An RRT* Path Planning Implementation for a 4-DOF Laparoscope with 3D Visualization

Kaan Akduman, Noelle Nelson, Tyler Petrie

Electrical, Computer, and Systems Engineering 499

Project 2 Report

***Abstract*— This report describes the process and results of creating a motion planning algorithm for a 4 DOF laparoscope as well as a program in Unity to interactively view the results. This was accomplished by using an RRT* motion planning algorithm implemented in Python.**

I. Introduction

Laparoscopic procedures are commonly used as a minimally invasive way to operate on a patient. This technique allows the surgeon to only make a few small incisions, inserting a camera and other instruments through the small incisions. While this makes the recovery much easier for the patient, the surgeon has a much poorer visualization of the operating environment. It can be hard to go from looking at the 2D visualization of the camera view from the laparoscope to working in the 3D space of the surgery.

Our project's goal was to create a program in python that utilizes a path planning algorithm to guide a 4 degree of freedom (4-DOF) laparoscope from its initial position to a point where it could view the target, additionally we replicated the robot's motion, the path, and the target in Unity. A user can interact with the system using two joysticks to follow that path using that camera. The path that we found is an optimized and smooth path so that the user can follow it to the goal efficiently and with minimal risk of disrupting other organs.

II. Methods

In order to accomplish our goal, we set up the environment in Python and used Denavit-Hartenberg parameters to place the scope in world space. We then used an RRT* path planning algorithm in order to find an optimized path which we then smoothed the path using a moving average filter. Finally, we replicated the system in Unity to provide better visualization, as well as an interactive system.

A. Environment

In order to create a simple version of the surgical environment, we created small boxes, with a point on the inside that was our target. Realistically, the robot merely needs to be able to see inside of the box. However, in order to implement the motion planning algorithm, it was useful to have an exact goal point that the camera simply needed to get close to.

In order to accomplish this, we created a cloud of points that made up the box and the goal. The user can then input angles of rotation and distances of translation in order to move the box anywhere within the workspace. This box serves as both the target and the obstacle as the goal is to be able to look inside the box and see the target without hitting the box.

B. Model Kinematics

For this project, we based our robot model on a 4-DOF laparoscopic surgery robot developed by Alassi et. al [1] and shown in Figure 1. This robot can account for four parameters, the pitch and yaw of the insertion tool, the depth of the tool, and the roll of the tool which drives the orientation of the camera. This can be represented as an RRPR robot, as shown in Figure 2. For our implementation, we are placing a camera at the end effector, rather than a surgical tool. The camera was mounted on the end of the probe at a 45-degree angle.

Working from this model, we built a Denavit-Hartenberg table to construct our rotation matrices to build a motion model for our system. This model takes four parameters: *q1, q2, q3*, and

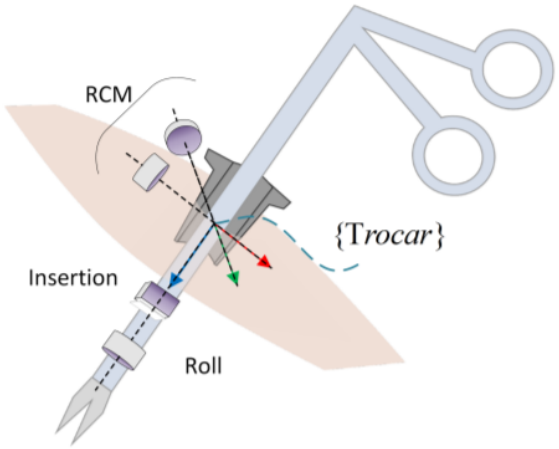

Figure 1: Visualization of laparoscopic robot with revolute and prismatic joints superimposed [1]

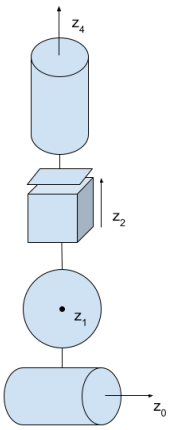

Figure 2: RRPR configuration of a manipulatable laparoscopic camera

q4. These parameters are the control commands for the pitch, yaw, depth, and roll, respectively. We used the Denavit-Hartenberg parameters to construct a rotation matrix that contains the orientation and displacement of each robotic joint relative to the previous one. By multiplying these matrices together, we can determine the orientation and location of the end point, or the camera, which is used to calculate the distance to the target in our RRT* algorithm.

Table 1: Denavit Hartenberg Parameters for a 4-DOF Laparoscope Robot

Joint	***r***	***d***	$\propto$	Θ
0	0	0	-pi/2	q1
1	0	0	-pi/2	q2
2	5 + q3	0	-pi/2	0
3	0	0	0	q4

*C. RRT * Implementation*

We decided to use Python for our algorithm. In our RRT* algorithm, a random configuration is chosen. We refer to this configuration as qrand. We then loop through our list of visited nodes looking for the node that is closest to random configuration in worldspace, which we call qnearest. We then compute a step toward qrand from qnearest. To compute this step node, qstep, we set qstep to qrand, calculate qstep's worldspace, and then until qstep is within a specified step distance of qnearest, update qstep's configuration space to be halfway between its current configuration space and qnearest's configuration space. Then, if qstep does not collide with any of our obstacles, we connect it to the graph and recursively update both the cost and the parent of all of its neighbors. When a node's worldspace is within a given range of the goal, we consider the goal found. However, we can and do continue to iterate through this process to find a more optimal path to the goal. If you wish to learn more about our algorithm, you can view the source code on GitHub.

D. Smoothing

The purpose of smoothing the function is to create a clearer path for the surgeon to follow. The output path of the RRT* algorithm can often be choppy, with sharp corners, the smoothing algorithm simply smooths the path out in order to create a cleaner path.

Originally, the plan was to use a Kalman Smoothing Algorithm, such as the Rauch-Tung-Striebel algorithm [2], however, the Kalman Filter approach required measurement data which we did not have, therefore, we opted to use a simpler moving average filter, as it was much easier to implement and was effective at smoothing out the path. The moving average filter simply takes the average of the previous point, the current point, and the future point, in order to calculate a new path.

E. 3D Visualization

The goal for this project is to be able to visualize the path and control the movement of the camera in 3D. Using Unity, a game development engine, we built a 3D cylindrical representation of our

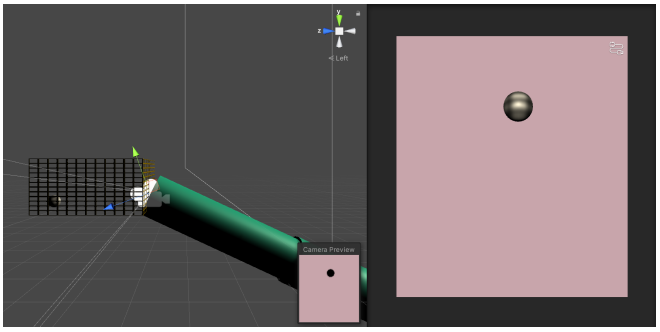

Figure 3: Side-by-side third-person and first-person view in Unity. The left side shows the laparoscope reaching the box, but not passing through it, to look at the sphere. The right side shows the first-person camera view, a sphere in a pink fleshy environment.

laparoscopic camera. This 3D model is composed of three cylinders put together, each controlling a different motion of the camera located coincident to the end of the robot and oriented at a 45° angle around its y-axis. This model, as well as a sample path, can be seen in Figure 4.

To control the robot, the user can operate two 2-DOF joysticks shown in Figure 5. The direction of these joysticks is read by an Arduino Uno and passed to Unity to handle the motion of the robot. These values are multiplied by a scaling factor set by the user to increase or decrease the speed of rotation and translation, and multiple scaling factors can be used for different motions.

Our optimal path calculated by RRT* can be plotted by hand in Unity for the user to follow with their camera. When using this program to train, the user can use the first-person camera view or a third-person view where they can see the whole environment. A side-by-side comparison of these points of view is shown in Figure 3. While a first-person view may be more akin to a real-world scenario, the third person view may be useful for training and familiarizing the user with controls.

III. Discussion & Limitations

This project was overall successful in accomplishing the goals that were originally set for ourselves. There are still things that could be improved upon before implementing this program as a tool.

A. Discussion of Results

Overall our algorithm was successful in reaching the goal and avoiding obstacles. We originally ran our algorithm with just one box and goal. The result of this can be seen in Figure 6. The blue path is the

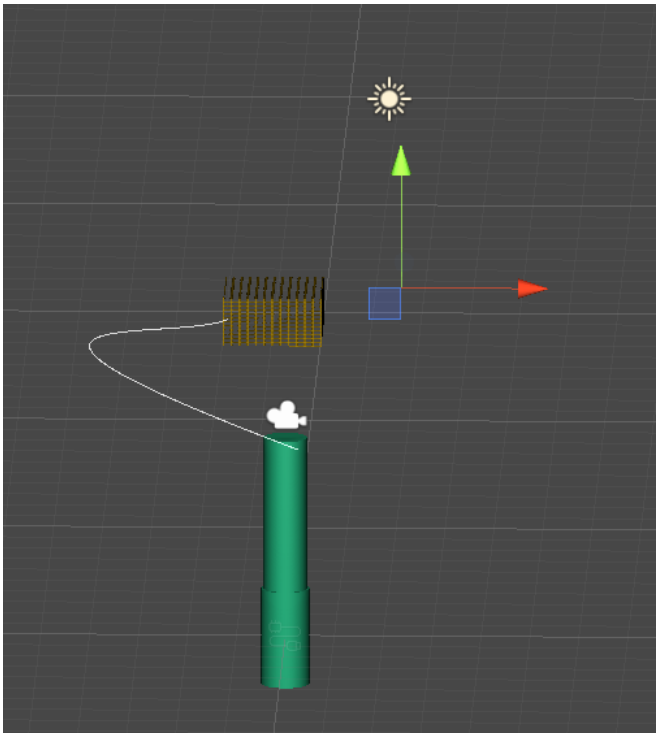

Figure 4: Screenshot of third-person view of 3D rendering in unity. The laparoscope is shown in teal, with a sample path shown in white. The grid box is the target region, with a sphere inside as the target for the camera to look at.

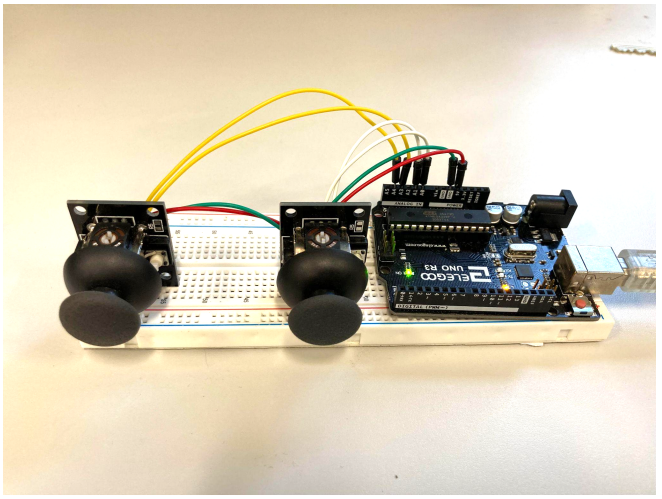

Figure 5: Image of the two joysticks connected to the Arduino. Horizontal movement of the left joystick controls pitch while vertical movement controls yaw. Horizontal movement of the right joystick controls roll while vertical movement controls depth.

original path from the RRT* algorithm and the orange path is the path after smoothing. We then added a second obstacle that was in the way of the original path to see if the algorithm could adjust and

avoid the obstacle. The results of this can be seen in Figure 7. When the two figures are compared it can clearly be seen that the algorithm adjusts the path to avoid the obstacle.

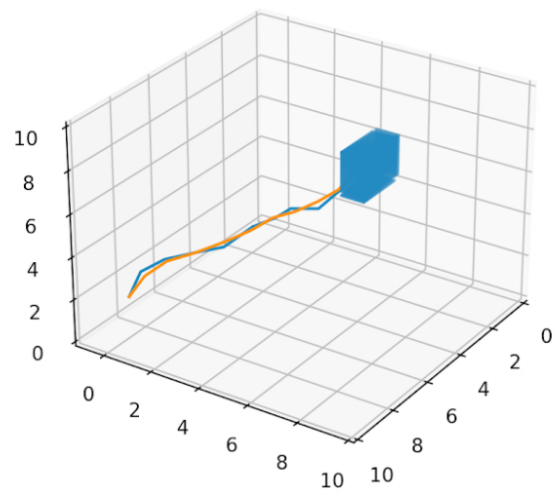

Figure 6: Results of the RRT algorithm with one box and goal.*

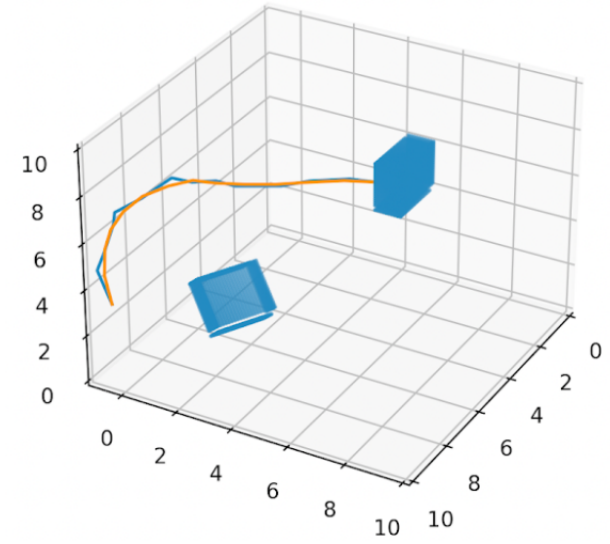

Figure 7: Results of the RRT algorithm after adding a second obstacle. The goal is the same as in the figure above.*

B. RRT* Computational Requirements

RRT* differs from RRT in a few ways. In RRT, when we calculate the position of qstep, we would connect qstep to the map through its closest visited configuration. In RRT* however, we connect it to the neighbor that gives it the smallest cost to reach that node. Additionally, after connecting a new node, we must recalculate the costs of nodes that are near the new node. If any of the nearby nodes can be reached through the new node to give it a lesser cost, we reconnect those nodes through the new node. After updating any nodes this way, we must also recursively update their neighbors to reflect the current node's new cost. [3] This recursive processing makes RRT* take much longer than RRT. Consider the scenario where we have thousands of nodes in a graph. If our next random node is near the initial configuration and we find that one of our visited configurations has a better cost when connected to our new node, that node will update which will recursively update all of its children nodes. Therefore, as the number of iterations of the loop increases, the computational requirements increase exponentially.

However, even with only a few iterations, RRT* is slow. One of the slowest parts of the algorithm apart from recursively updating neighbor nodes is collision checking. Collision checking slowed things down because we wanted to check to make sure that the camera's vision was unobstructed. We did this by iterating through the line segment between the camera and its viewpoint and checking if that area was obstructed in each iteration.

Despite these disadvantages, RRT* can sometimes be preferred over RRT because it can, given infinite time, find an optimal path between two points. And even with only a finite amount of time, it will continuously update and optimize the cost to reach the goal.

C. Range of Motion

There are some limitations of where the Laparoscope can reach. Since a laparoscope has a minimum and maximum depth that it can reach and its yaw and pitch can only rotate so far, there lies only a limited area through which objects can be seen by the laparoscope.

This made choosing targets somewhat difficult as our targets were made using a rotation translation matrix. We sometimes generated targets that could not be reached due to the limited range of motion of the scope. In the future, it would be useful to have a method to ensure that the euclidean position of the goal is within reach of the scope.

D. Realism

One shortcoming of this project is that it is not the most realistic representation of what laparoscopic surgery is actually like. Our version of the environment is a greatly simplified version of reality.

In our model, the obstacles are very clearly defined, and everything is static and stable.

In a real operating situation, the goal and the obstacle will not be able to be as clearly defined. Additionally, there are other complicating factors such as potential movement of the obstacles, organs, and even the point of entry.

Additionally, the Unity visualization clearly looks much different than the view from an actual laparoscope camera would. The physical controls are different, and the velocity was chosen by the user and may not reflect actual laparoscopes. Other considerations with Unity include that the rendering was spotty at times, and the box would disappear if the user got too close. Additionally, the 3D environment was incredibly simplified and would need to be scaled and shaped to better resemble a human body.

E. Offline Path Planning

Another shortcoming of this algorithm is that all of the processing is done offline. The obstacles and goals need to be fully known before the algorithm can be run. This would make it difficult to apply to real-life situations where there may need to be measurements or imaging to fully characterize the environment. Additionally, the path needs to be fully computed before it can be displayed and if there are any small adjustments that need to be made, the process would have to start again from the beginning.

IV. Conclusions & Future Work

Overall this project was successful as a proof of concept for our algorithm, but there are still some significant barriers to real-life implementations.

Future work may include being able to shift the web of nodes in order to account for any movement that may occur either at the point of entry of the probe or with the obstacles themselves. Another interesting idea to pursue would be to test the algorithm with multiple goals to see if it could go from one goal to another while adding on to the same web of nodes. This would add another layer of complexity and usefulness to the program.

References

[1] A. Alassi, N. Yilmaz, M. Bazman, B. Gur and U. Tumerdem, "Development and Kinematic Analysis of a Redundant, Modular and Backdrivable Laparoscopic Surgery Robot," *2018 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, 2018, pp. 213-219, doi: 10.1109/AIM.2018.8452712.

[2] Jeffrey W. Miller. “Lecture Notes on Advanced Stochastic Modeling”. *Duke University, Durham, NC*. This work is licensed under a Creative Commons BY-NC-ND 4.0 International License. 2016.

[3] T. Chin, “Robotic Path Planning: RRT and RRT*,” *Medium*, 26-Feb-2019. [Online]. Available: https://theclassytim.medium.com/robotic-path-planning-rrt-and-rrt-212319121378.