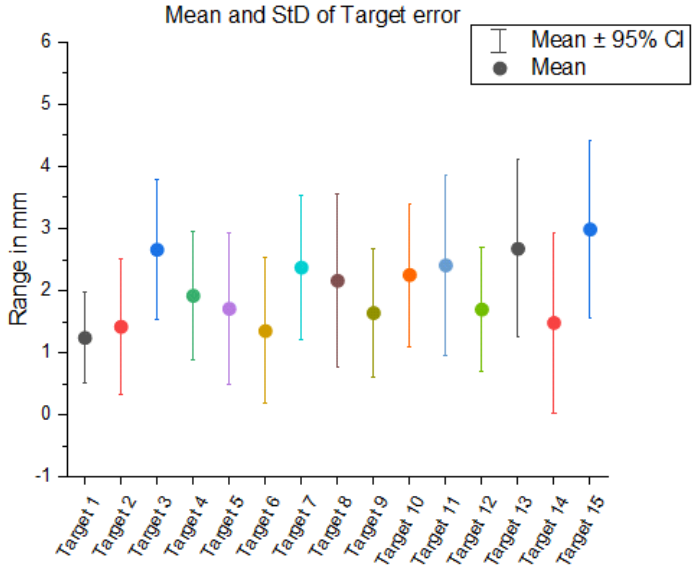


Figure 8.13: Standard deviation and mean of the target error

9 Conclusion

9.1 Discussion

In this section, the results in chapter 7 will be discussed. The workspace test was studied first. From the plots, it can be seen that the reachable space was 13.7 cm and 13.6 cm in x and y, respectively, for the first depth range. The reachable space was 14.2 cm and 15 cm in x and y, respectively, for the second depth range. The second depth range is 4 cm lower. It can be seen from this test that the deeper the needle is inserted, the bigger the reachable workspace of the needle. By comparing these results with the usual pancreas size, it can be inferred that in order to reach the tumor cells, the robot could be placed in the abdomen as designed. The average pancreas length is around 15 cm, with each part of the pancreas shown in figure 9.1. This implies that most of the tumor cells would be within the reachable space. It must be considered that the robot height over the body is 10.8 cm, the needle length is 20 cm, and in the current design, the needle can be inserted only 7 cm in depth. The needle would be able to reach more workspace with a longer needle.

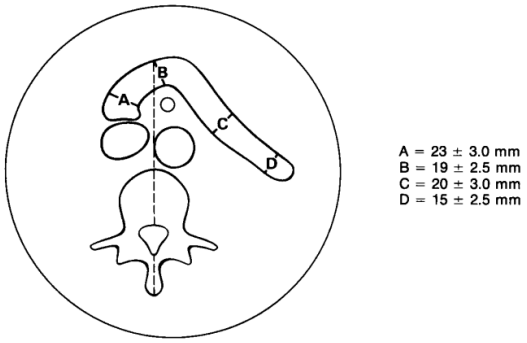


Figure 9.1: The size of the pancreas and its chambers.

The next thing to be discussed is the results of the tracker comparison and the calibration procedure. From the tracker comparisons plots and the tables in the results chapter, it could be seen that without considering the test performance failures, two repeated tests, MOSSE seemed to outperform all the other trackers. In the end, a decision had to be made between overall accuracy, reliability and overall performance. The KCF tracker was chosen instead due to its high reliability. The tracker came close second when considering both accuracy and repeatability. The literature review also indicated that KCF is the preferred tracker choice in many other applications. However, the accuracy error of the KCF tracker when continuously tracking the joint ranges from 6 to 2 mm. On the other hand, the tracker seemed to perform much better when only the final position of the joint was checked, which was the focus of the calibration test. The maximum error during this procedure was within 1-2 mm. The cause of this tracking error must be discussed further as well. Throughout the tests, all the trackers had difficulties following the movement of the joints and lagged slightly behind the moving object. When dealing with multiple trackers and a live camera feed, such lag is expected. Besides the lag, the trackers also had a hard time centring the object. Transforming the marker centroids into real-life coordinates comes with its errors, specifically less than two percent of the distance, which is a big reason behind the error in the results. A combination of all the above-mentioned reasons is the hypothesized cause behind the significant error seen in these vision tests.

Lastly, by studying the target test, one can see the error range of the final needle procedure. The error ranges from 1 to 5 mm. The target error is an accumulation of calibration errors,

combined with the physical discrepancies between the simulation robot and the real life-robot registered in the workspace tests.

9.2 Conclusion

To test the reliability and accuracy of visual feedback through a 3D camera, literature was looked at, and tests were performed. The results of both the calibration and the tracker comparison tests were discussed in the previous sections. By looking at the results, a conclusion can be drawn on whether the usage of vision as feedback is a realistic option when dealing with a needle insertion robot. When considering which tracker implementation could provide the best results, it was found out that MOSSE outperformed the other trackers slightly, with an error range of 1-5 mm. Nevertheless, this tracker failed to succeed in all the tests, and thus KCF was opted for instead. From the tests where KCF was used, it could be seen that the error ranged from 2-6 mm for tracking and 1-2 mm for calibration. For the current robot, an error of 1 mm in one of the joints can cause an error of 1 mm in the final needle position. Taking into account that these position discrepancies might come from multiple joints, the magnitude of the error accumulation needs to be considered. In a medical procedure such as IRE, which requires significantly high accuracy, a change of 2-6 mm is notably high, especially when taking into account the accumulation.

In the end, the RGB-D camera was opted for when measuring the actual distance instead of approximating through 2D information. The reason behind this was the accuracy and on-chip calibration provided by the RGB-D camera as well as its accompanying software SDK, which simplified the alignment and stitching complexity that comes from stereo cameras. The 3D depth information made it much easier to subtract the background during the detection phase of the vision implementation. In the future, the 3D camera could be useful for introducing targets under the robot through a point-cloud and transforming it in the robot workspace using the same transformation principles used during the EM tracker testing.

Finally, the target test showed that the average error per target ranged between 1 to 3 mm with instances where the error even reached up to 4 mm. Such high errors were already expected because of the results workspace test as well as the vision tests. The physical discrepancies between the simulation robot and the real-life robot are a result of coupling between the top side and the bottom side motors. The accumulated error is definitely a combination of calibration errors together with the aforementioned physical discrepancies. In the end, it can be seen that for the final prototype, the usage of a physical sensor needs to be explored. Such a sensor could provide superior results and improve on both the calibration errors as well as give a more accurate way of validating the physical robot design. Another aspect that has introduced errors is the holder of the needle. The ill-fitting needle introduces slight rotation errors that are hard to measure with the manual insertion. This could be fixed by introducing a holder that properly grasps the needle.

From the current interface, it was seen that using ROS did, in fact, make it easier to make changes and update the robot prototype. When small changes needed to be made in the robot, such as limits or 3D collision matrices, the current implementation required one only to change a value or mesh location in the URDF. The scripting also makes it easier to add new joints and links, as well as background objects to the robot and its workspace. ROS also lowered the difficulty in integrating different languages, software, hardware and simulation programs such as Gazebo. Implementing the 3D camera with the Arduino and the rest of the software was made possible and significantly easier by using specific ROS messages. Most of the hardware used did, in fact, already offer libraries for ROS integration available in multiple languages (Python, C++). The possibility of keeping the framework in one language (Python) made the learning and programming flow more structured. Lastly, the ROS implementation of the kinematics through MoveIt made it possible to do collision checking by using the actual 3D format of the

robot, which increased the accuracy of the overall implementation. All the aforementioned points are proof that the modularity that ROS offers is quite suitable in the prototyping phase. The majority of the implementation can still be kept, and the rest can be slightly modified in future implementations. Thus, the goal of having a working first prototype and implementation has been reached.

Overall, a proper framework was built, and the robot prototype is functional, although still in need of improvement. The modular framework makes it easier to switch to different feedback or design in future prototypes. The existing vision implementation could still be a feasible option for testing new designs and prototypes without fully implementing new sensors.

9.3 Future Work

To improve the overall implementation performance, the existing robot design can be improved in a number of ways. First, the robot's passive lower translation joint must be converted to an actuated joint. Notable errors were introduced during the insertion of the needle because of the passivity of the joint. The introduction of a motor would allow for a more stable movement and lower the needle error in the y axis in both position and orientation. Next, the needle holder also needs to be updated so that the needle is clasped firmly. This update would remove the residual orientation introduced to the final position caused by the holder. Lastly, a decoupling of the motors for the upper and lower translation joints should be taken into consideration. In the current implementation, it is not possible to correct for discrepancies between the two coupled motors.

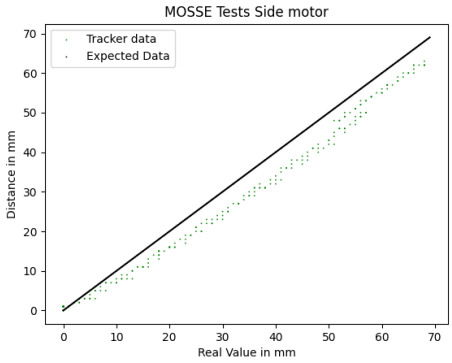
In terms of vision feedback, the current implementation could benefit from the addition of a lighting source. Constant pre-processing can be quite expensive in a real-life tracking scenario, where speed is of high importance. Another approach that can be taken to improve on the errors introduced by the environment illumination is using a combination of Infrared (IR) Markers and the current RGB implementation or a combination of Infrared (IR) Markers and Augmented Reality (AR) markers. The Realsense camera already uses two IR modules, making such an implementation feasible even with the current setup. A comparison test between the usage of visual feedback and different Augmented (AR) markers can also be performed to provide the most accurate results. The IR markers could make it easier to center and track the moving joints since they are not as effected by changes in the illumination. A recent similar study was conducted by Ehambram et al. (2019) which combined Aruco and IR markers and detected them using a RealSense D435 camera. The results of both markers were integrated through sensor fusion for more accurate results. The paper concluded that the usage of the IR markers improved the results remarkably.

The tracking of the markers can also be improved. As discussed in the previous chapters, KCF is an accurate tracker but suffers with fast-moving objects. On the other hand MOSSE better results but failed in two tests making it a less reliable option. A combinat ion of the two trackers could be implemented to achieve more satisfying results. Additionally, a multi-tracker implementation can be quite expensive for real-life implementation. To improve the speed of the tracker, code optimisation could be performed using multithreading or more efficient data structures.

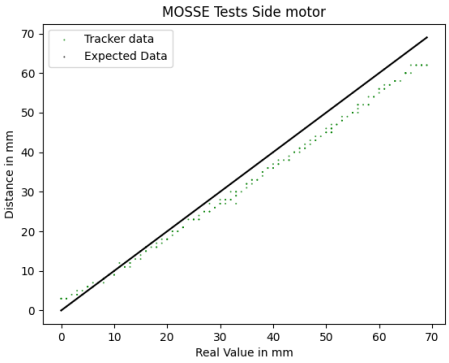
In the final prototype, the sensor should be changed to an MRI compatible sensor. The visual feedback is still not usable inside the MRI and can mainly be used for prototyping and calibration. This sensing alternative can be explored as it would also introduce the ability to drive the motors at a higher speed. However, the produced sensor needs to be researched in detail such that it fulfills the requirements of not being bulky and not disrupting the robot's design.

A Tracker Results

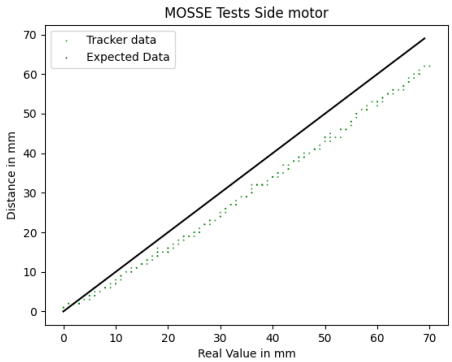
The plots in these chapter display the results of all seven tests for each of the tested markers. This was done to have a better understanding of the results data. All the trackers are plotted against the ground truth data, denoted with a continuous black line. The recorded tracker data is discrete thus is plotted as a scatter plot as seen in the figures.



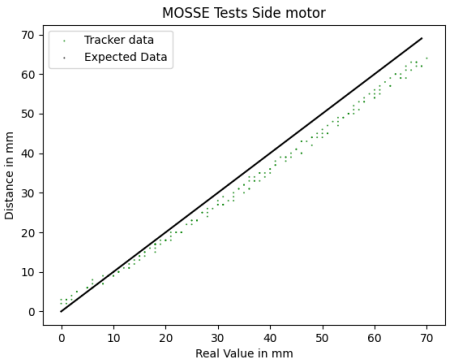
(a) Test1 side motor



(b) Test2 side motor

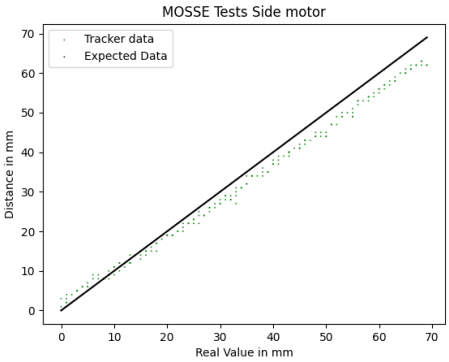


(c) Test3 side motor

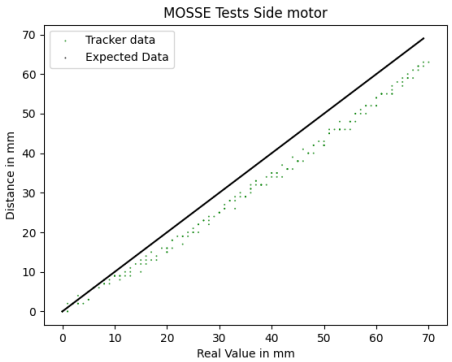


(d) Test4 side motor

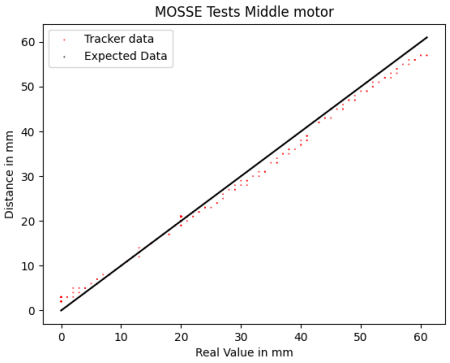
(e) Test5 side motor



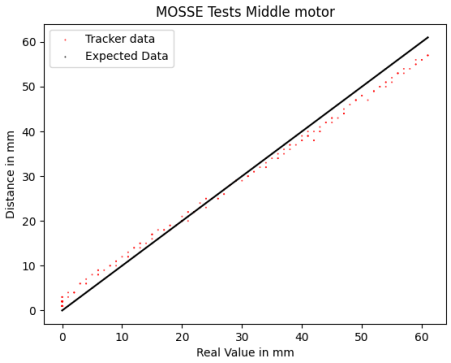
(f) Test6 side motor



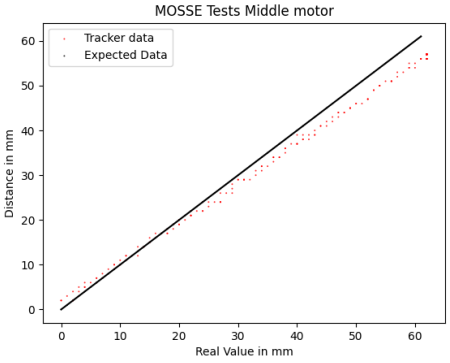
(g) Test7 side motor



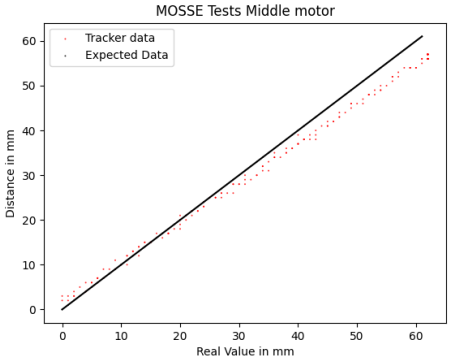
(a) Test1 middle motor



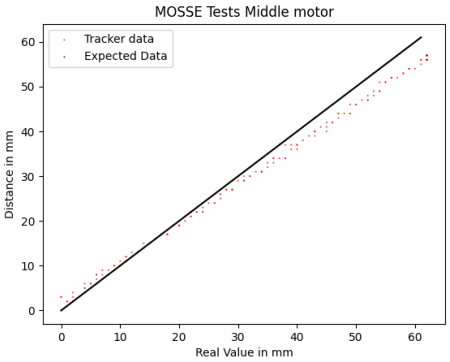
(b) Test2 middle motor



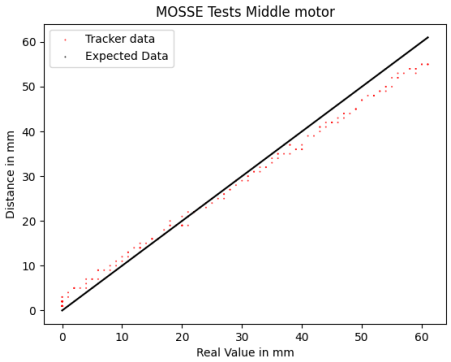
(c) Test3 middle motor



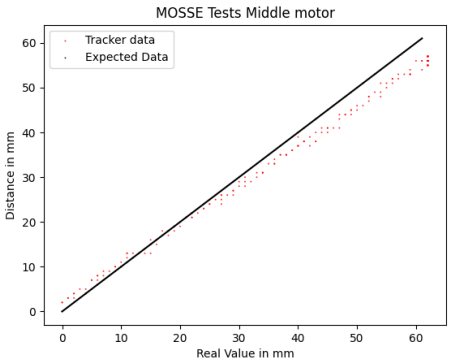
(d) Test4 middle motor



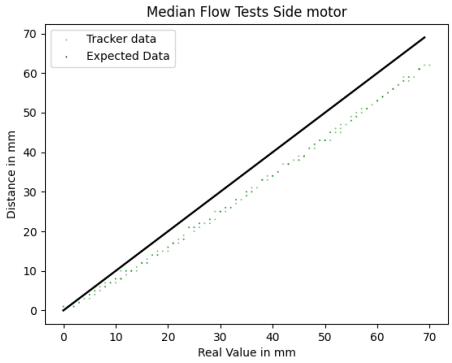
(e) Test5 middle motor



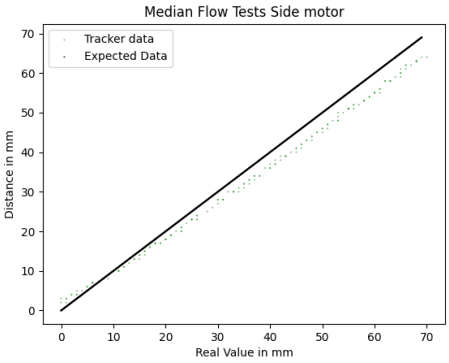
(f) Test6 middle motor



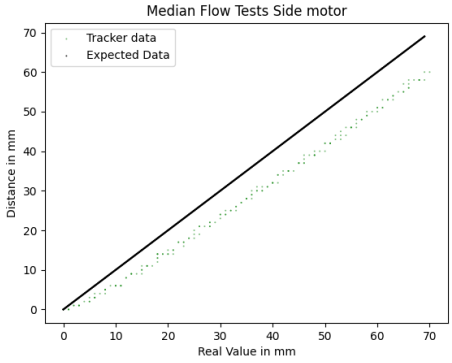
(g) Test7 middle motor



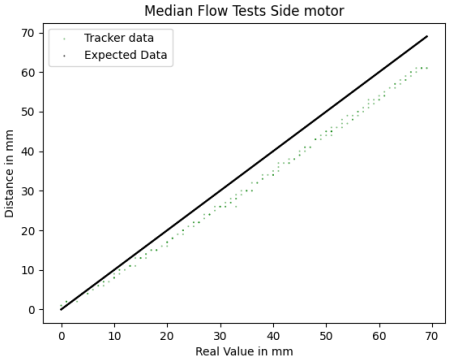
(a) Test1 side motor



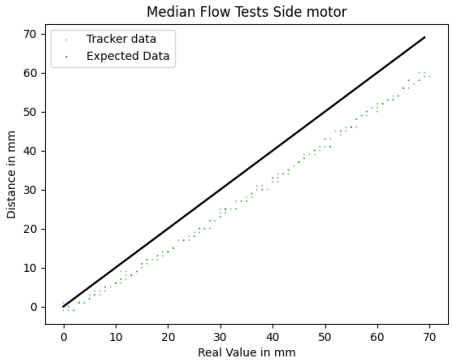
(b) Test2 side motor



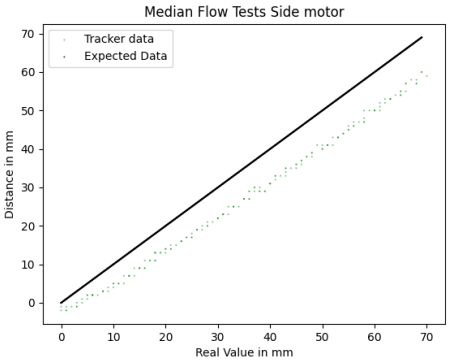
(c) Test3 side motor



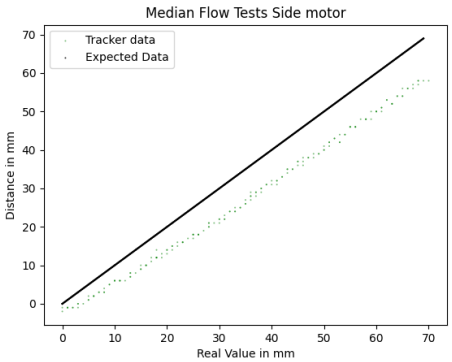
(d) Test4 side motor



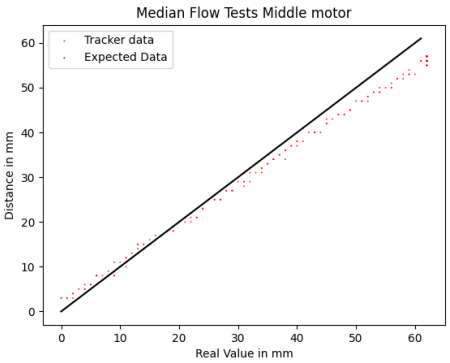
(e) Test5 side motor



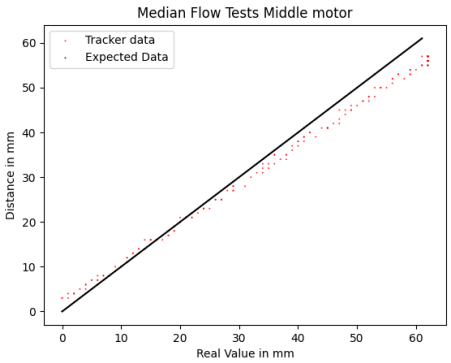
(f) Test6 side motor



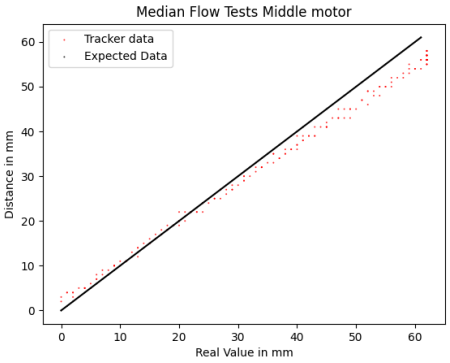
(g) Test7 side motor



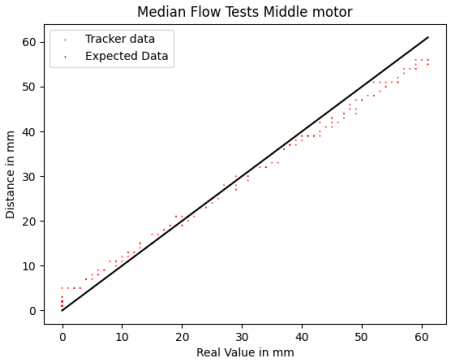
(a) Test1 middle motor



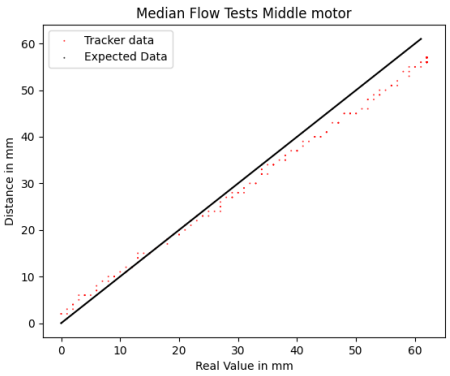
(b) Test2 middle motor



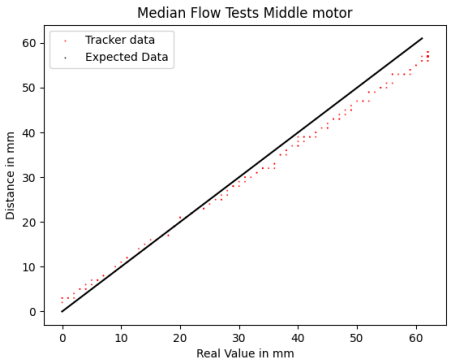
(c) Test3 middle motor



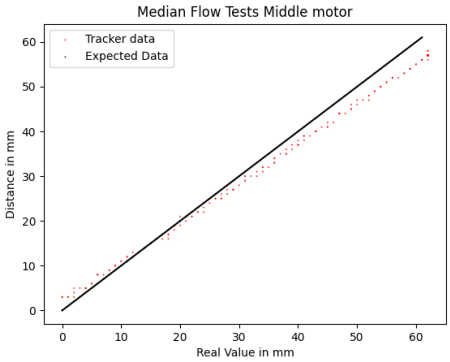
(d) Test4 middle motor



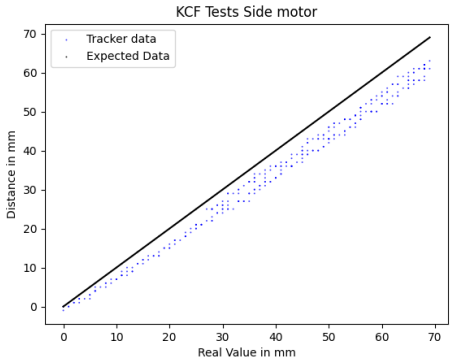
(e) Test5 middle motor



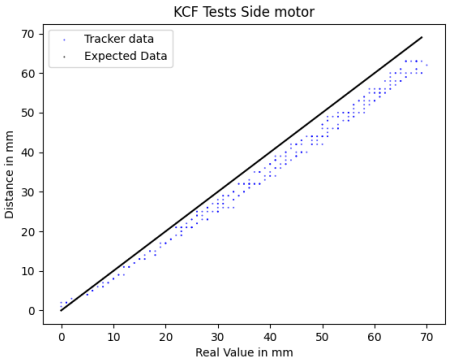
(f) Test6 middle motor



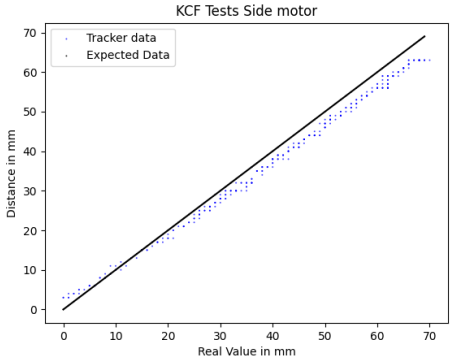
(g) Test7 middle motor



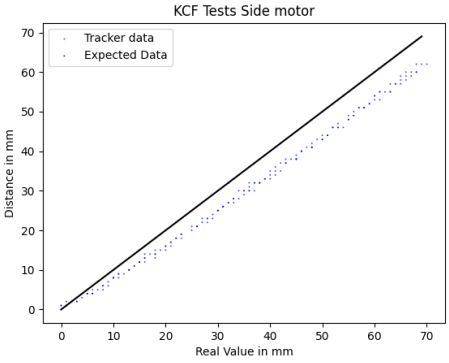
(a) Test1 side motor



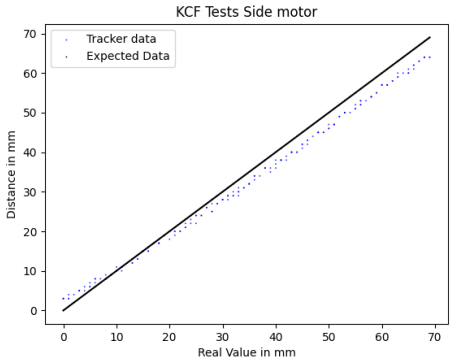
(b) Test2 side motor



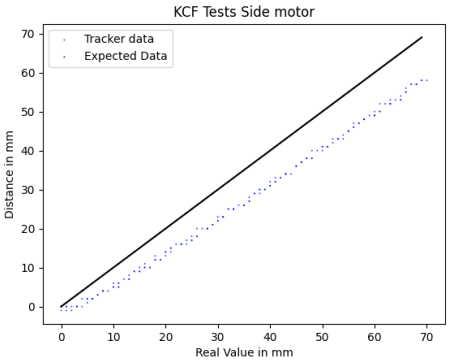
(c) Test3 side motor



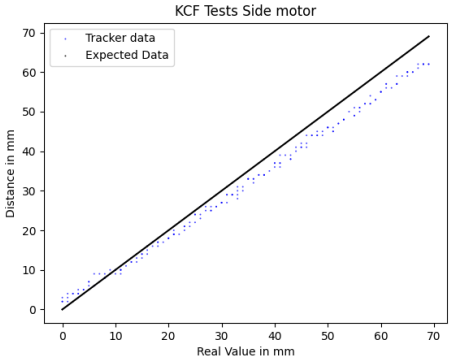
(d) Test4 side motor



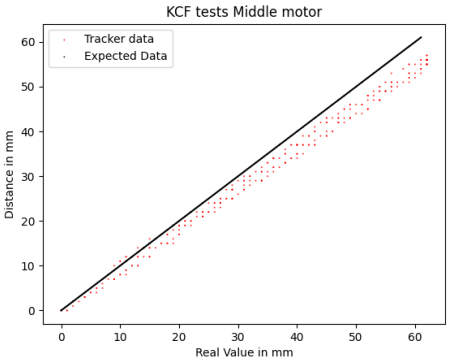
(e) Test5 side motor



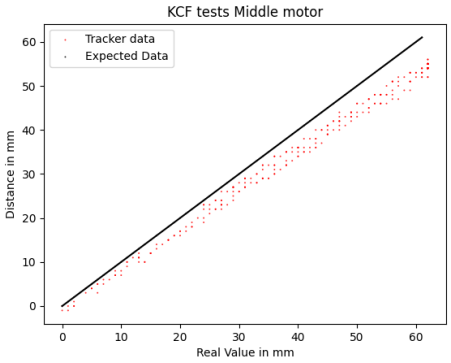
(f) Test6 side motor



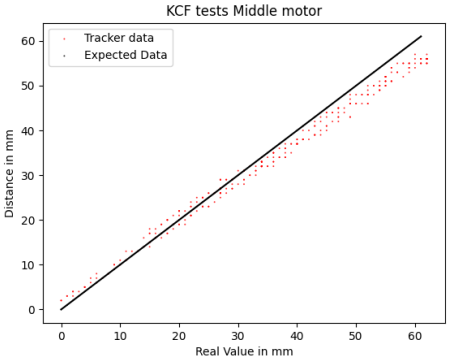
(g) Test7 side motor



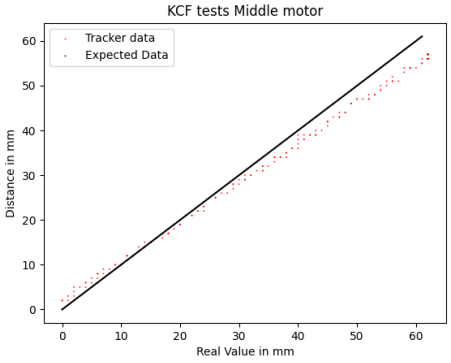
(a) Test1 middle motor



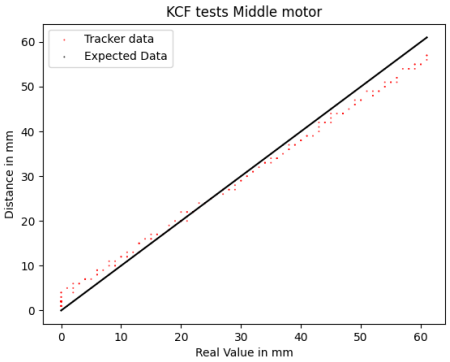
(b) Test2 middle motor



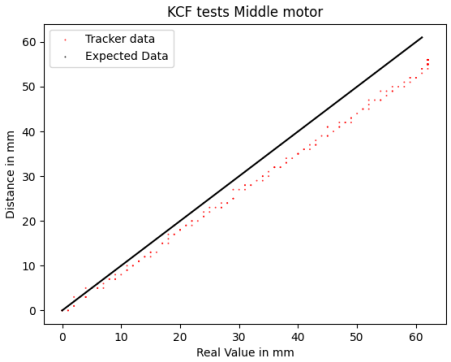
(c) Test3 middle motor



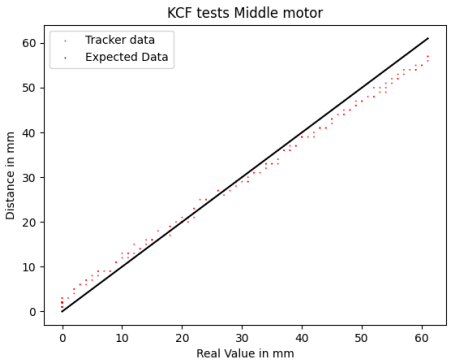
(d) Test4 middle motor



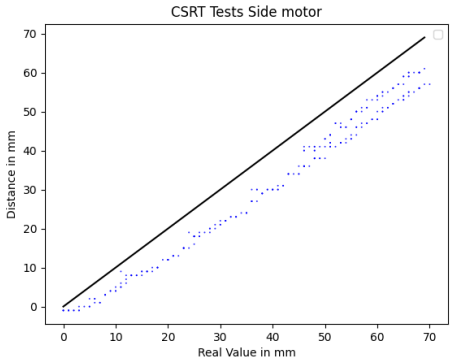
(e) Test5 middle motor



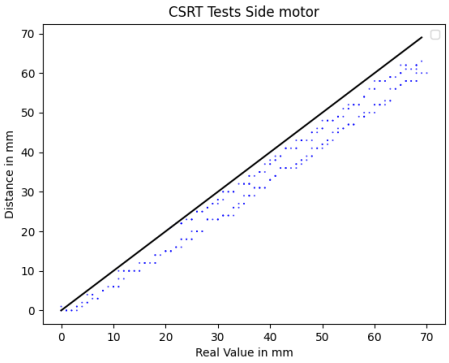
(f) Test6 middle motor



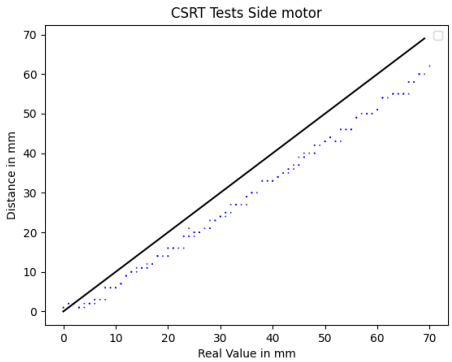
(g) Test7 middle motor



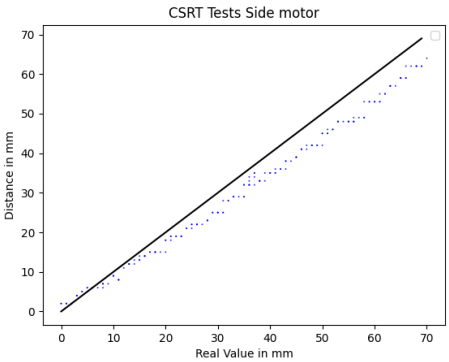
(a) Test1 side motor



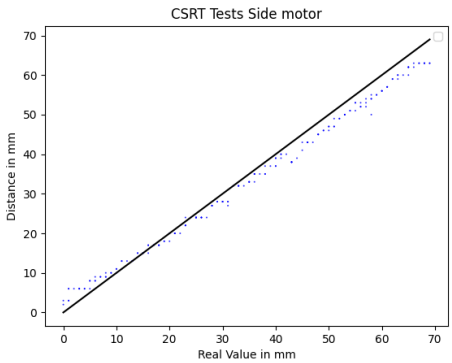
(b) Test2 side motor



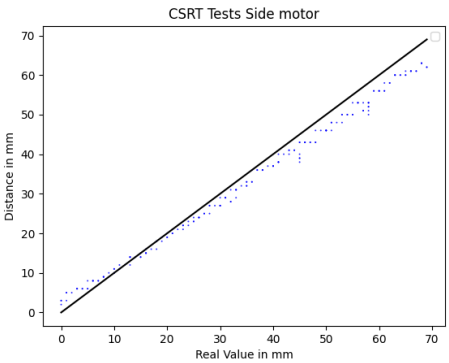
(c) Test3 side motor



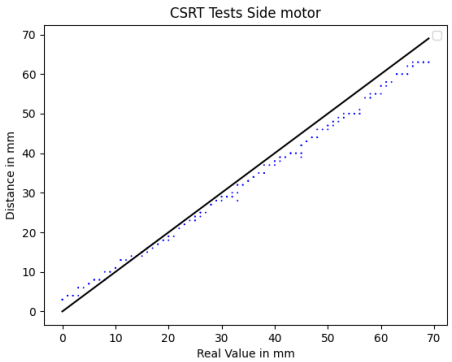
(d) Test4 side motor



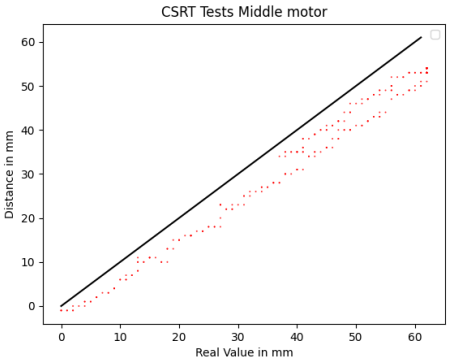
(e) Test5 side motor



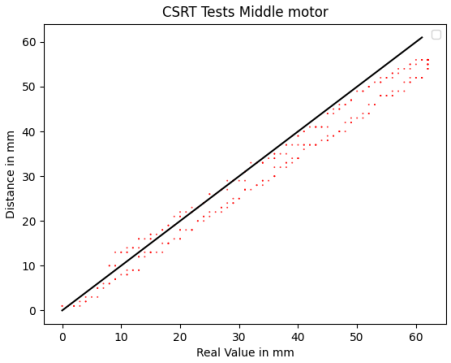
(f) Test6 side motor



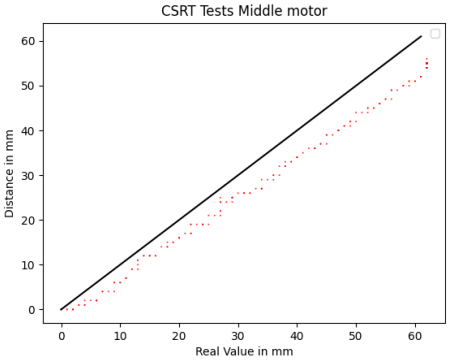
(g) Test7 side motor



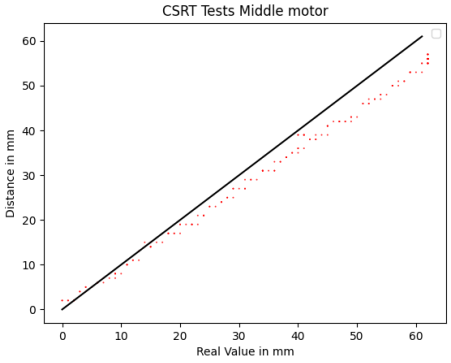
(a) Test1 middle motor



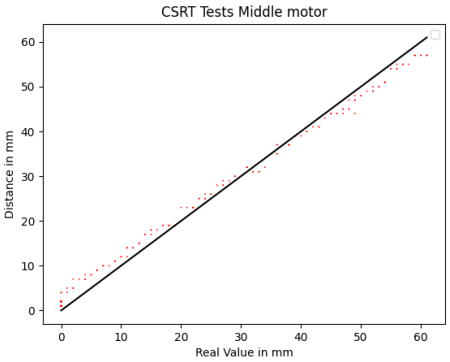
(b) Test2 middle motor



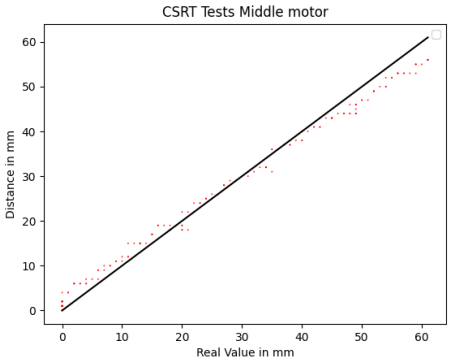
(c) Test3 middle motor



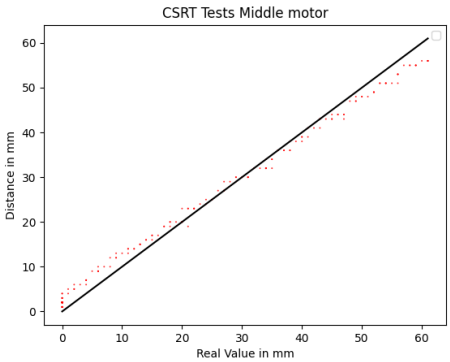
(d) Test4 middle motor



(e) Test5 middle motor



(f) Test6 middle motor



(g) Test7 middle motor

B Figures

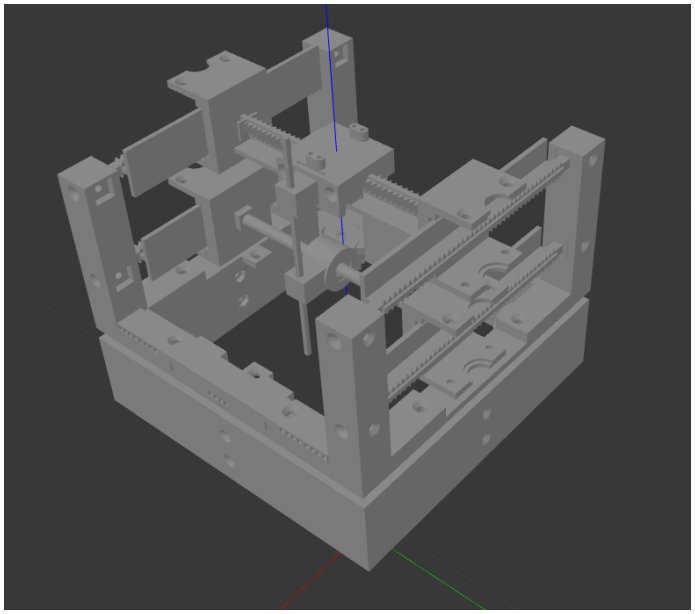


Figure B.1: The closed-loop robot in gazebo

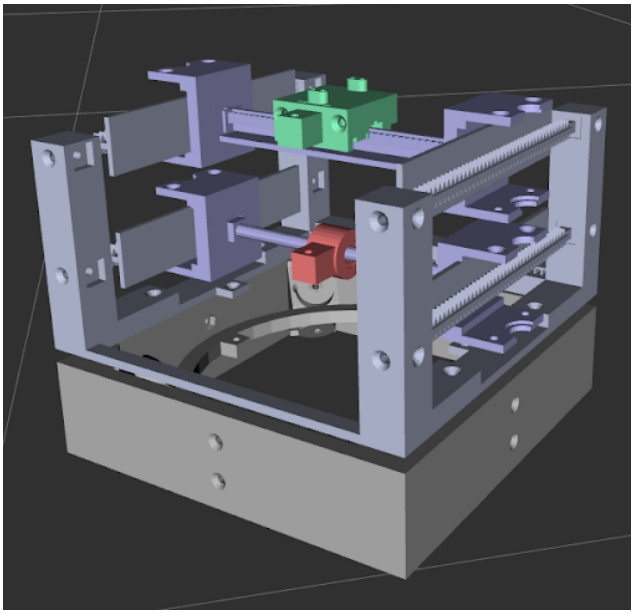


Figure B.2: The open loop robot in gazebo

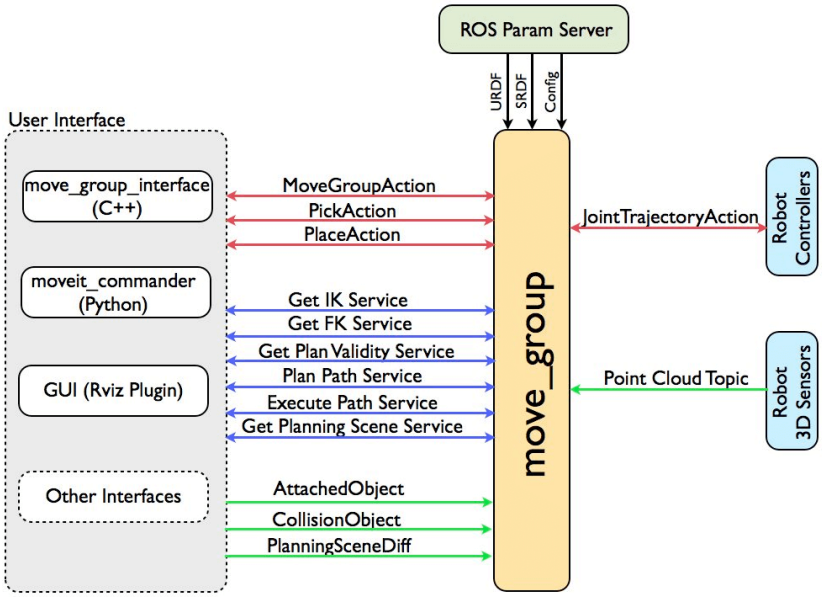


Figure B.3: Interface of the central MoveIt node noted as *move_group*.

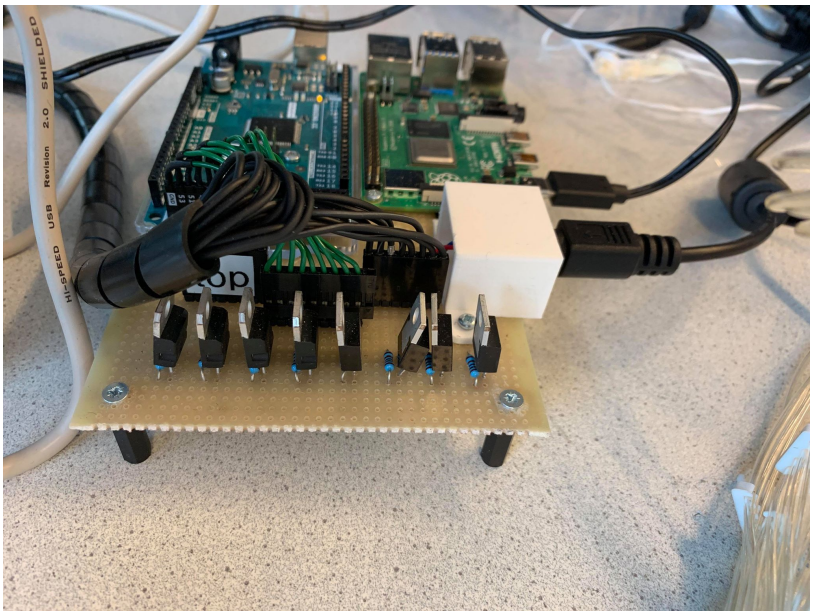


Figure B.4: The constructed circuit needed to connect the motor valves to the Arduino pins.

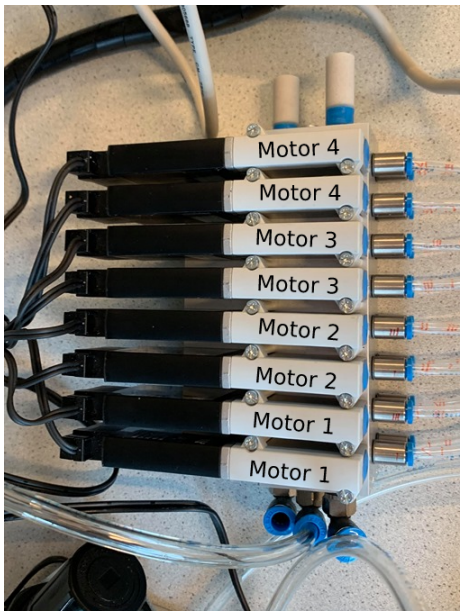


Figure B.5: The used motor valves in the current implementation.

C Motion of a rigid body

This appendix will be an explanation of the kinematics of the rigid body, forward kinematics and calculation of geometric Jacobian. This material is taken from lecture notes on geometry and screw theory for robotics. Stramigioli and Bruyninckx (2001)

C.0.1 Kinematics of rigid body

The pose of a rigid body with respect to the inertial reference frame is expressed as a configuration matrix in the Lie group SE(3) in equation C.1. In this equation R_1^0 denotes the rotation matrix in of the body with respect to the reference frame, while p_1^0 denotes the position of the body with respect to the inertial frame. The schematic of the reference and inertial frames can be seen in figure C.1.

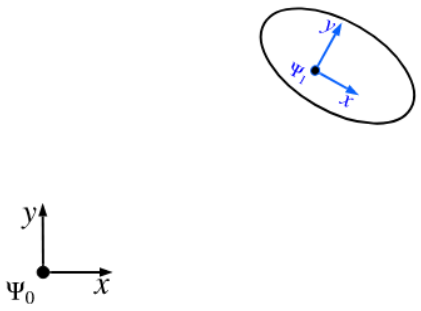


Figure C.1: Schematic showing the fixed world frame and rigid body's inertial frame.

$$H_1^0 = \begin{bmatrix} R_1^0 & p_1^0 \\ 0_3 & 1 \end{bmatrix} \tag{C.1}$$

The general velocity of the rigid body with respect to the reference frame expressed in the reference frame is written as a twist in equation C.2.

$$T_1^{0,0} = \begin{bmatrix} \vec{w} \\ \vec{v} \end{bmatrix} \tag{C.2}$$

C.0.2 Forward kinematics

Brocket's formula in equation C.3 can be used to find the position of the end-effector of a serial manipulator. The formula calculates the end-effector position by using the joints position in the joint space, the calculated unit twists of the joints, and the initial configuration of the robot. The unit twists are calculated using the manipulators inital configuration in figure C.2. The tilde form of the twist mentioned in the equation C.3 can be found by using equation C.4.

$$H_i^{i-1}(\theta) = e^{(i-1)\widetilde{T}_i^{i-1}\theta_i} H_i^{i-1}(0) \tag{C.3}$$

$$\widetilde{T}_1^{0,0} = \dot{H}_1^0 (H_1^0)^{-1}) \tag{C.4}$$

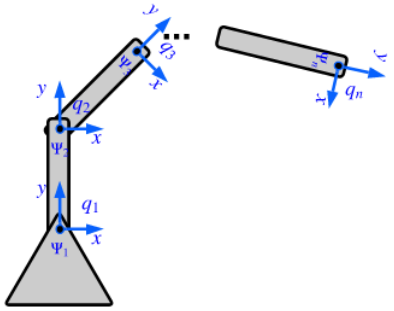


Figure C.2: Schematic showing the frames of a serial kinematic chain.

Tho find the homogeneous matrix of the initial configuration in figure C.2 the chain rule in equation C.5 is used.

$$H_i^{i-1}(\theta) = H_1^0 H_2^1 .... H_{ee}^{ee-1} \tag{C.5}$$

Lastly the product of exponentials in equation C.6 is used to calculate the pose of the end-effector in a specific configuration, where theta is the value of the joints.

$$H_{ee}^0(\theta) = e^{\tilde{T}_1^{0,0}\theta_1} e^{\tilde{T}_2^{0,1}\theta_2} e^{\tilde{T}_3^{0,2}\theta_3} e^{\tilde{T}_4^{0,3}\theta_3} e^{\tilde{T}_5^{0,4}\theta_4} H_{ee}^0(0) \tag{C.6}$$

C.0.3 Geometric Jacobian

The geometric Jacobian is a mapping from the joints' velocities to the end-effector twist of a serial manipulator, as shown in equation C.8 .

$$T_n^{0,0} = J(\theta)\dot{\theta} \tag{C.7}$$

The columns of the Jacobian are the position dependent unit twists of the joints expressed in the reference frame as shown in equation

$$J(\theta) = \begin{bmatrix} T_1^{0,0}(\theta_1) & T_2^{0,1}(\theta_2) & .... & T_{ee}^{0,ee-1}(\theta_n) \end{bmatrix} \tag{C.8}$$

Bibliography

Bekku, A., J. Kim, Y. Nakajima and K. Yonenobu (2014), A body-mounted surgical assistance robot for minimally invasive spinal puncture surgery, in *5th IEEE RAS/EMBS International Conference on Biomedical Robotics and Biomechatronics*, pp. 19–23, doi:10.1109/BIOROB.2014.6913745.

Biswas, D., H. Su, C. Wang and A. Stevanovic (2019), Speed Estimation of Multiple Moving Objects from a Moving UAV Platform, *International Journal of Geo-Information*, **vol. 2019**, p. 259, doi:10.3390/ijgi8060259.

Bui, M. T., R. Doskocil and V. Krivanek (2018), Distance and angle measurement using monocular vision, in *2018 18th International Conference on Mechatronics - Mechatronika (ME)*, pp. 1–6.

Charalampaki, E. and A. Malamos (2017), Tracking football players with a conventional mobile device camera, pp. 1–2, ISBN 978-1-4503-5355-7, doi:10.1145/3139367.3139411.

DANDIL, E. and K. K. ÇEVİK (2019), Computer Vision Based Distance Measurement System using Stereo Camera View, in *2019 3rd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, pp. 1–4, doi:10.1109/ISMSIT.2019.8932817.

Dehghani, H., S. Zhang, P. Kulkarni, P. Biswas, L. Simms and S.-E. Song (2018), Design and Simulation of Robotic Needle Guide for Transperineal Prostate Biopsy.

Ehambram, A., P. Hemme and B. Wagner (2019), An Approach to Marker Detection in IR- and RGB-images for an Augmented Reality Marker, pp. 190–197, doi:10.5220/0007810301900197.

Gassert, R., D. Chapuis, H. Bleuler and E. Burdet (2008), Sensors for Applications in Magnetic Resonance Environments, **vol. 13**, no.3, pp. 335–344, doi:10.1109/TMECH.2008.924113.

Ghelfi, J., A. moreau gaudry, N. Hungr, C. Fouard, B. Véron, M. Medici, E. Chipon, P. Cinquin and I. Bricault (2018), Evaluation of the Needle Positioning Accuracy of a Light Puncture Robot Under MRI Guidance: Results of a Clinical Trial on Healthy Volunteers, *CardioVascular and Interventional Radiology*, **vol. 41**, doi:10.1007/s00270-018-2001-5.

Gong, J., Y. Mei and Y. Zhou (2020), Research on an Improved KCF Target Tracking Algorithm Based on CNN Feature Extraction, in *2020 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*, pp. 538–543, doi:10.1109/ICAICA50127.2020.9182522.

Groenhuis, V., F. Siepel and S. Stramigioli (2018), Dual-Speed MR Safe Pneumatic Stepper Motors, robotics: Science and Systems 2018, RSS 2018 ; Conference date: 26-06-2018 Through 30-06-2018.
http://roboticsconference.org/

Hata, N., S.-E. Song, O. Olubiyi, Y. Arimitsu, K. Fujimoto, T. Kato, K. Tuncali, S. Tani and J. Tokuda (2016), Body-mounted robotic instrument guide for image-guided cryotherapy of renal cancer, *Medical Physics*, **vol. 43**, pp. 843–853, doi:10.1118/1.4939875.

Herrmann, K.-H., C. Gärtner, D. Güllmar, M. Krämer and J. Reichenbach (2014), 3D printing of MRI compatible components: Why every MRI research group should have a low-budget 3D printer, *Medical engineering physics*, **vol. 36**, doi:10.1016/j.medengphy.2014.06.008.

Hungr, N., I. Bricault, P. Cinquin and C. Fouard (2016), Design and Validation of a CT- and MRI-Guided Robot for Percutaneous Needle Procedures, **vol. 32**, no.4, pp. 973–987, doi:10.1109/TRO.2016.2588884.

Kang, M., C. Lee, B. You and Y. Chung (2015), A 3D object measurement method using a single view camera, in *2015 International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 790–792, doi:10.1109/ICTC.2015.7354666.

Lehtola, V., H. Huttunen, F. Christophe and T. Mikkonen (2017), Evaluation of Visual Tracking Algorithms for Embedded Devices, pp. 88–97, ISBN 978-3-319-59125-4, doi:10.1007/978-3-319-59126-1_8.

Mane, S. S. and C. G. Yangandul (2016), Calculating the dimensions of an object using a single camera by learning the environment, in *2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*, pp. 457–460, doi:10.1109/ICATCCT.2016.7912042.

Meng, Z., X. Kong, L. Meng and H. Tomiyama (2018), Distance Measurement and Camera Calibration based on Binocular Vision Technology, in *2018 International Conference on Advanced Mechatronic Systems (ICAMechS)*, pp. 342–347, doi:10.1109/ICAMechS.2018.8506743.

Min-jeong Kang, Choong-Ho Lee, Jin-Hwan Kim and Uk-Youl Huh (2008), Distance and velocity measurement of moving object using stereo vision system, in *2008 International Conference on Control, Automation and Systems*, pp. 2181–2184, doi:10.1109/ICCAS.2008.4694460.

Monfaredi, R., E. Wilson, R. Sze, K. Sharma, B. Azizi, I. Iordachita and K. Cleary (2015), Shoulder-Mounted Robot for MRI-guided arthrography: Accuracy and mounting study, in *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 3643–3646, doi:10.1109/EMBC.2015.7319182.

Mustafah, Y. M., R. Noor, H. Hasbi and A. W. Azma (2012a), Stereo vision images processing for real-time object distance and size measurements, in *2012 International Conference on Computer and Communication Engineering (ICCCE)*, pp. 659–663, doi:10.1109/ICCCE.2012.6271270.

Mustafah, Y. M., R. Noor, H. Hasbi and A. W. Azma (2012b), Stereo vision images processing for real-time object distance and size measurements, in *2012 International Conference on Computer and Communication Engineering (ICCCE)*, pp. 659–663, doi:10.1109/ICCCE.2012.6271270.

Orlando, F. and M. Joseph (2017), Development of closed loop coordinated control of a robot guided SMA actuated flexible active needle with multimodal sensory feedbacks, in *IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*, pp. 2846–2851, doi:10.1109/IECON.2017.8216480.

Pallapotu, K. (2019), DATA SET GENERATION USING DEEP LEARNING ALGORITHMS AND VISUAL FEATURE TRACKING.

Patel, N. A., J. Yan, D. Levi, R. Monfaredi, K. Cleary and I. Iordachita (2018), Body-Mounted Robot for Image-Guided Percutaneous Interventions: Mechanical Design and Preliminary Accuracy Evaluation, in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1443–1448, doi:10.1109/IROS.2018.8593807.

Raghava, N., K. Gupta, I. Kedia and A. Goyal (2020), An Experimental Comparison of Different Object Tracking Algorithms, in *2020 International Conference on Communication and Signal Processing (ICCSP)*, pp. 0726–0730, doi:10.1109/ICCSP48568.2020.9182101.

Rani, A., V. Maik and B. Chithravathi (2017), Robust object tracking using kernalized correlation filters (KCF) and Kalman predictive estimates, pp. 587–591, doi:10.1109/RTEICT.2017.8256664.

RealSense, I. (2020), Product Family D400 Series.

Rebersek, M., D. Miklavcic, C. Bertacchini and M. Sack (2014), Cell membrane electroporation-Part 3: the equipment, **vol. 30**, no.3, pp. 8–18, doi:10.1109/MEI.2014.6804737.

Shen, J., D. Yu, L. Deng and X. Dong (2018), Fast Online Tracking With Detection Refinement, **vol. 19**, no.1, pp. 162–173, doi:10.1109/TITS.2017.2750082.

Stramigioli, S. and H. Bruyninckx (2001), Geometry and screw theory for robotics, p. 75.

Tannús, J. (2020), Comparison of OpenCV Tracking Algorithms for a Post-Stroke Rehabilitation Exergame, in *2020 22nd Symposium on Virtual and Augmented Reality (SVR)*, pp. 272–276, doi:10.1109/SVR51698.2020.00049.

Tokuda, J., L. Chauvin, B. Ninni, T. Kato, F. King, K. Tuncali and N. Hata (2018), Motion compensation for MRI-compatible patient-mounted needle guide device: Estimation of targeting accuracy in MRI-guided kidney cryoablations, *Physics in Medicine and Biology*, **vol. 63**, doi:10.1088/1361-6560/aab736.

Ullah, K., I. Ahmed, M. Ahmad and I. Khan (2019), Comparison of Person Tracking Algorithms Using Overhead View Implemented in OpenCV, doi:10.1109/IEMECONX.2019.8877025.

Watkins, C., T. Kato and N. Hata (2016), Disposable patient-mounted geared robot for image-guided needle insertion, in *Medical Imaging 2016: Image-Guided Procedures, Robotic Interventions, and Modeling*, volume 9786 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, Eds. R. J. Webster and Z. R. Yaniv, p. 97860S, doi:10.1117/12.2216779.

Wei Liu, Xin Ma, Ling Chen, Jinghao Yang and Zhenyuan Jia (2016), A monocular vision 3D measurement method based on refraction of light, in *2016 IEEE 25th International Symposium on Industrial Electronics (ISIE)*, pp. 880–884, doi:10.1109/ISIE.2016.7745006.

Yao, R., S. Xia, F. Shen, Y. Zhou and Q. Niu (2016), Exploiting Spatial Structure from Parts for Adaptive Kernelized Correlation Filter Tracker, **vol. 23**, no.5, pp. 658–662, doi:10.1109/LSP.2016.2545705.