Sampling-based Algorithms for Surgical Task with Remote Center of Motion Constraint

Akhil Bandamidapalli (ME20B016), Prithvi Raj B (ME20B137)
Dept. of Mechanical Engineering, Indian Institute of Technology - Madras

***Abstract*—In this project, we conducted a comparative study of different motion planning algorithms for surgical tasks. In the second part, we incorporate a remote center of motion (RCM) constraint. We calculate forward kinematics using the DH parameters of the KUKA IIWA arm and utilize an inverse kinematic solver to generate points that satisfy the RCM constraint. These motions are then simulated in a PyBullet environment, and the performance of the algorithms is evaluated.**

***Index Terms*—Sampling-based planners, Remote Center of Motion, Inverse Kinematic solver, PyBullet, KUKA IIWA**

I. Introduction

Sampling-based motion planners are algorithms used for motion planning in higher dimensional spaces, making them highly applicable in surgical scenarios. An additional constraint is the enforcement of the remote center of motion (RCM), a fixed point in space around which a mechanism or robotic device can move. Typically located outside the device, the RCM acts as a pivot point, enabling the device to maintain a fixed relationship with the surrounding environment while moving.

Our work focuses on the issue of transitioning from RCM sampling to sampling in the task space, which allows for quicker search for valid solutions. This presents a significant speed improvement compared to existing joint sampling methods, where sampling occurs in the configuration space. However, this transition involves using an IK solver (Levenberg-Marquardt). In addition to the alternative sampling method, we also manually enforce the constraints of the RCM.

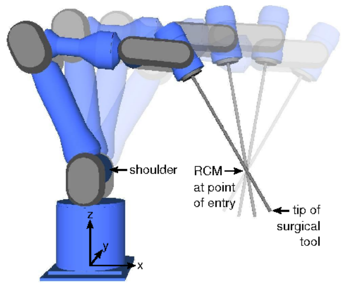


Fig. 1: Remote Center of Motion, Image Credits : [1]

II. Literature Review

Since the introduction of Lavalle et al.'s Rapidly Exploring Random Trees (RRT) algorithm [2], several variants have been developed. Kuffner et al.'s RRT-Connect [3], uses bidirectional trees to expedite the search for a viable path. The RRT* algorithm [4] incorporates a rewiring step to optimize the path by rearranging the trees. When these algorithms are applied to surgical tasks, additional constraints, such as the remote center of motion (RCM) constraint, need to be considered. Sandoval et al.'s work [5] presents a novel approach to framing the RCM constraint, while Zhou et al.'s work [6] demonstrates a new mechanism for implementing the RCM constraint. Marinho et al.'s work [7] compares between the different algorithms that can be used for RCM generation.

The project consists of two plans. Plan 1 involves executing three algorithms- RRT, RRT*, and Bi-RRT - to plan a path from one joint configuration to another. Plan 2 involves executing a remote center of motion with an RRT planner. The movements have been carried out using a PyBullet simulator on a KUKA IIWA 7 arm.

III. Methodology

A. Forward and Inverse Kinematics

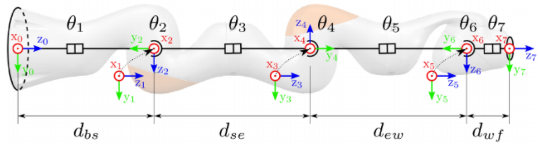
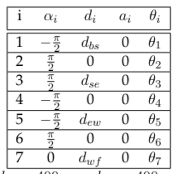


Figure 0.1: Illustration of the robot arm used (7-DOF) without gripper and the alignment of the coordinate systems as well as the dimensions. In my case the arm is vertically mounted.



i	α_i	d_i	a_i	θ_i
1	$-\frac{\pi}{2}$	d_{bs}	0	θ_1
2	$\frac{\pi}{2}$	0	0	θ_2
3	$\frac{\pi}{2}$	d_{se}	0	θ_3
4	$-\frac{\pi}{2}$	0	0	θ_4
5	$-\frac{\pi}{2}$	d_{ew}	0	θ_5
6	$\frac{\pi}{2}$	0	0	θ_6
7	0	d_{wf}	0	θ_7

with $d_{bs} = 340mm$, $d_{se} = 400mm$, $d_{ew} = 400mm$ and $d_{wf} = 126mm$

Fig. 2: KUKA IIWA 7 and its DH parameters

The project's development begins with setting up a forward and inverse kinematic solver. Since the KUKA IIWA 7 arm is a 7 DOF arm, we plan by using seven joint angles simultaneously, given by

$$q = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6, \theta_7]^T \tag{1}$$

We use the transformation matrices T_i^{i-1} to perform the forward kinematic. The kinematic equation of a serial manipulator with 7 links, with corresponding joint parameters, θ_i, is given by,

$$T_7^0 = \prod_{i=1}^{7} T_i^{i-1}(\theta_i) \tag{2}$$

where, T_i^{i-1} is the transformation matrix from link i to $i-1$.

Once we obtain T_n^0 for each link, the position of the joint is simply the first 3 rows of the 4th column, which can be obtained as,

$$p_7^0 = T_7^0 \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^T \tag{3}$$

where the first 3 rows of p_7^0 give the coordinates of the end-effector of link 7.

Inverse kinematics was set using an inbuilt Levenberg-Marquardt solver [8] in the Robotics Toolbox [9]. The solver uses damped least squares methods to solve after setting up a non-linear least squares problem

B. Environment

The environment has been set up for two scenarios. In scenario 1, a humanoid URDF is kept on top of a table to simulate a surgical operation accurately. In scenario 2, to test the RCM constraint and reduce the system's complexity, we have a box of dimensions $700mm \times 300mm \times 300mm$ with an opening or radius of $25mm$ on the top of the center face. The KUKA IIWA 7 is then controlled using PyBullet's inbuilt controllers. The environment can be seen in figure 3.

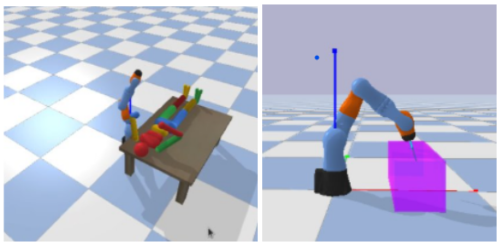


Fig. 3: To the left is Scenario 1, to the right is Scenario 2

C. Algorithms

For path planning, the following RRT algorithms are implemented:

1) RRT: Classical RRT or Rapidly Exploring Random Tree is a sampling based algorithm that uses random samples in the configuration space to expand a tree and discover a path. The algorithm is given at Algorithm 1.

2) RRT:* RRT* is a variant of RRT which ensures that paths found are asymptotically optimal. This means that as the number of iterations goes to infinity, the path found by RRT* converges to the optimal path. It accomplishes this by maintaining a cumulative cost at each node and rewiring nodes in a specified radius of the newly added to have a lower cumulative cost. The algorithm is given in Algorithm 2.

3) Bi-directional RRT: Bi-directional RRT maintains two trees to explore points: one from the start and one from the goal. When a point is expanded on one tree, the nearest node to that point on the other tree tries to connect to it by stepping towards it till the point is reached or a collision is detected. The algorithm is given at Algorithm 3.

Algorithm 1: Rapidly-exploring Random Tree (RRT)

```
Require: Initial position $x_{\text{init}}$, goal position $x_{\text{goal}}$, number of
    iterations $N$, step size $\delta t$
 1: $T \leftarrow \{x_{\text{init}}\}$
 2: for $i = 1$ to $N$ do
 3:     $x_{\text{rand}} \leftarrow$ RandomSample()
 4:     $x_{\text{nearest}} \leftarrow$ Nearest$(T, x_{\text{rand}})$
 5:     $x_{\text{new}} \leftarrow$ Steer$(x_{\text{nearest}}, x_{\text{rand}}, \delta t)$
 6:     if CollisionFree$(x_{\text{nearest}}, x_{\text{new}})$ then
 7:         $T \leftarrow T \cup \{x_{\text{new}}\}$
 8:     end if
 9:     if Distance$(x_{\text{new}}, x_{\text{goal}}) \leq \delta t$ then
10:         return Path$(T, x_{\text{init}}, x_{\text{goal}})$
11:     end if
12: end for
13: return Failure
```

Algorithm 2: RRT* (RRT-Star)

```
Require: Initial position $x_{\text{init}}$, goal position $x_{\text{goal}}$, number of
    iterations $N$, step size $\delta t$, rewire radius $r$
 1: $T \leftarrow \{x_{\text{init}}\}$
 2: for $i = 1$ to $N$ do
 3:     $x_{\text{rand}} \leftarrow$ RandomSample()
 4:     $x_{\text{nearest}} \leftarrow$ Nearest$(T, x_{\text{rand}})$
 5:     $x_{\text{new}} \leftarrow$ Steer$(x_{\text{nearest}}, x_{\text{rand}}, \delta t)$
 6:     if CollisionFree$(x_{\text{nearest}}, x_{\text{new}})$ then
 7:         $X_{\text{near}} \leftarrow$ Near$(T, x_{\text{new}}, r)$
 8:         $x_{\text{min}} \leftarrow x_{\text{nearest}}$
 9:         $c_{\text{min}} \leftarrow$ Cost$(x_{\text{nearest}})$ + Cost$(x_{\text{nearest}}, x_{\text{new}})$
10:         for each $x_{\text{near}}$ in $X_{\text{near}}$ do
11:             if CollisionFree$(x_{\text{near}}, x_{\text{new}})$ AND
                Cost$(x_{\text{near}})$ + Cost$(x_{\text{near}}, x_{\text{new}}) < c_{\text{min}}$ then
12:                 $x_{\text{min}} \leftarrow x_{\text{near}}$
13:                 $c_{\text{min}} \leftarrow$ Cost$(x_{\text{near}})$ + Cost$(x_{\text{near}}, x_{\text{new}})$
14:             end if
15:         end for
16:         $T \leftarrow T \cup \{(x_{\text{new}}, x_{\text{min}})\}$
17:         for each $x_{\text{near}}$ in $X_{\text{near}}$ do
18:             if CollisionFree$(x_{\text{new}}, x_{\text{near}})$ AND
                Cost$(x_{\text{new}})$ + Cost$(x_{\text{new}}, x_{\text{near}}) <$ Cost$(x_{\text{near}})$ then
19:                 $x_{\text{parent}} \leftarrow$ Parent$(x_{\text{near}})$
20:                 Remove$(T, (x_{\text{near}}, x_{\text{parent}}))$
21:                 Add$(T, (x_{\text{near}}, x_{\text{new}}))$
22:             end if
23:         end for
24:     end if
25:     if Distance$(x_{\text{new}}, x_{\text{goal}}) \leq \delta t$ then
26:         return Path$(T, x_{\text{init}}, x_{\text{goal}})$
27:     end if
28: end for
29: return Failure
```

Algorithm 3: Bi-Directional RRT

```
Require: Initial position $x_{\text{init}}$, goal position $x_{\text{goal}}$, number of
    iterations $N$, step size $\delta t$
Ensure: Path from $x_{\text{init}}$ to $x_{\text{goal}}$
 1: $T_{\text{init}} \leftarrow \{x_{\text{init}}\}$
 2: $T_{\text{goal}} \leftarrow \{x_{\text{goal}}\}$
 3: for $i = 1$ to $N$ do
 4:     $x_{\text{rand}} \leftarrow$ RandomSample()
 5:     $x_{\text{nearest}} \leftarrow \text{Nearest}(T_{\text{init}}, x_{\text{rand}})$
 6:     $x_{\text{new}} \leftarrow \text{Steer}(x_{\text{nearest}}, x_{\text{rand}}, \delta t)$
 7:     if CollisionFree$(x_{\text{nearest}}, x_{\text{new}})$ then
 8:         $T_{\text{init}} \leftarrow T_{\text{init}} \cup \{x_{\text{new}}\}$
 9:         $x_{\text{new\_nearest}} \leftarrow \text{Nearest}(T_{\text{goal}}, x_{\text{new}})$
10:         while True do
11:             $x_{\text{new\_new}} \leftarrow \text{Steer}(x_{\text{new\_nearest}}, x_{\text{new}}, \delta t)$
12:             if CollisionFree$(x_{\text{new\_nearest}}, x_{\text{new\_new}})$ then
13:                 $T_{\text{goal}} \leftarrow T_{\text{goal}} \cup \{x_{\text{new\_new}}\}$
14:                 if $x_{\text{new\_new}} = x_{\text{new}}$ then
15:                     return Path$(T_{\text{init}}, T_{\text{goal}})$
16:                 end if
17:                 $x_{\text{new\_nearest}} \leftarrow x_{\text{new\_new}}$
18:             else
19:                 break
20:             end if
21:         end while
22:     end if
23:     Swap$(T_{\text{init}}, T_{\text{goal}})$
24: end for
25: return Failure
```

D. Constraints

1) RCM check: : We implement RCM by going back to the fundamental equations of RCM,

$$P_{rcm} = P_7 + \lambda (P_8 - P_7) \tag{4}$$

Differentiating,

$$\dot{P}_{rcm} = 0 = \dot{P}_7 + \dot{\lambda} (P_8 - P_7) + \lambda \left(\dot{P}_8 - \dot{P}_7\right) \tag{5}$$

where, P_{rcm} represents the position of the RCM, P_7 represents the end of the end-effector and P_8 represents the tip of the needle probe. λ is a parameter that describes the position of p_{rcm} on the needle probe. Hence, $\lambda \in [0, 1]$. Equation 5 is obtained by differentiating equation 4 and setting the RHS term to zero, as P_{rcm} is a fixed point and does not vary in position during the operation.

This leads to two cases, as shown in Figure 4,

Case 1: The first case describes a motion when the needle is perpendicular to the skin's surface. This condition allows the needle to move inside the surface along the normal to the skin, $\hat{n}$, or to rotate about P_{rcm}. Mathematically,

$$\dot{\lambda} = 0, (P_8 - P_7) \times \hat{n} = 0 \tag{6}$$

Case 2: The second case describes a more common case: the needle is not along the normal direction and can only move about its own P_{rcm}. Hence,

$$\dot{\lambda} = 0 \tag{7}$$

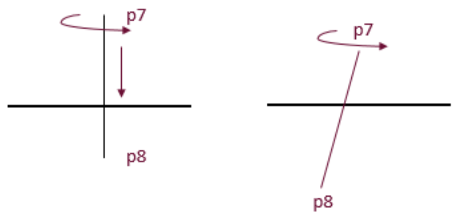


Fig. 4: Two distinct possible motions in RCM

2) Collision check: We check for collisions between the links and the obstacles in the environment by using a simple distance-to-link collision checker. While this isn't implicitly implemented in this project, it is assumed that it is satisfied by the other checks and the region of the task space that is being manipulated.

IV. Results

The algorithms were tested on two scenarios with collision, RCM constraints: first with the manipulator's starting position to the side of the table, second with the manipulator's initial position underneath the table. The tests were repeated 10 times, and the averages were taken after removing unusually high values.

A. Manipulator to the Side

The following results were obtained:

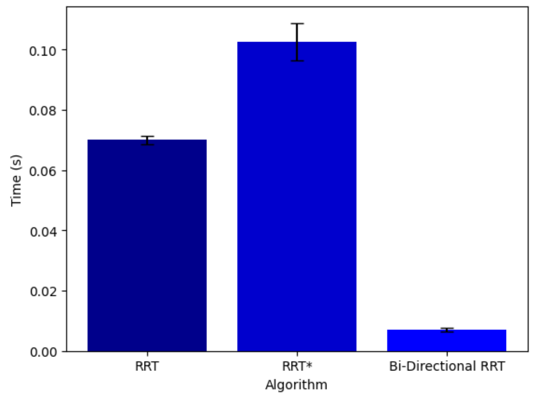


Fig. 5: Time taken for each algorithm in case A

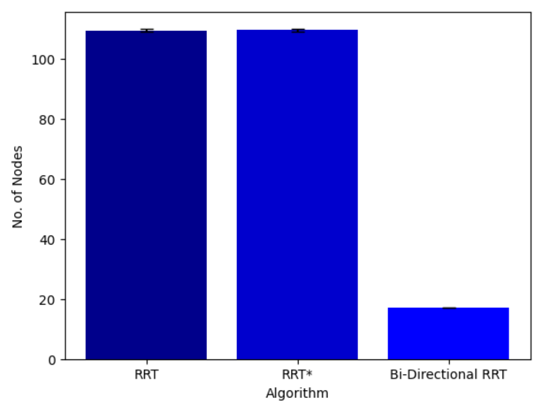


Fig. 6: Number of nodes expanded for each algorithm in case A

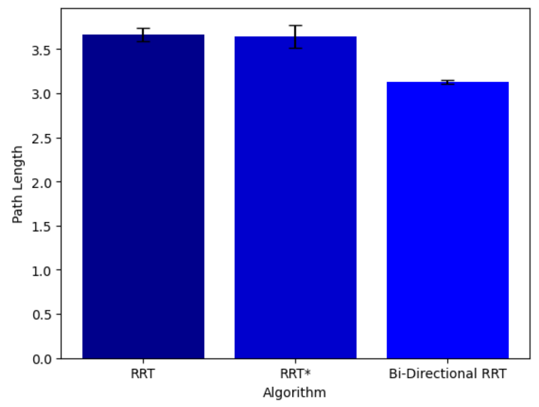


Fig. 7: Path length for each algorithm in case A

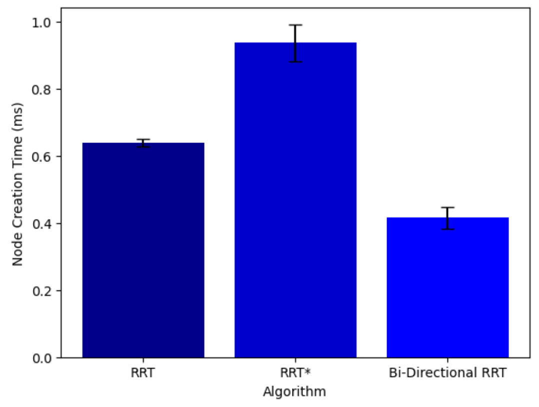


Fig. 8: Node creation time for each algorithm in case A

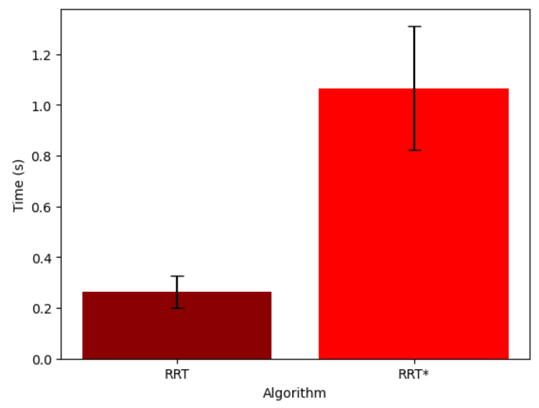


Fig. 9: Time taken in each algorithm for case B

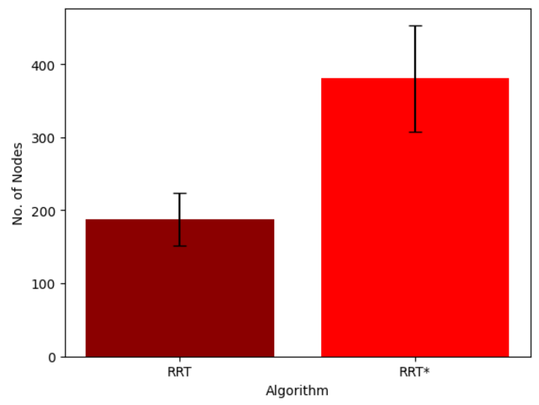


Fig. 10: Number of nodes in each algorithm for case B

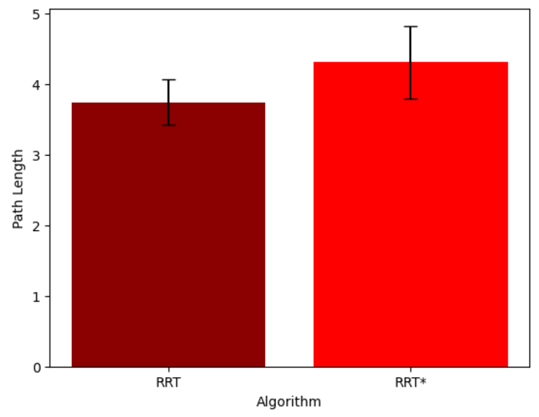


Fig. 11: Path length in each algorithm for case B

B. *Manipulator Underneath the Table*

It should be noted that Bi-Directional RRT did not give a solution for this case.

V. Discussions

Hence, the above algorithms have been tested and studied for the above parameters. We notice that RRT* takes approximately 40% to 60% more time to create a node than the other algorithms. This is naturally due to the rewiring nature of the algorithm. RRT* also takes more path length despite rewiring. This is because optimizing in configuration space does not directly correspond to optimization in the workspace.

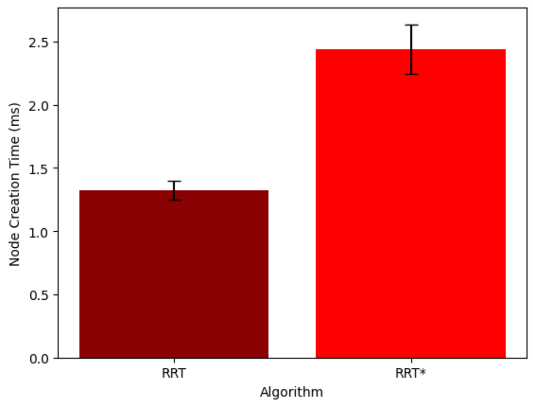


Fig. 12: Node creation time in each algorithm for case B

One could overcome this by simply redefining the cost for optimality, i.e., by setting the cost in the task space. In terms of finding a solution, Bi-Directional RRT could not find a solution in the second scenario because of its greedy nature. RRT and RRT* had goal biasing only every 100 iterations so they could explore more. There is much more variance in RRT*, as seen in the error bars in every graph.

While implementing RCM, we assumed the RCM point to have a tolerance radius of 2.5 cm. We assumed this because, in practice, it is difficult to sample points while also upholding the RCM constraint. The step size would have had to be lowered, drastically increasing the computation time to find a solution.

VI. Conclusions

The above results compare the optimality and computational efficiency in sampling-based motion planning algorithms, particularly in surgical applications with RCM constraints. While RRT* offers potential in path optimality, its practical application has higher computational demands and greater variance, as seen in the results. RRT provides more consistent and practical solutions in constrained environments. Bi-RRT provides faster solutions with shorter path lengths in environments with fewer obstacles. Future research may offer insights into tackling some of the disadvantages of each algorithm to meet the stringent requirements of surgical tasks.

Some of the challenges we faced and how we overcame them were,

1) Setting up the Simulation: : Setting up the simulation environment posed a lot of challenge. Coppeliasim gave us problems with collision detection. Finally we settled on PyBullet with direct Python integration.

2) An Appropriate Environment for Analysis : Too few obstacles between the manipulator's start and the goal meant that goal biasing gave an instant solution. In order to properly compare algorithms we needed an obstacle map that the algorithms could navigate but not easily.

3) Finding Solutions: Initially, no algorithm could find a solution for even a slightly complicated case. This was fixed by increasing the step size and goal radius.

VII. Contributions and attached codes & video

A. *Github Repository*

We have attached all the codes and important documents to this GitHub repository. One may go through the readme file for instructions on installation and usage.

B. *Video demonstration*

We have some videos to this Google drive briefly illustrate the project and running the code to explain the simulation results.

C. *Contributions*

- Akhil Bandamidapalli - Literature review, Environment setup, RCM constraint visualization and code, IK solver, Bi-RRT, Report writing -50%
- Prithvi Raj B - Environment Setup, Collision Checker, RRT, RRT*, Bi-RRT implementations with code, Result tabulation, Report writing -50%

VIII. Acknowledgment

We thank Prof. Bijo Sebastian and Prof. Nirav Patel for allowing us to explore path-planning algorithms in surgical robots through their Intro to Motion Planning course.

References

[1] Locke, R.C., & Patel, R.V. (2007). Optimal Remote Center-of-Motion Location for Robotics-Assisted Minimally-Invasive Surgery. Proceedings 2007 IEEE International Conference on Robotics and Automation, 1900-1905.
[2] LaValle, Steven. "Rapidly-exploring random trees: A new tool for path planning." Research Report 9811 (1998).
[3] Kuffner, James J., and Steven M. LaValle. "RRT-connect: An efficient approach to single-query path planning." Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065). Vol. 2. IEEE, 2000.
[4] Karaman, S., & Frazzoli, E. (2010, May 3). Incremental Sampling-based Algorithms for Optimal Motion Planning. arXiv.org. https://arxiv.org/abs/1005.0416
[5] J. Sandoval, G. Poisson and P. Vieyres, "A new kinematic formulation of the RCM constraint for redundant torque-controlled robots," 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 2017, pp. 4576-4581, 10.1109/IROS.2017.8206326.
[6] Zhou, X., Zhang, H., Feng, M. et al. New remote centre of motion mechanism for robot-assisted minimally invasive surgery. BioMed Eng OnLine 17, 170 (2018). https://doi.org/10.1186/s12938-018-0601-6
[7] M. M. Marinho, K. Harada and M. Mitsuishi, "Comparison of remote center-of-motion generation algorithms," 2017 IEEE/SICE International Symposium on System Integration (SII), Taipei, Taiwan, 2017, pp. 668-673, doi: 10.1109/SII.2017.8279298.
[8] Moré, J. J. (2006, August). The Levenberg-Marquardt algorithm: implementation and theory. In Numerical analysis: proceedings of the biennial Conference held at Dundee, June 28–July 1, 1977 (pp. 105-116). Berlin, Heidelberg: Springer Berlin Heidelberg.
[9] P.I. Corke, "Robotics, Vision & Control", Springer 2017, ISBN 978-3-319-54413-7
[10] Coumans, E., & Bai, Y. (2021). PyBullet quickstart guide.