Path planning to generate an optimal needle pose using the da Vinci Surgical Robot

1st John E. Draganov
M.Eng Robotics
University of Maryland,
College Park, USA
jdragano@umd.edu

2nd Jayesh Jayashankar
M.Eng Robotics
University of Maryland,
College Park, USA
jayeshj@umd.edu

***Abstract*—This paper presents a method to perform an efficient path planning operation to generate the optimal needle pose and performing a grasping operation with the da Vinci Surgical robot. The RRT* algorithm is used to implement the same by selecting a random set of joint parameters based on the joint limits to obtain the end effector positions using forward kinematics and then subsequently perform inverse kinematics to verify if the joint angles are valid or not real time. Collisions with other objects or with itself is continuously checked to ensure that the robotic arm takes an obstacle free path. The advantage of our method is that instead of planning the path using the end effector positions, usage of joint angles to perform motion planning results in more optimal trajectories to generate the final needle pose. The advantages of the RRT* over other planning algorithms is of its ability to rewire itself after finding the nearest node by checking the costs of its neighboring nodes. The algorithm is simulated on a da Vinci robot in gazebo and also verified with the MoveIt motion planning interface to prove the successful execution of the resultant optimal trajectories.**

Index Terms*—surgical robots, path planning, RRT, kinematics, RRT

I. Introduction

The growing presence of surgical robots are to overcome the limitations of minimally-invasive surgical procedures and to enhance the capabilities of surgeons. But most of these operations are teleoperated with the surgeon required to maneuver the robot arms to perform the surgery. Making these robots autonomous is a solution where the robotic arms find the best path to reach the destination in the patient's body, check for collisions by avoiding obstacles and for the arms to not self collide with each other. Suturing [1]is a crucial task in the surgical process which requires needle manipulation between the two arms and to optimally handoff the needle while maintaining the appropriate grasp. The global market for surgical robots is projected to be around USD 14.4 billion by 2026. The COVID-19 pandemic would also cause a lot of surgical procedures to be carried out using a robotic instrument to prevent the transmission of the disease. Some of the advantages of surgical robots include less blood loss, reduced recovery time and infection risk. In order to improve object-manipulation efficiency, dual-arm robots have been introduced into industrial environments.

To move the needle arms to different areas in a surgical process various motion planning algorithms have been explored over the years such as RRT [2], [3], Probabilistic Roadmap (PRM) [4], A* and DIjkstra to name a few. We felt the RRT* algorithm offered more advantages than the other methods due to the tree expansion process, the states expanding towards an emphasis on the goal by having a cost computation and also on its ability to rewire itself by comparing the cost of the obtained nearest node with that of its surrounding neighbors to obtain the best fit. This would be explored in depth in the upcoming sections.

This paper proposes our version of the RRT* algoithm with an emphasis on reaching the goals state by creating a collision free path using forward and inverse kinematics and selecting nodes not just by comparing the costs to come but by also considering the cost to go which results in more efficiency. The main objective is to generate an optimal needle pose and also to handoff the needle from the needle arm to the gripper arm. Previous work related to this topic are discussed in II. Secion III describes the RRT* algorithm in detail along with the pseudocode. Section IV goes into detail on the implementation of the algorithm and the different steps involved. The design of the custom CAD model of the da Vinci robot is described in IV-A. IV B defines the input that would be passed to the RRT* algorithm. IV-C talks about the forward kinematics performed to obtain the end effector position for a random set of joint angles. The validation of the obtained end effector position is verified through inverse kinematics in section IV-D. The implementation of our version of the RRT* and its advantages are described in IV-E. Section V describes the simulation results and the validation steps undertaken in the different parts of the pipeline.

II. Related Work

Path planning for dual arm manipulators have been explored using various implementations of Dijkstra's, A*, RRT and PSM. Not only should the shortest distance be taken to reach the goal state, care must be taken to detect and avoid collisions and the algorithm must be capable of deciding which arm should take the appropriate action such that the xcost is minimized for that particular operation. The computation of forward and inverse kinematics to determine the validity of the states is essential as well and needs to be done realtime.

Planning for pick and place operations using dual arm manipulators [5], [6] require a lot of considerations such as in [7] where the different poses, graps are pregenerated and stored in a database. The database is searched using the Dijkstra's algorithm where the database consists of the different poses, info about the obstacles present and info about the object used to pick and place to perform the motion planning. This method though significant requires the kinematics to be performed beforehand and not real time.

Probablisitc roadmaps (PRM) in [2], [8] is used to perform the grasping and regrasping operations by generating a "SuperGrasp" which merges the graphs of the right and left arms to form a collision free roadmap. This method though requires offline computation of the workspace and information of the environment where the motion planning is supposed to take place is required beforehand. The above methods fall short by not bring able to compute the workspace realtime. When the environment around which the robot operates is dynamic there could be major issues if the workspace is not computed realtime to avoid collisions. In critical processes such as suturing it is important for the robot to adapt to the changing scenarios.

The process of suturing [9] [10] requires the usage of an external needle which must be regularly passed between the two arms with/without changint he grasp transformation. The daVinci robot in [11] consists of two gripper arms and the work focuses on performing needle handoffs between the two arms using the RRT connect algorithm to generate a collision free path for an external needle to be transferred by a single arm in cartesian space and a handoff between the two arms but not moving the needle. The RRT tree is expanded by the usage of a heuristic distance function which is defined based on which the new states are obtained as the tree expands. This work also focuses on an optimal way in which the needle can be grasped based on which the handoffs are performed. In this method the location of the needle in Cartesian space is not known real-time but only the needle's geometric model and the grasp transformations are stored for the planning process.

Our project draws inspiration from the above but instead of the RRT connect algorithm the RRT* algorithm is used. RRT* differs from RRT because even after obtaining the nearest node from a random point. The algorithm finds all the neighbors of this interim nearest node and compares if the costs of any of the neighboring points to reach the random point is less than the cost associated with the interim nearest node to reach the random point. If a node is found satisfying this criteria, the nodes are rewired and the process continues. The RRT* implementation as part of this work has been enhanced in such a way that the nodes are continuously verified to see if they are closer to the goal by finding out the cost with respect to the goal. Thus increasing the efficiency in reaching the goal state and also to avoid expansion of the tree in different directions but with an emphasis on reaching the goal state. No prior data is stored as forward and inverse kinematics are performed to real-time to compute the end-effector workspace and subsequently verify the joint parameters and use this information to execute the motion planning.

III. Methodology

This section talks about the steps taken to implement the algorithm and the advantages of RRT* over the RRT sampling based planning method. Each step in the pipeline will be discussed in more depth in the next section. Firstly a CAD model of the daVinci robot is designed using Solidworks. The model is converted into a URDF format and imported to ROS, Gazebo and RVIZ to publish values to the different joints and perform the planning process. The code for the RRT* algorithm is implemented by considering the joint limits of the different joints of the two arms. A linearly uniform random set of joint angles are chosen for the different joints and forward kinematics are performed to find out the end effector position. The end effector position is in turn verified by computing the joint angles and checking if it lies within the joint limits and is within the reachable workspace. This makes sure that the point is a valid state and if such, the node is added to the tree. Once added the cost of its neighbors are computed and compared with the random point chosen and the node with the lowest cost from this set is chosen as the new node. The tree is then rewired. As new nodes are continuously added to the tree, the cost with respect to the goal is checked constantly to ensure that the tree is expanding towards the goal thus generating the optimal trajectory. The validation of this algorithm includes testing if the two arms can reach different goal states by avoiding collisions with the obstacles and with each other and also to perform a handoff operation where the arm containing the needle can be moved towards the gripper arm allowing the gripper arm to grasp the object. The advantages of RRT* over RRT are in its ability to rewire itself once the closest node is found and also on its capability to producing optimal trajectories in the form of straight paths. A notable disadvantage of RRT* though is in its performance which could take a long time to execute as rewiring the graph during each iteration can take a long time for the algorithm to execute.

IV. Algorithm Description

A. Design of robot model

The robot model was designed in Solidworks, using the readily available online diagrams of the daVinci robot. The system consists of three main bodies: the master (surgeon controlled monitor system), the active slave manipulator surgical arm which has seven degrees of freedom (DOF), and the imaging system. For the purpose of this proposal we focused on modeling the latter two. The standard actuated surgical arm consists of the following links: a shaft with 4 degrees of motion, with a wrist at the end which imposes a 2 degree of freedom constraint on the shaft (Figure 1). Our solidworks model closely followed this, with a gripper and needle joint each with four degree of freedom and two fixed joints (Figure 2).

The dimensions were loosely based on the height and size of the actual daVinci robot, with joints extended enough to

Fig. 1. Active arm joints

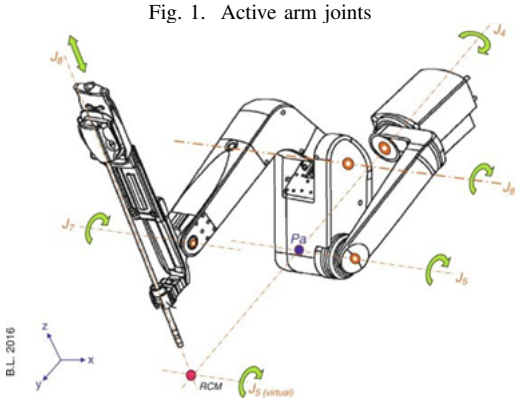
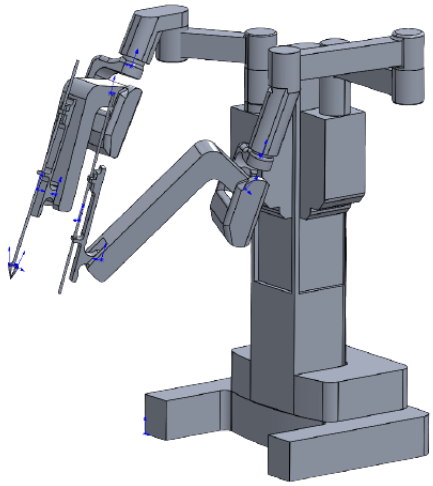


Fig. 2. Modeled daVinci system



allow for enough movement without exessive self collision. The default "home" position was chosen as see in Figure 2 and adjusted for before URDF export. Discrete cutouts were made in some areas in the model to allow for more movement in the arms and less collisions. The origins of each joint and link combo were also adjusted to facilitate the URDF export and allow for easier calculation of the Forward and Inverse Kinematics.

Slight adjustments were made to the abstracted sizes of the model arms in order to make up for the limitation imposed by only operating four of the arms. After the URDF was adjusted and exported, the model was loaded into Gazebo and the inertia matrixes were adjusted in order for the model to spawn correctly.

A joint controller was implemented by assigning all 9 joints as position controllers, and giving them nominal PID values (the latter of which were adjusted to allow for smoother motion which did not disturb the patient or base frame).The RQT Joint Controller node was invoked, with both arms as kinematic chains selected for its target. This allowed for easy initial testing and operation of the robot in order to validate with inverse kinematics coarsely (for example, sending the arms to very obvious angles such as straight up or parallel to the ground).

A speaker (made up of the 9 publishers of the joints) node was then written to send commands to the controllers in an array of the 9 joints, running through the list of nine publisher one by one, based on the number of inverse kinematic angles passed to the speaker as seen in figure 10. Values from the inverse kinematic solver were passed in a list to the speaker, allowing for them to be published one by one to all of the joints, allowing for instant control of the arms after receiving joint values/angles. Another initial for loop was initially implemented in order to calibrate the robot movement by giving it some initial values to go to, allowing for more precise joint movement within the non-collision causing inverse kinematic solutions.

B. Forward Kinematics

Due to the natural complexity of the daVinci model, inverse and forward kinematics functions were imported based on previously created matlab scripts, and ported into callable functions within the script. In order to use Denavit-Hartenburg (DH) Parameters, needed multiple dummy frames were used to ensure that the zn-1 and xn axes intersected and were perpendicular (Figure 3).

Fig. 3. DH Parameters

Frames	θ	d	a	α
0→1	θ_1^{**}	$-l_1$	0	90
1→2	-90	l_2	0	180
2→3	θ_2^{**}	0	l4	90
3→4	0	l_3	0	90
4→5	θ_3^{**}	0	l_5	-90
5→6	0	d^{**}	0	0

C. Inverse Kinematics

Calculation and formulation of the inverse kinematics was done in two parts (inverse kinematics with a joint constraint of the gripper and work-space variables). The fourth degree of freedom is the direction that the end effector faces constraining the gripper to always face straight down.

Fig. 4. IK Equations

$$\theta_1^{**} = \psi - \phi = cos^{-1}\left(\frac{l_2}{\sqrt{x^2+y^2}}\right) - tan^{-1}\left(\frac{x}{y}\right)$$

$$\theta_2^{**} = tan^{-1}\left(\frac{l_4}{l_3}\right) - cos^{-1}\left(\frac{r-l_5}{\sqrt{l_3^2+l_4^2}}\right)$$

$$\theta_3^{**} = \theta_2^{**} - \frac{\pi}{2}$$

$$d^{**} = z - l_1 - \sqrt{l_3^2 - l_4^2} * sin\left(cos^{-1}\left(\frac{r-l_5}{\sqrt{l_3^2+l_4^2}}\right)\right)$$

D. Input parameters

The function call of our path planner takes the joint limits, start and goal (in terms of joint parameters), the maximum number of iterations to traverse and the discretization of your workspace. Defining the area traversed by the planner allows for coarse control of the joint limits (i.e. defining the maximum and minimum value that can be randomly chosen for of each joint).

An initial pose is given to the needle, commanding all of its joints to it's "initial" start point [0.617, -0.350, -0.0225, -0.150] corresponding to end effector coordinates of (1.20, -1.11, 0.71) in XYZ space. Initial pose of the gripper was given as [0, 0, 0, 0], with a goal pose of [0.92, -0.70, -0.42, -0.25, 0.30]. A notable factor that influences overall discretization of the joint space were the smallest change necessary in the joint parameters i.e. starting at 0.01 and having to move to 0.1 would result in only one intermediate step if the space is broken up into variable between 0.01 and 0.1. This is especially noticeable if another joint parameter needs to shift from a very small initial start to a much large goal (0.01 to say 1 in this context), resulting in large "intermediate" steps for that particular joint.

E. Implementation of RRT*

Below provided is a snippet of pseudo-code used in our path planner

Algorithm 1 RRT*

```
0: NodeList ← Start
0: for iteration = 1, ..., N do
0:    x_rand ← sample(min, max)
0:    FWDKin ← x_rand
0:    IK ← FWDKin
0:    if collision = False then
0:       while IK = Invalid dox_rand
0:          X_nearest ← nearest
0:          X_new ← Steer(Near, New)
0:       if collision = False then
0:          x_new ← x_parents
0:          for iteration = 1, ..., lenParents do
0:             SmallestCost ← x_nearnodes
0:             Rewire ← steer(x_new, NearInd)
0:          if MaxIter = False then
0:             LastIndex ← cost  =0
```

After initial parameters are passed to the RRT state solver, a random point is selected based on the minumum and maximum joint parameters, the discretization of the space, as well as the goal and start states. Joint parameters once calculated are then passed to the Forward Kinemtatics solver to find their end effector position, and then to the Inverse Kinematics Solver respectively to determine legality of the points.

Once this random point is chosen, the distance between the node and it's parent vertex is calculated, used as a cost. After a close node is found, a series of it's neighbour is examined, where the chosen neighbours are taken based on two conditions of validity: IK validity and collision interference.

A "virtual" obstacle is defined within the context of the traversible workspace in the code. Actual obstacle placement within gazebo was avoided due to encountered errors. This is mainly due to our insertion of viable obstacles being one's that float in space. Disabling gravity for those obstacles caused unexpected errors in the gazebo environment. Instead, a paramterized equation is provided for the spherical obstacle. Each chosen point is checked against state validity by finding it's distance to the center of the sphere and returning "False" for a collision if this distance is less than or equal to the radius of the obstacle plus some "needle" factor.

Once the distances of these neighbours is also found, if a closer node (cheaper cost) is found, it is chosen as the next node to be traversed. After it is inserted into the list of tree variables, the neighbours are checked again, this time to see if them being rewired will lower their cost, and if so they are rewired to the latest vertex, allowing for path smoothing.

Upon completion, the path is examined, and the traversed shortest path is returned as an array of consecutive four joint variables. This array is given state by state to the Joint Publisher, with sleep commands in between the state changes to allow for the joints to settle (the base is unfixed due to errors encountered with the model when fixing the base, and large jerky motions tend to move the entire model). The gripper arm is opened at the start of this execution, and closed once it completes, ideally over the needle simulating a handoff. After execution, visual verification is performed to see how close the gripper is to the needle. Additional verification of the path traversed can be seen by mapping the path and comparing it to the generated MoveIt Path (more in the Results and Discussion section).

V. Results and Discussion

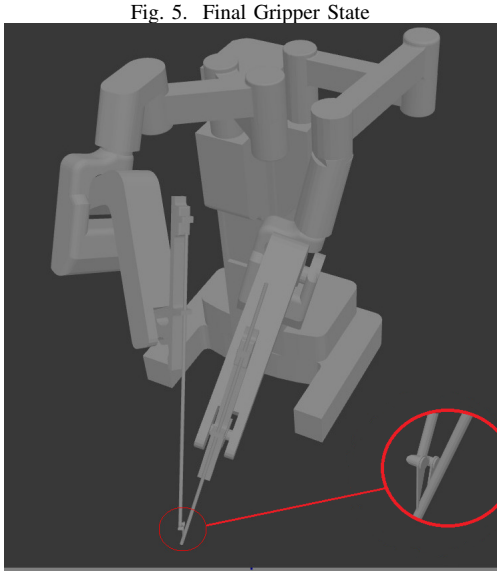
Upon successful generation of traversable nodes, comparisons were made to paths generated in the MoveIt environment. Successful generation was made based on visual proximity to the goal, as well as a function used to determine proximity to the goal state similar to that implemented in Project 3 Phase 2.

Optimization can be performed on improving the inverse kinematics model to determine better joint states; current inverse kinematics limits a 90 degree offset between joint 2 and 3, instead of potentially using a combined model for the overall added angle between both.

The output states were fed to the Publlisher as described above, producing a successful path with obstacle collision implemented; path discretization varied depending on input variables. Final position achieved displayed below in Figure 5.

The Moveit configuration was displayed with different obstacles implemented than the Gazebo configuration. Figure 6 shows the traces of the path traveresed, giving some sense of the optimal path comparison between both. Overall, smoothness of the movement states with the RRT* path planner can

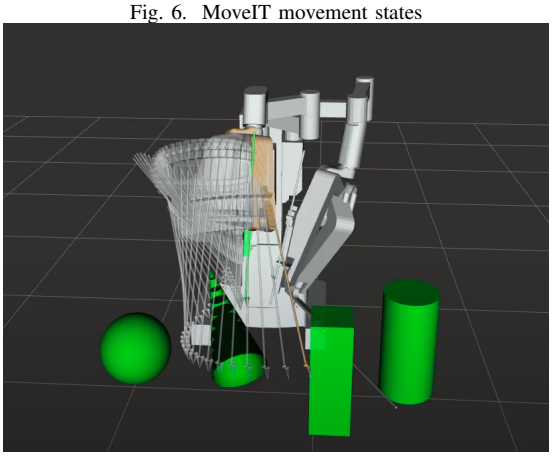
Fig. 5. Final Gripper State



be improved, as well as discretization of each of the joint parameters (as dsicussed, smaller joint goal states tend to throw the results off).

Run time for the RRT* planner was consistently under 4 seconds, largely in part to the larger discritzation as well as the larger joint limits used in the MoveIt model; the initial state of the gripper arm was set to be much closer in the Gazebo/RRT* planner versus the further origin state of the MoveIT model.

Fig. 6. MoveIT movement states



Joint parameters needed to be defined as such that all goal joint variables are positive; the cost function operates only in positive increments, where feeding it any negative goal states will cause it to increment until the first found positive joint variable.

VI. Conclusion

In this paper we have presented how the RRT* algorithm produces optimal trajectories to generate the pose of the end effector of the arms of the da Vinci Robot and also perform a handoff operation from the needle arm to the gripper arm. The motion planning was successfully executed in a custom gazebo world with objects set as obstacles and performing the mentioned actions by computing the forward and inverse kinematics to check if a point is in a workspace and to detect collisions. Future work would include the usage of a vision system which would greatly assist in obstacle detection and also when performing handoff operations between the two arms. The model of the da Vinci robot can be further enhanced by giving each arm an additional 2 degrees of freedom since two joints of each arm are fixed.

References

[1] J. M. E. D. S. A. Pedram, P. Ferguson and J. Rosen, "Autonomous suturing via surgical robot: An algorithm for optimal selection of needle diameter, shape, and path," 2017 IEEE International Conference on Robotics and Automation (ICRA), 2017, pp. 2391-2398, doi: 10.1109/ICRA.2017.7989278.

[2] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[3] C. L. M. K. S. K. J. Park, W. J. Park and H. J. Kim, "Endoscopic camera manipulation planning of a surgical robot using rapidly-exploring random tree algorithm," 2015 15th International Conference on Control, Automation and Systems (ICCAS), 2015, pp. 1516-1519, doi: 10.1109/ICCAS.2015.7364594.

[4] S. L. E. A. C. M. A. Baumann, D. C. Dupuis and J. J. Little, "Occlusion-free path planning with a probabilistic roadmap," 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008, pp. 2151-2156, doi: 10.1109/IROS.2008.4651035.

[5] N. S. et al., "Global/local motion planning based on dynamic trajectory reconfiguration and dynamical systems for autonomous surgical robots," 2020 IEEE International Conference on Robotics and Automation (ICRA), Paris, France, 2020, pp. 8483-8489, doi: 10.1109/ICRA40945.2020.9197525.

[6] J. C. D. S. J. Saut, M. Gharbi and T. Siméon, "Planning pick-and-place tasks with two-hand regrasping," 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2010, pp. 4528-4533, doi: 10.1109/IROS.2010.5649021.

[7] A. S. C. R. H. Marino, M. Ferrati and M. Gabiccini, "On the problem of moving objects with autonomous robots: A unifying high-level planning approach," in IEEE Robotics and Automation Letters, vol. 1, no. 1, pp. 469-476, Jan. 2016, doi: 10.1109/LRA.2016.2519149.

[8] V. G. P. Sudhakara and K. Sundaran, "Trajectory planning using enhanced probabilistic roadmaps for pliable needle robotic surgery," 22018 International Conference on Recent Trends in Electrical, Control and Communication (RTECC), 2018, pp. 61-64, doi: 10.1109/RTECC.2018.8625678.

[9] T. Liu and M. C. Cavusoglu, "Needle grasp and entry port selection for automatic execution of suturing tasks in robotic minimally invasive surgery," in IEEE Transactions on Automation Science and Engineering, vol. 13, no. 2, pp. 552-563, April 2016, doi: 10.1109/TASE.2016.2515161.

[10] R. C. Jackson and M. C. Çavuşoğlu, "Needle path planning for autonomous robotic surgical suturing," 2013 IEEE International Conference on Robotics and Automation, 2013, pp. 1669-1675, doi: 10.1109/ICRA.2013.6630794.

[11] T. S. S. Lu and M. C. Çavuşoğlu, "Dual-arm needle manipulation with the da vinci®surgical robot," 2020 International Symposium on Medical Robotics (ISMR), 2020, pp. 43-49, doi: 10.1109/ISMR48331.2020.9312930.