Motion Planning of a Surgical Robot with Remote Centre of Motion Constraint

Sidharth Tadeparti, Tejas Rao, Shrung DN
Department of Mechanical Engineering, Indian Institute of Technology, Madras

***Abstract*—In this project, the remote centre of motion constraint is enforced using sampling based motion planning techniques. The forward kinematics are implemented using DH parameters of the manipulators. The RCM constraints are embedded through hand-written constructs. The motion is simulated in a Gazebo environment while making use of the sunrise controller. The performance of the various algorithms are compared to each other.**

***Index Terms*—Sampling Based Planners, Forward Kinematics, Gazebo, ROS, KUKA IIWA**

I. Introduction

Remote Centre of Motion (RCM) are of great importance for surgical robots which require surgical tools to be inserted into the patient's body through small incisions, and move without damaging surrounding tissues. The remote centre of motion (RCM) is a fixed point in space around which a mechanism or a robotic device can move. This point is typically located outside of the device, and it acts as a pivot point that allows the device to move in a way that preserves a fixed relationship with the surrounding environment.

A large number of surgical robots employ the use of physical mechanisms to enforce the RCM constraint. However, these mechanisms are often complex and have a reduced range of motion. In order to solve this, this project tries to enforce the RCM constraint algorithmically at the motion planning stage.

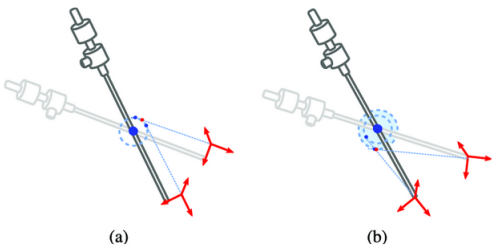


Fig. 1. Remote Centre of Motion Constraint, Image Credits : [1]

Literature Review

Classical work in [2] uses geometric methods for manipulation planning on constraint manifolds. However, such methods are at large restricted to lower-dimensional planning problems and are intractable when dealing with larger dimensions. The most common method used to establish the RCM constraint is through task space control. Work such as [3] uses differential inverse kinematics (jacobians of motion) to ensure that the RCM constrain is maintained while small motions are made in the task space. Additionally, RNNs have been used in [4] to improve constrained planning, albeit on a lower DoF manipulator.

This project provides an implementation of three motion planning algorithms: RRT, RRT Star, and Informed RRT Star and provides a comparison of the performances for this application. A 7 DOF KUKA IIWA manipulator is considered in this case along with a custom trocar.

II. Methodology

The development of this project involved the setup of the environment, the implementation of the motion planning algorithms and the supporting forward kinematics and collision check subroutines. The problem involves a 7 DoF manipulator and hence each of the 7 joint angles make up the 7 dimensional configuration space. The state may be given by,

$$\mathbf{Q} = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6, \theta_7]$$

,

where, θ_i, $i = 1...7$, represents the corresponding joint angle. The corresponding pose in the task space is computed using the forward kinematics obtained in the following user manual. The position and orientation of the points on the end-effector can be found from the final transformation matrix,

$$T_f = T_{7,1}.$$

Simultaneously collision is checked using a hand-written script while utilizing the previously described forward kinematics module. A main script ties all the modules together and published the planned path using the modified joint angle controller.

The flow of information between the various modules is given by Figure 6.

A. Environment

The environment builds on the one provided [5]. A custom controller, based on the sunrise controller is setup to enable joint angle control. In addition to this, the gazebo environment is modified so as to display the RCM constraint as a sphere. The robot's URDF is also modified to add the tool or trocar. The start and goal points are represented through spheres as well. A screenshot of the simulation environment prior to planning is given in figure 2

Fig. 2. Gazebo Environment

B. Algorithms

1) RRT: Randomly samples in C-Space to identify nodes in free space, and constructs a tree that eventually the start and goal nodes, if a path exists. The algorithm terminates as soon as a valid path is found.

Algorithm 1 RRT

Require: Start state Q_{start}, goal state Q_{goal}, maximum iterations K, step size Δq

Initialize tree T with root node Q_{start}
for $k = 1$ to K **do**
  $Q_{random} \leftarrow Random\_State()$
  $Q_{near} \leftarrow nearest(T, Q_{rand})$
  $Q_{new} \leftarrow Extend(Q_{random}, Q_{near}, Stepsize)$
  if not $Collision(Q_{new}, Q_{near})$ **then**
    $T \leftarrow Q_{new}$
    $T \leftarrow addEdge(Q_{new}, Q_{near})$
    if $Q_{new} \in Q_{goal}$ **then**
      return the path from Q_{start} to Q_{new}
  else
    continue
return Failure

2) RRT:* Similar to RRt, however, the algorithm does not terminate when a valid path is found. Instead, rewires the original path to find more optimal solutions.

Algorithm 2 RRT*

Require: Start state Q_{start}, goal state Q_{goal}, maximum iterations K, step size Δq, neighbourhood size r

Initialize tree T with root node Q_{start}
for $k = 1$ to K **do**
  $Q_{random} \leftarrow Random\_State()$
  $Q_{near} \leftarrow nearest(T, Q_{rand})$
  $Q_{new} \leftarrow Extend(Q_{random}, Q_{near}, Stepsize)$
  if not $Collision(Q_{new}, Q_{near})$ **then**
    $neighbours \leftarrow Nearest(Q_{new}, T, r)$
    $Q_{min} \leftarrow ChooseParent(Q_{new}, neighbours, Q_{near}, T)$
    $T \leftarrow Q_{new}$
    $T \leftarrow addEdge(Q_{new}, Q_{min})$
    $T \leftarrow Rewire(Q_{new}, Q_{min}, Q_{nearest}, T)$
    if $Q_{new} \in Q_{goal}$ **then**
      return the path from Q_{start} to Q_{new}
  else
    continue
return failure

Algorithm 3 Informed RRT*

Require: Start state Q_{start}, goal state Q_{goal}, maximum iterations K, step size Δq, neighbourhood size r

Initialize tree T with root node Q_{start}
for $k = 1$ to K **do**
  if not $path$ **then**
    $Q_{random} \leftarrow Random\_State()$
    $Q_{near} \leftarrow nearest(T, Q_{rand})$
    $Q_{new} \leftarrow Extend(Q_{random}, Q_{near}, Stepsize)$
  else
    $Q_{random} \leftarrow Ellipsoid\_Sampler(path)$
    $Q_{near} \leftarrow nearest(T, Q_{rand})$
    $Q_{new} \leftarrow Extend(Q_{random}, Q_{near}, Stepsize)$
  if not $Collision(Q_{new}, Q_{near})$ **then**
    $neighbours \leftarrow Nearest(Q_{new}, T, r)$
    $Q_{min} \leftarrow ChooseParent(Q_{new}, neighbours, Q_{near}, T)$
    $T \leftarrow Q_{new}$
    $T \leftarrow addEdge(Q_{new}, Q_{min})$
    $T \leftarrow Rewire(Q_{new}, Q_{min}, Q_{nearest}, T)$
    if $Q_{new} \in Q_{goal}$ **then**
      return the path from Q_{start} to Q_{new}
  else
    continue
return failure

3) Informed RRT:* The sampler constructs an ellipse with focii as start and goal, and major axis as path length. The sampling space shrinks as more optimal paths are found, thus improving performance.

C. Constraint checkers

To ensure that each sampled point corresponds to a configuration that both maintains to RCM constraint and corresponds to a satisfactory pose of the robot arm, three key conditions are enforced. If a point fails to satisfy any one of these constraints, the point is discarded and a new point is sampled. The constraints are discussed as follows.

1) RCM Check: This condition checks whether the needle meets the RCM condition. This is done by stepping through finite distances on the needle and checking if any of the points lie inside the needle. This is done by checking the distance of the point from the RCM and flagging it if it within a threshold. he RCM check is shown in figure 3

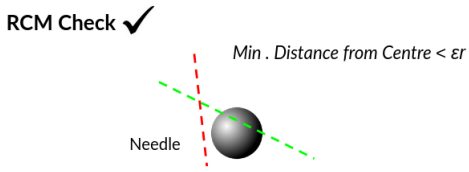


Fig. 3. Remote Centre of Motion Check

2) Orientation Check: This constraint checks that the needle points in the right direction, i.e, into the body of the patient. This is done by taking the inner product of the vector representing the needle's orientation with the RCM orientation. If the inner product is positive, then the orientation check is passed, this is illustrated by figure 4

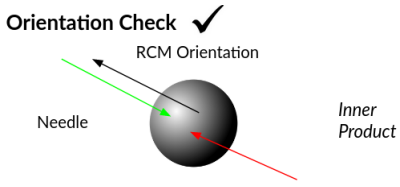


Fig. 4. Orientation Check

3) Safety Radius Check: The tip of needle shouldn't enter too deep into the body cavity, and hence a safety radius is defined. A point satisfies the safety radius condition if the tip of the needle lies inside the safety sphere and simultaneously the base of the needle lies outside. This prevents the patient from being impaled. A visual depiction is given by figure 5

4) Self Collision Check: The self collision check refers to the absence of collision between the links of the manipulator and the needle. While this isn't implicitly implemented in this project, it is assumed that it is satisfied by virtue of the other checks and the region of the task space that is being manipulated.

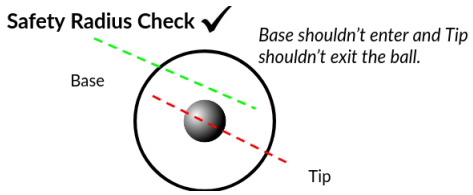


Fig. 5. Safety Radius Check

III. Results

Each of the algorithms presented have been implemented and simulated in gazebo. While all of them identify a feasible path in finite time, the time taken, the path length, the distance to goal and number of nodes expanded will all be different. This section examines how these quantities vary for the different algorithms, further given the probabilistic nature of these algorithms, the planning task is repeated multiple times.

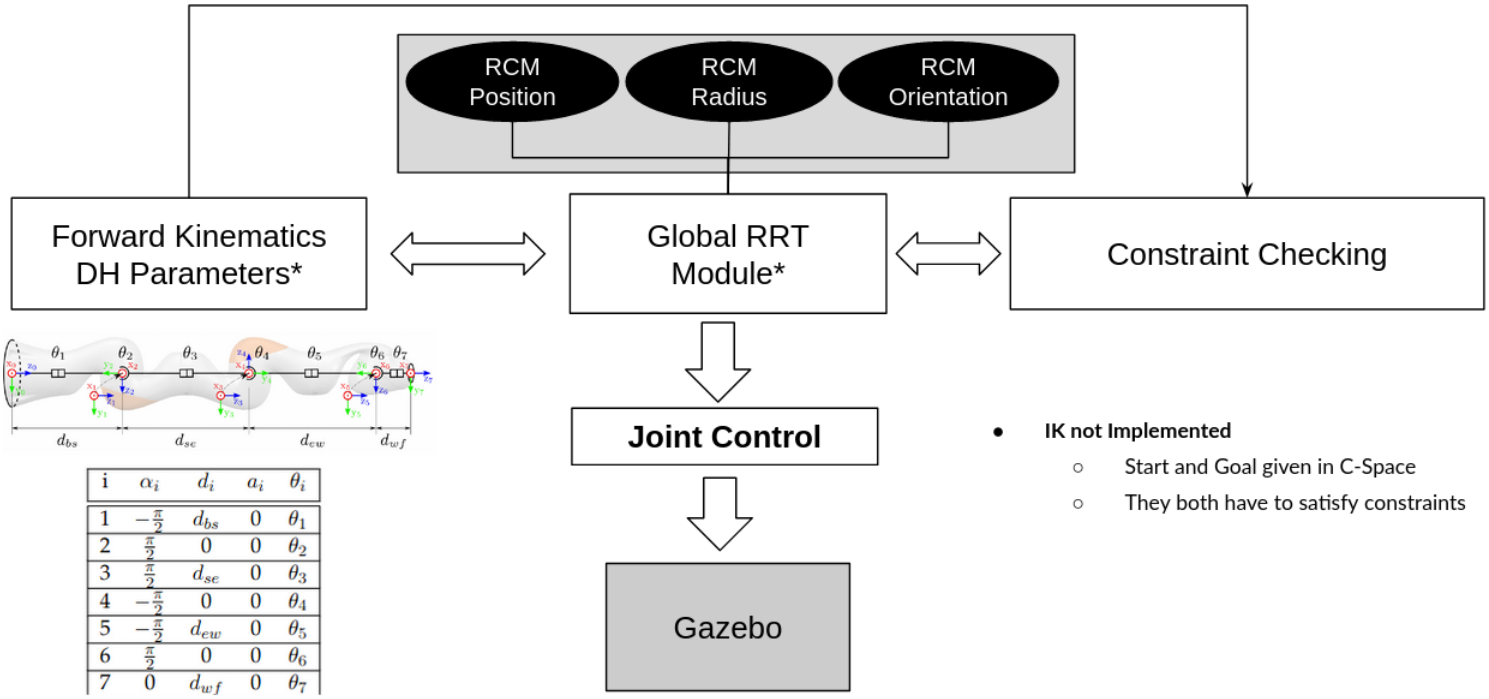
For all the planning problems used in this demonstration the following parameters are held constant:

- Number of Iterations : 300
- RCM Size : 10 mm (0.01 m)
- RCM Location : [0.695342 ,0.475406 ,0.477245]
- RCM Orientation : [-1,-1,1]
- Start Configuration : [26 ,27,22,-51,-23 ,62,-41]
- Goal Configuration : [44,23,15,-44,-37,80,0]
- Goal Radius (threshold) : 0.005

Apart from the start and goal configuration, which is given in degrees, all other parameters are in SI units. The start and goal have been identified using a random search of configuration that obey the constraints in the C-Space. The safety radius can be take to be 200 mm, however it is seen to not affect the planning and hence is disabled in interest of computational efficiency. The goal radius is the threshold value in the C-Space that defines whether the goal has been reached or not, this is checked using the L2 norm of the difference between final node in the search tree and the goal configuration.

A. Time Taken

The time taken by each algorithm to generate the final path is given by figure 7. The time taken for RRT is independent of the maximum umber of node-expansions/iterations, as the algorithm finishes running as soon as a path is identified. On the other hand the RRT Star and IRRT Star algorithms take longer due to fact that all the iterations have to be completed before a path is generated. It can be seen that on average RRT finds a fesible solution in less than 10s. This is with a goal threshold of 0.05 rad, which is relatively high. At lower goal thresholds, this number is anywhere between 10 s and 50 s. The other two algorithms take significantly more time due to the expensive re-wiring steps involved. While IRRT Star takes lesser time on average, it has a wider spread as seen from figure 7.



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

Fig. 6. Schematic of Subroutines and Modules in the Environment. The RRT module houses all the planner can be modified through a change of input parameters.

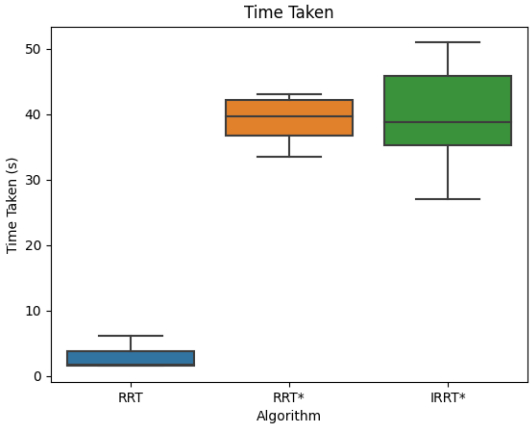


Fig. 7. Time taken to obtain final path in each algorithm.

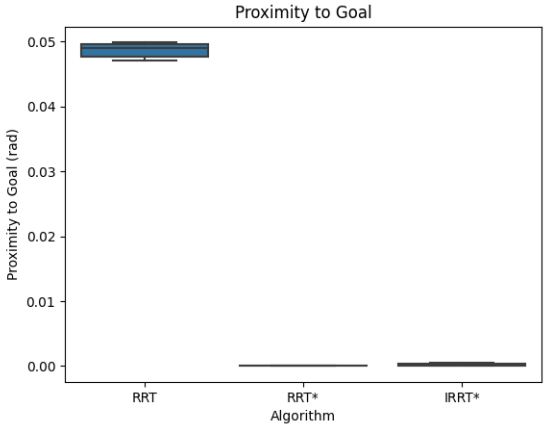


Fig. 8. Closest distance to goal for each algorithm.

B. Closest Distance to Goal

The proximity to the specified goal point is given by figure 8. The RRT algorithm generates a path as soon as the threshold is reached, hence the proximity to the goal is around 0.05. The RRT Star and IRRT Star both yield solutions very close to the goal point after 300 expansions. Their distance is of the order of 0.001 rad.

C. Number of nodes expanded

The number of nodes expanded is given by figure 9. In the case of RRT, the number of nodes expanded depends from case to case on the quality of exploration, in some cases it may get stuck, while in others it might be quite fast. The number of nodes expanded in the case of RRT Star and IRRT Star is constant and given by the maximum number of iterations.

D. Path Length

The path length or cost of the final path generated is given by figure 10. The RRT algorithm has the least optimal path length as it generates the fist feasible path it can identify. The RRT Star and IRRT Star algorithm on the other hand have more optimal paths. The RRT Star algorithm shows two

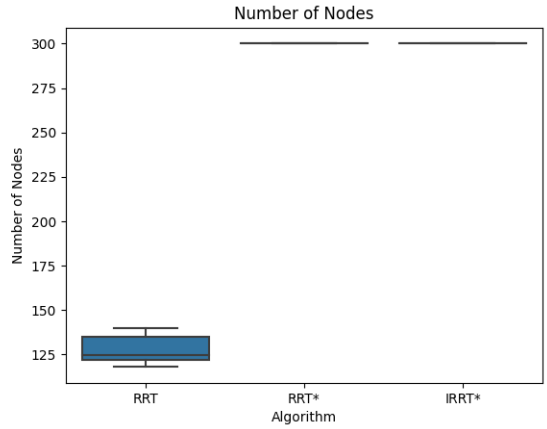


Fig. 9. Number of nodes expanded by each algorithm.

outliers, and has a higher average path length compared to the IRRT Star case. The IRRT Star case has a larger spread though compared, to the RRT Star case. The lower average path length can be explained by the more-effective ellisoid-sampling as shown in algorithm 3.

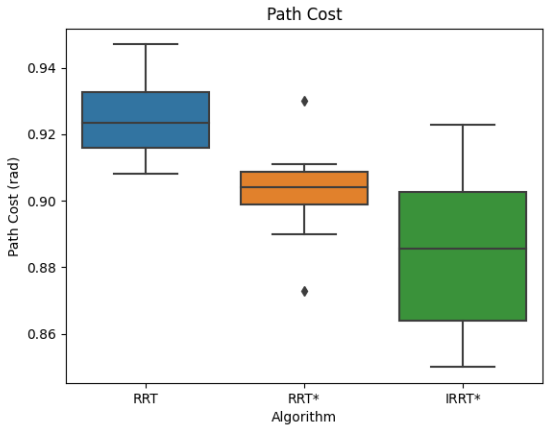


Fig. 10. Length of final path identified by each algorithm.

IV. Discussion

The algorithms and their performance has been evaluated both qualitatively and quantitatively. Given the probabilistic nature of the algorithm, the various metrics vary from trial to trial. It is apparent that, while the behavior of the algorithms differ once a feasible solution is found, they remain largely the same prior to that. The initial solution in a general case using RRT is takes anywhere between 5 and 15 seconds to find, depending on whether the algorithm gets stuck in un-related alleys, this is largely the same using RRT Star and IRRT Star as well.

From a qualitative point of view, it is seen that the vanilla RRT path while produced in a very short time, is quite random and un-directed with un-necessary motions. In comparison the RRT Star and IRRT Star path is much more intuitive and has a shorter path length that implicitly results in fewer way-points on the path. IRRT Star is much faster post the identification of the first feasible solution as the sampling is now focused in an area close to a feasible solution, thus the samples are more likely to satisfy the RCM constraints.

Quantitatively, its is seen RRT find a feasible solution in as few as 150 node expansions. The RRT Star and IRRT Star algorithm get closer significantly closer to the goal compared to the RRT algorithm. Finally, The path length is the lowest on average for the IRRT Star algorithm, followed by the RRT Star algorithm and then vanilla RRT.

V. Conclusion

The proposed methodology shows success in finding feasible paths to the goal, with each of the planning algorithms having advantages and disadvantages. However there is much scope for improvement in terms of both performance and capability.

The vanilla RRT algorithm finds a path in under 10s despite the skewed C-Space. However it is still not on par with other open source solvers, that can given solutions in under 1s. Therefore, there is scope to improve the efficiency by parallelizing the code, especially the computationally intensive collision checker that iterates over multiple locations on the needle.

Another limitation of the code is the fact that the start and goal points are currently specified in configuration space, while from a practical point of view, they need to specified in task space. This is because, the code doesn't incorporate inverse kinematics (IK) at any stage. A future iteration of the code-base would include IK that identifies goal and start configurations that don't violate the RCM constraint.

Additionally, it has been assumed that the robot start configuration is within the body cavity itself (and obeys the RCM constraint), this isn't case in practical application. Hence future work will include guiding the needle through the RCM and to the goal. This once again will require IK and hybrid planner that for both guidance and RCM maintenance.

Additionally, advanced road-map techniques such as HRM's as proposed in [6] have been used for lower-dimensional planning problems such as the motion of mobile robots through cluttered spaced. Future work could include extending the approach to lower dimensional manipulators in same vein. The authors have expressed reservations about the capability of the method for higher dimensional manipulators, and hence this remains an important juncture in motion planning research.

*1) **Video Demonstration**:* The video demonstration may be found in the following drive folder. The folder contains four videos, the first being a general environment setup video and the other three being simulations of the robot while using different planning algorithms.

*2) **GitHub Repository**:* The code used in this project is an adaptation of [5]. The modified code along with the custom scripts used in this project can be found in the following

repository. Basic setup instructions can be found in the video or in the read-me file of the repository.

REFERENCES

[1] S.-K. Kim, W.-H. Shin, S. Y. Ko, J. Kim, and D.-S. Kwon, "Design of a compact 5-dof surgical robot of a spherical mechanism: Cures," 08 2008, pp. 990 – 995.

[2] D. Berenson, S. S. Srinivasa, D. Ferguson, and J. J. Kuffner, "Manipulation planning on constraint manifolds," in *2009 IEEE International Conference on Robotics and Automation*, 2009, pp. 625–632.

[3] N. Aghakhani, M. Geravand, N. Shahriari, M. Vendittelli, and G. Oriolo, "Task control with remote center of motion constraint for minimally invasive robotic surgery," in *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 5807–5812.

[4] H. Su, Y. Hu, J. Li, J. Guo, Y. Liu, M. Li, A. Knoll, G. Ferrigno, and E. De Momi, "Improving motion planning for surgical robot with active constraints," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 3151–3156.

[5] C. Hennersperger, B. Fuerst, S. Virga, O. Zettinig, B. Frisch, T. Neff, and N. Navab, "Towards mri-based autonomous robotic us acquisitions: a first feasibility study," *IEEE transactions on medical imaging*, vol. 36, no. 2, pp. 538–548, 2017.

[6] S. Ruan, K. L. Poblete, H. Wu, Q. Ma, and G. S. Chirikjian, "Efficient path planning in narrow passages for robots with ellipsoidal components," *IEEE Transactions on Robotics*, vol. 39, no. 1, pp. 110–127, 2023.