Robot System Programming Final Project
Path Planning and Visualization of Flexible Needle

Yanzhou Wang

May 14, 2020

1 Introduction

In robot-assisted minimally invasive surgery, it is beneficial to be able to extend the current capability of *ROS*, *MoveIt!*, and *RViz* to perform nonholonomic path planning for flexible needles that can be attached to the end-effector of a manipulator arm; however, a few challenges needs to be overcome:

- Representing a continuous flexible needle with finite number of jointed rigid segments
- Nonholonomic path planning for a combined system consisting of a manipulator arm and a flexible needle
- Visualizing a returned trajectory for any part of the combined system

Because the *ROS* ecosystem is not built for soft-body simulation, one has to resolve a continuous soft body into jointed rigid components that are connected by elementary joint types. For a flexible needle, the segments are approximated by short cylinders and connected by revolute joints. However, this presents a big challenge for path planning, because the planning is done in the joint space, and the more cylinders employed to approximate the needle, the greater the joint space will be.

Although there exists published models for simulating the body velocity of the needle tip using a unicycle model, resolving this body velocity to joint velocity is not a trivial task, given the configuration of the needle tip is dependent on the manipulator arm as well as each individual segment of the needle. In addition, if the end user decides to increase/decrease the number of segments used to approximate the needle, the mapping from body velocity to joint velocity needs to change accordingly. At the time of writing, the author is not aware of any libraries that perform such kind of task, and therefore a planning algorithm is not actually implemented in the current project—a planning framework is simply presented for extension of the project in the future.

In terms of visualization, *RViz* provides an impressive collection of plugins to deal with various types of data, yet it currently does not support trajectory visualization based on planned

path returned by *MoveIt!*. Noticeably, although *Move Group C++ Interface* has the capability of capturing the returned plan and republish the end-effector positions to a MarkerArray for *RViz*, it is not desirable to hard-code the planning group and end-effector in a node; rather, a separated plugin should be dedicated to this task.

2 Software

This section introduces the packages that are created as part of this project. All packages are extended from the default *MoveIt!* and *RViz* API's, and no additional external libraries are used.

display_trajectory_msgs: A Stamped extension of moveit_msgs/DisplayTrajectory. This extension allows RViz plugins to built in order to work on the returned trajectory via ROS topics. The result is a simple message type display_trajectory_msgs/DisplayTrajectoryStamped.

display_trajectory_msg_converter: A single node that subscribes to topic under the message type moveit_msgs/DisplayTrajectory, and reworks it into display_trajectory_msgs/DisplayTrajectoryStamped. The default outlet is /trajectory_stamped, which can be used in the traj_visualization_plugin. This node is included in a demo.launch file and remapped to subscribe to the correct topic.

traj_visualization_plugin: An *RViz* plugin that allows visualization of a planned trajectory by *MoveIt!*. The topic should take in messages of type display_trajectory_msgs/DisplayTrajectoryStamped. The Target link field should be filled with a desired link in the robot kinematic chain. Note that in order for the plugin to behave correctly, the entire kinematic chain should be set up as a planning group. For example, in case of a UR5 robot with a needle attached, the UR5+needle should be bundled together as one planning group. The reason being that the trajectory returned does not have any information about the planning group; if the UR5 and needle are two planning groups, there is no way to tell (at least from a publisher/subscriber perspective) which planning group this returned trajectory is for. A correct example is set up in the companion package ur5_needle_moveit_config.

needle_planner_manager: A planner manager that loads a custom RRT planner prototpye for needle steering. The planner manager exposes the parameters that are relavent to non-holonomic planning for needle steering using RRT. The parameter values can be changed in *MoveIt!* GUI. Additional parameter values can be added and default values can be changed in ur5_needle_moveit_config/config/needle_planning.yaml.

needle_description: Contains the URDF of a needle model. Adjustable parameters include: needle length, needle diamter, approximated maximum curvature, and number of needle segments in the mesh. The reason why the maximum curvature is adjustable is because this curvature is used to generate joint limits for each needle segment, but it does that at the cost of fidality of the linear approximation of the curved needle, because ideally the approximated line segment depends on the curvature itself, which is variable in real world. Adjust the approximated value to suit your need. The needle can be attached to any other robot. See an example in package ur5_needle_description.

ur5_needle_description: Contains the URDF of a plain UR5 with a needle attached at its tool0 link.

ur5_needle_moveit_config: Companion *MoveIt!* package for ur5_needle_description, traj_visualization_plugin, needle_planner_manager, display_trajectory_msgs and display_trajectory_msgs_converter. The demo.launch file is modified such that display_trajectory_msg_converter node is launched and remapped to the correct topic, and the default planning library is set to use the custom library. The config file is also modified to look for traj_visualization_plugin upon launch.

3 Workflow

Because currently no working RRT algorithm for needle steering is implemented, the two parts of this project are currently disjointed for the most part.

To use the custom planner manager, go to ur5_needle_moveit_config/launch/move_group.launch and change the "planning_pipeline.launch.xml" argument value from "ompl" to "needle". This will launch the custom planner manager and the planning algorithm.

Upon launching demo.launch, the "Planning Library" will be "Needle Planning", and the "RRTNeedlePlanner" can be selected from the drop-down menu. The planner parameters are displayed on the right-hand side and the parameter values can be changed to suit one's need (see Figure 1). *rosparam* and *rosservice* calls can be made to retrieve and validate the updated parameter values.

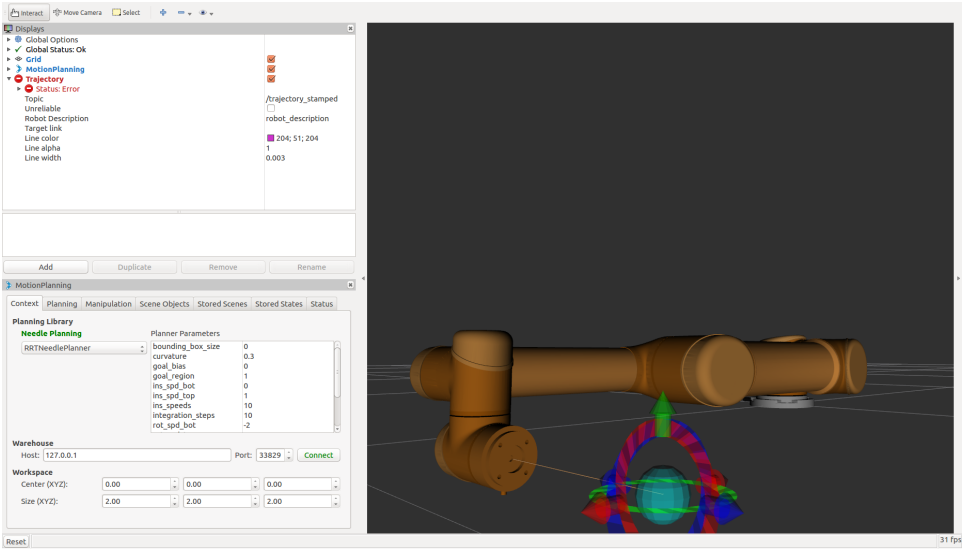

Figure 1: Custom planner manager with custom RRT planner loaded.

To see the traj_visualization_plugin, change the argument back to "ompl" and launch. The "Trajectory" plugin should show an "Error" status because the "Target link" is not yet specified. Put in any valid link name in the robot kinematic chain will change the status from "Error" to "OK". This time, using OMPL, a plan can be generated and the path of the target link can be seen on the screen (see Figure 2). Color, alpha, and line width can be changed.

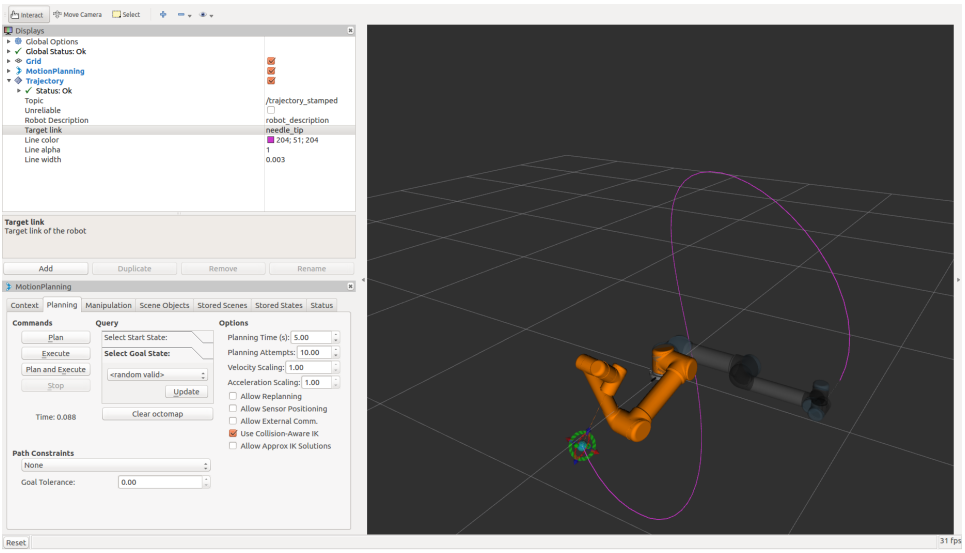

Figure 2: Trajectory plugin for visualizing link trajectories from a returned path.

The packages are available on *GitHub*. The videos are uploaded to YouTube: Part I and Part II.

4 Lesson Learned

- Nonholonomic path planning with RRT
- Kinematic modeling with Lie group approach
- *xacro:if* tag can be used to generate URDF iteratively
- *RViz* plugin and using *Qt* library
- A much deeper understanding of the inner workings of *MoveIt!*

5 Conclusion and Future Work

A summary of difficulties encountered during the course of this project is offered here.

1. After a plan is generated by a planning algorithm, *MoveIt!* will publish the return trajectory in the message type moveit_msgs/DisplayTrajectory. Interestingly, even though there are stamped fields in this message, the message itself does not contain a stamp on its top level, and therefore this message cannot be used as-is in *RViz*. Therefore, a new message type and a separate node have to be built in order to translate the original un-stamped verion to a stamped version. This is rather cumbersome, yet there isn't an alternative at this time of writing.

2. In nonholonomic path planning, because certain velocities are not achievable, and solving a boundary-value problem while considering collision is impractical, one has to resort to sampling-

based approach and discretize the allowable "actions" at each time step. Although there exists literature on discretized model for bevel-tip flexible needle, using a 4th-order Runge-Kutta integration approach results in a huge disparity for symmetric inputs. For example, a {+insertion, -rotation} input pair vs a {+insertion, +rotation} input pair on the same scale will result in two vastly different, asymmetric trajectories. In addition, integrating the system model over time will result in unexpected outcome, as shown in Figure 3. It is important to address the root cause of such unexpected behavior in numerical integration before moving on to path planning.

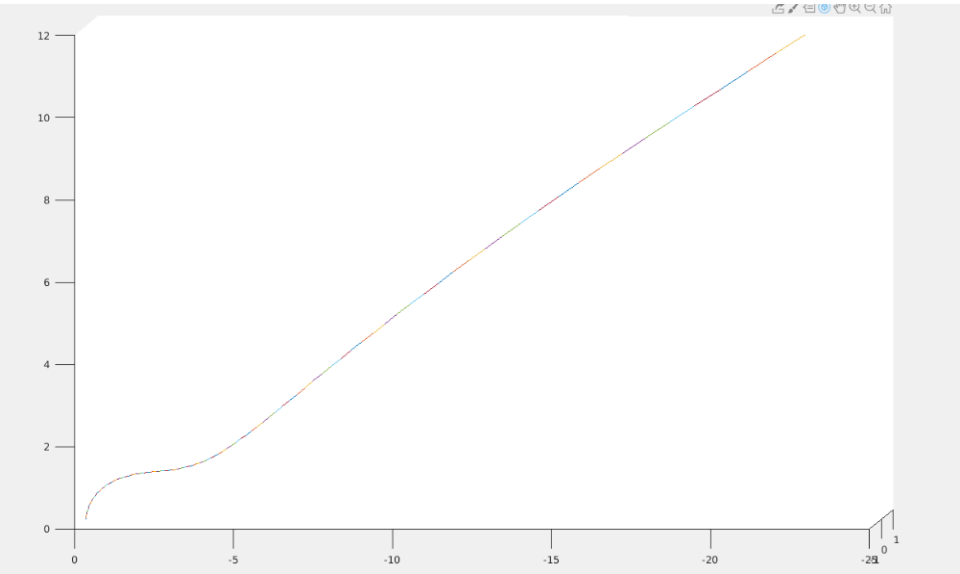

Figure 3: Needle path generated using 4th-order Runge-Kutta integration method with the same action applied over time. The most apparent issues are (a) the curvature should remain the same, and (b) the steps should remain constant.

6 Acknowledgement

The author would like to thank Dr. Simon Leonard for providing assistance and offering directions during times of difficulty. The author would also like to thank Dr. Jin Seob Kim for his suggestions for modeling and nonholonomic planning part of this project.