Workspace Visualization for Robotic Arms

24783 Project Spring 2022

Team NamingSoHard
Daniel Deng, Kirtan Patel, William Hemler, Maheer Sayeed, Troy Gu

1. Problem Statement

The team aims to develop a program that will compute and visualize the workspace (i.e. reachable location of the end effector) of a serial robotic arm, given a standardized URDF input file that specifies the serial manipulators configuration.

2. Motivation

The motivation behind this project is to create an engineering tool that can be useful in many applications involving robotic manipulators. A visualization of the workspace of a robotic manipulator will give the user a realistic understanding of the robot's limits. If we further explore the project, we plan to give the user the ability to edit the configuration of the robotic arms in order to get a manipulator with a desired workspace. This can greatly simplify the complex task of designing robotic arms by giving immediate visual feedback for any changes.

Furthermore, working on this problem will allow us to gain experience using many of the topics taught in class such as multi-threading, data structures, file reading and optimization in a project scenario.

3. Project Feasibility

In order to achieve the minimum objective of the project, the final program needs to be able achieve the following tasks:

1. Read in a URDF file and extract parameters
2. Discretize input space and generate transformation matrices for forward kinematics computation.
3. Parallel-compute all reachable locations of the end-effector as a point cloud.
4. Store and output the results in a reasonable format.
5. Visualize the point cloud.

3.1 Reading Input File

Universal Robot Description Format (URDF) are standardized xml files that store robot configurations. An example URDF file of UR5 robot can be found here. There are existing interfaces available, such as MSXML from Microsoft or URDF parser from ROS to read in the file. Once we extract all the relevant information, we can store the arm configuration within a "robot Class", which contains a linked list. Each node of the linked list should be a "joint class" which holds information about the joint just like in the xml file. We could also implement a function to detect if a URDF is a valid arm, i.e. one fixed joint(the base), and all other joints should form a tree-like format (no detached joint).

3.2 Kinematics

Forward kinematics operation consists of a series of rigid body transformations based on the configuration of each link of the robotic arm structure. Each of these transformations between two connected links can be expressed in matrix form and thereby the location of the end-effector can be computed through a chain of matrix multiplications. We plan to use Eigen 3.4.0, which is an existing C++ library for linear algebra related algorithms**.**

In order to consider all possible combinations of joint angles, we will discretize the input space (e.g. length of prismatic joints, angle of revolute joints) and generate the corresponding transformation matrices. The operation is expected to be computationally expensive as the degree of freedom increases so we will implement multithreading to help reduce computation time. There are also matrix tricks to speed up the operation, such as constructing a large matrix that contains multiple configurations, which will be investigated.

3.3 Multithreading

Multithreading helps maximize the use of resources available and reduce the time needed to perform all the matrix operations, which can now be computed in parallel, and point cloud generation.

3.5 Output Results

For the basic need of outputting results, a text file would probably suffice. However, in order to use RVIZ from ROS to visualize the point clouds, we might need to output the results in a compatible format/data structure.

3.5 Visualization

For basic functionality, we'll be visualizing the point cloud (and the arm itself) using RVIZ from ROS. The implementation is expected to be very concise (about 10 LOC and a new CMakeLists.txt). Using an existing visualization tool serves as a simple and quick way for us to validate all the other work. Note that the main functionalities of the program will **NOT** use ROS. If time permits, we will implement our own visualization code instead of relying on RVIZ.

4. Possible Additional Features

Here we present a few extra functionalities for the project that we are considering to implement if time permits.

1. Collision check for the robotic arm.
2. Change the density and/or color of the point cloud for better visualization.
3. Inverse kinematics functionality.
4. GUI with clickables for adjusting robot configuration and/or selecting desired end-effector location for inverse kinematics.
5. Write out new URDF files.