

1 Robot Manipulator Collision Avoidance Path Planning using STOMP

Notes:

- This project is of **30% CA**. Report and Codes count for 15% CA. Presentation counts for the other 15% CA. The evaluation will be based on
 - Correctness, quality, efficiency, practical considerations, and additional new features/novel implementation of your code,
 - Difficulty and novelty of your design and planning results,
 - Clarity, effectiveness, and visual illustration of your presentations, etc.
 - Your understanding of the theorem/algorithms,

Please highlight any of the above in your report and representation.

- This is a group project and will be evaluated by group.
- An incomplete sample code will be provided for your group to work on.

Tasks:

1. Complete the sample code given to you such that upon completion, one should be able to run the MATLAB live script `KINOVA_STOMP_Path_Planning.mlx` to successfully plan a collision-free path and show the animation (under the original collision avoidance settings). The following code files are incomplete: `helperSTOMP.m`, `updateJointsWorldPosition.m`, `stompDTheta.m`, `stompSamples.m`, `stompObstacleCost.m`.

2. Choose a different robot manipulator and plan the path for it in a similar scenario to the one in Task 1. On the help page of the MATLAB function `loadrobot()`, you can find a list of robot manipulators that are shipped with Matlab.

You can also load external robot. Here is a list of the robot manipulators supported in MoveIt: <https://moveit.ros.org/robots/>. The Matlab function `importrobot()` allows you to import an external robot by its URDF. But you also need to specify the mesh file in the command for visualization in Matlab (need some research). [Here](#) you can find the xacro and mesh file for Kuka KR210, which has a prismatic joint. You can convert the xacro file to URDF using the ROS command:

```
1  rosrun xacro xacro model.xacro > model.urdf
```

Use your chosen robot manipulator for the rest of the tasks.

3. Replace the MATLAB built-in forward kinematics (`getTransform`) with programs based-on the **twist theory**, using the **Product of Exponentials (PoE)** formula. You can use the code provided by the textbook (available on [Github](#), where the function `FKinSpace.m` calculates the forward kinematics given the space twists and end-effector pose in fixed frame. Note that it depends on other functions in that library). The textbook can be found in this [page](#). But your implementation/integration into your code need to consider code efficiency where necessary. You will need to determine the twists for each axis of your chosen robot according to its geometric parameters, which can be found by checking the corresponding URDF file, or inspecting the homogeneous transformation matrices associated with each link (see Section 1.1). You should explain clearly how you determine the twists in your report and presentation.

4. Create your own collision avoidance scenarios, such as adding additional objects, starting with different initial and final target configurations. Show your path planning results through overlapping the intermediate configurations or animation. The difficulty and novelty of your scenario will be taken into consideration for evaluation.
5. On top of the previous Tasks, further add constraints to the robot manipulator end-effector, such that it's y -axis is kept upright during moving from the initial position to the target position (imaging the task of moving a cup of coffee). Show the difference of the planning results between and after adding the constraints.

Remarks

- You do need to understand all the code to debug and finish all the tasks.
- It is possible that you have to modify the given code to adapt to your own code, for example, adopting a particular data format you defined and used, adjusting the obstacle positions according to the maximum reach of your chosen robot, etc. In addition, if you find the given code inconvenient or inefficient, feel free to modify/improve it. You can also use any external functions, libraries, code snippets, etc.
- Any contribution you make or features you add (e.g., more efficient calculation, adding a GUI for better interaction, fixing a bug in the existing code, improvement/extensions of the STOMP algorithm, etc.) will be considered for grading. Please point it out explicitly in your slides during the demonstration, as well as in your report, if any.

Submissions

- Make sure to submit both your PDF report and your MATLAB code. Include all your files into a folder and compress it to a zip file. **Only submit one zip file for your group.**
- **Naming convention:** Group<Number>_Report_Code.
- **Submission deadline:** Nov. 16th, 2023, 6:00 pm.
- For each task, please create a folder containing all necessary codes for reproducing your results. You may want to organize your code according to the 5 tasks, i.e., each folder for each task. You can also submit a single code base, but whatever you submit should make it easy for others to reproduce your solution for all tasks. You don't need to include the code provided by our textbook Modern Robotics (as mentioned in Task 3), if you didn't modify their code. It will be a good practice to include a readme.txt to guide the usage of your code.
- You can use PoE for all tasks in your submission.
- Your presentation serves to explain your work, and your understanding to us. You should explain what you have done, highlight what special features/interesting results you have achieved, etc.

Presentations

- **The presentation will be in our classroom E5-02-32 on Nov. 16th, Thursday, starting from 6PM to 9PM. Each group will have 10mins: 7mins for presentation and 3mins for Q&A.**
- You may choose to play a pre-record video for presentation, followed by the Q&A session (all members should present in person).
- **Each member should work on and present a part of your group's work.** You will present in group order, that is, from Group 1 to Group 14. Your group only need to join the meeting a few minutes before your scheduled presentation time and leave after the presentation.
- You will use your own laptop for presentations in E5-02-32.
- The whole presentation will be recorded (though we will meet in person).

1.1 Technical Tips

- The given code relies on the MATLAB Robotics System Toolbox. Make sure it is selected when you install the MATLAB.
- Webpage for how to use the `getTransform()` function. **Note** that when read the body names of the robot, Line 1 is much faster than Line 2.

```
1 robot_struct.BodyNames{k} //fast
2 robot_struct.Bodies{k}.Name //slow
```

In addition, there are some bugs in this function: try to use

```
1 // always correct, with respect to the base frame
2 getTransform(robot_struct, robot_struct.homeConfiguration, robot_struct.BodyNames{2})
3
4 // sometimes incorrect, depending on the robot model
5 getTransform(robot_struct, robot_struct.homeConfiguration, 'base_link',
robot_struct.BodyNames{2})
```

- To add constraint for the orientation of the end-effector in Task 5 (i.e., keep the y-axis upright, you can also choose to keep other axis upright), you can either
 1. choose an arbitrary orientation (rotation matrix), whose y-axis is upright wrt the world frame, and add a penalty (i.e., the constraint cost) to the cost function.
 2. Or, you can penalize the difference between the y-axis of the end-effector (the world coordinate of it, denote by \hat{y}_{sb} , which you can get from the forward kinematics) and the z-axis of the world frame (which is simply $\hat{z}_s = [0; 0; 1]$) only. For example, the penalty can be the so called l_1 norm of the residual vector: $\|\hat{y}_{sb} - \hat{z}_s\|_1 := \sum_{i=1}^3 |\hat{y}_{sb}(i) - \hat{z}_s(i)|$, which is simply the sum of the absolute values of each element in the residual vector. The l_1 norm can be calculated using the `norm(·, 1)` function, where the argument 1 denotes the l_1 norm.
- To find the robot's geometric parameters, you can load the robots:

```
1 >> robot_name = 'kinovaGen3';
2 robot = loadrobot(robot_name, 'DataFormat', 'column');
3 show(robot);
```

and then check the homogeneous transformation matrix. For example, the following shows the T_{01} , which is the configuration of the Joint 1 of KINOVA GEN3 to its Base frame:

```
1 >> robot.Bodies{1,1}.Joint.JointToParentTransform
2 ans =
3 1.0000    0    0    0
4 0   -1.0000    0.0000    0
5 0   -0.0000   -1.0000    0.1564
6 0    0    0    1.0000
```

- Alternatively, you can refer to the robot's URDF for its geometric information. The following lines shows where the URDF used by MATLAB is retrieved (the source URDF information):

```
1 >> [gen3Model, robotData]=loadrobot('kinovaGen3')
2 >> robotData.FilePath //the local path to the URDF of KinovaGen3
```

Open the URDF, you can find the following snippet, which is consistent with the information we found before: the `rpy="3.1416 0 0"` is the Euler angle (roll, pitch, yaw) representation of the orientation of Joint 1 with respect to Base, that is, if you rotate the Base round its x -axis by 3.1416 radian, then you will get the orientation of the Joint 1. The `xyz="0 0 0.15643"` shows the translation of the origin of the Joint 1 with respect to the origin of Base, expressed in Base frame.

```
1 <joint
2   name="Actuator1"
3   type="continuous">
4   <origin
5     xyz="0 0 0.15643"
6     rpy="3.1416 0 0" />
7   <parent
8     link="base_link" />
9   <child
10    link="Shoulder_Link" />
11   <axis
12     xyz="0 0 1" />
13   <limit
14     effort="39"
15     velocity="0.8727" />
16 </joint>
```

- Initialization of the joint angles can be influential, especially when there is a discontinuity/singularity. You might need to deal with the discontinuity of the joint angles $\theta \in [-\pi, \pi]$ to improve the initialization, thus improving the trajectory quality. You may also want to check the spherical linear interpolation function `slerp()` provided in MATLAB, which is however based on the quaternion representation of rotations.