Project Title
Humanoid robot writes words

Team members
Yixing Chen (yc3094)
Di Zhu (dz2311)
Cheng Zhang (cz2398)

Date
May, 04, 2016

Description
The main purpose of our project is to enable PR2 robot to write words using pen just like what human does in real life. The simulation is under ROS system with motion planning for arm using MoveIt!.

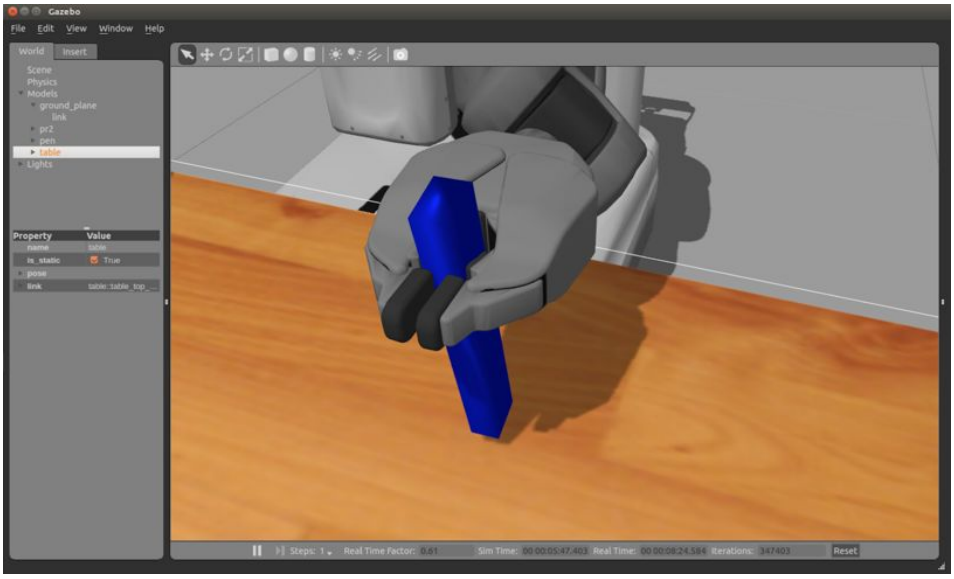
Project Structure

- system_launch - Start up Gazebo, MoveIt and load in PR2 robot and other scene objects
- robot_write - Move base towards the table + Grasp pen
- write_letter - Robot writes English alphabets based on user input
- points_and_lines - Display for letter in rviz

Process
1. Robot preparing movement

The first step of our project is for the PR2 robot to move towards the table and grasp the pen as firm as possible. The whole process involves the movement of the base and the arm, as well as the opening/closing of the gripper.

The manipulation of the base and the gripper is relatively straight-forward. The former is mainly based on TF package while the latter based on Pr2GripperCommandAction. However, in order to move the arm towards the given goal position in cartesian space (pen position), we have to solve a high dimensional IK problem. In fact, we use MoveIt! for the motion planning of robot arm. Actually, we have observed that this software works really well for our task.



The main challenge in this step is how to grasp the pen as steady as possible. The physical properties of the pen as well as the contact points between the gripper and pen all matters. We tried a set of grasping patterns and finally decide to adopt the way shown in the figure above. In addition, we modeled the pen as a long cuboid instead of a cylinder and increased the surface friction in urdf file, enabling the robot to write fluently without dropping the pen.
After all the preparing movement has been achieved, our robot is ready to write!

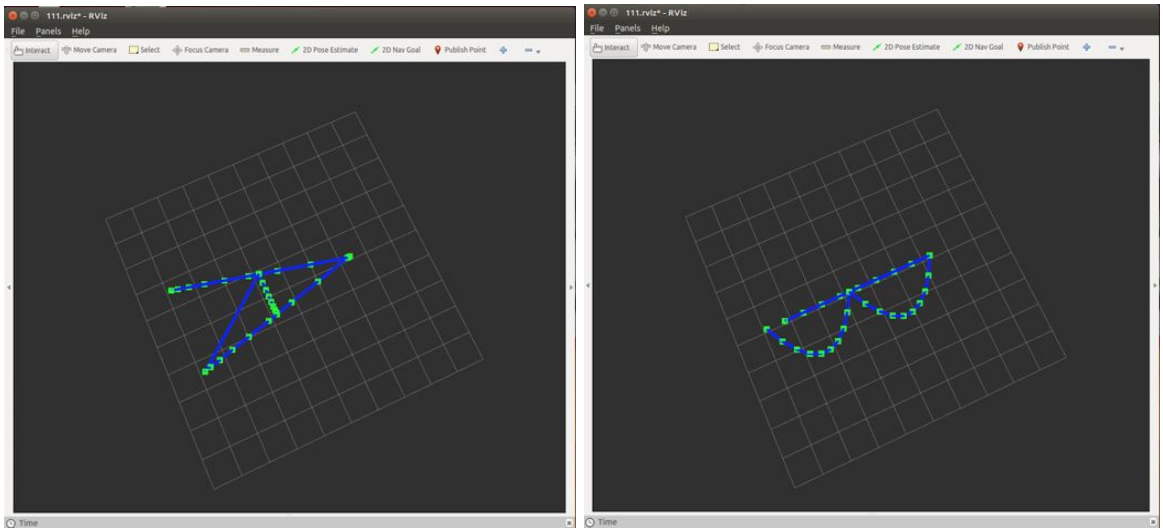
2. Motion planning

In this part, we compute the control points coordinates manually based on the characters “A” to “Z”. According to the rule of Bezier Curve, one path line is decided by three points, if the three points is in a straight line, the line is straight, otherwise the line is a curve.
Based on the scale of robot arm, we compute the points as follows.

Each line represents a character (26 lines, “A” to “Z”), in each line, three number represent a point, and three points decide a line.

3. Bezier Curve

According to the data of the Motion Planning part, we implemented the Bezier Curve algorithm. Instead of typing in the Coordinate of each point, we can simulate the whole trajectory by using several control points. The simulation trajectory works well in both straight line and curve.



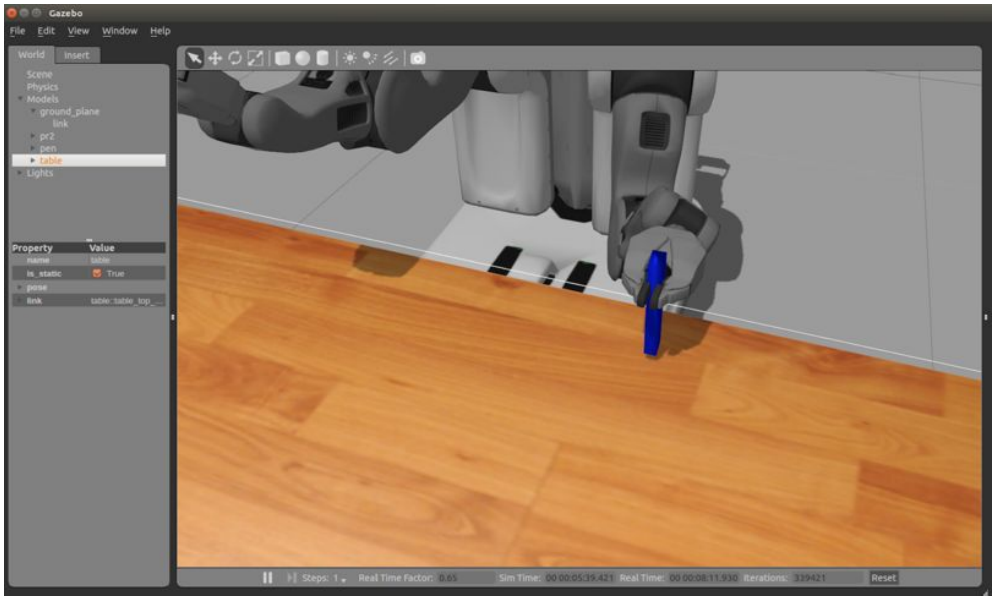
4. Word Display

The images above are the result of word display.

In this part, we build a cpp file named points_and_lines.cpp, this file continuously read the data of points coordinates in text file (the coordinates are produced by Bezier Curve algorithm), put the coordinates into vectors “points” and “lines”, and use `<visualization_msgs/Marker.h>` head file and `visualization_msgs::Marker` class to display the word trace in the Rviz window. Since the cpp file can continuously read the text file, so it can show the real-time word trajectory.

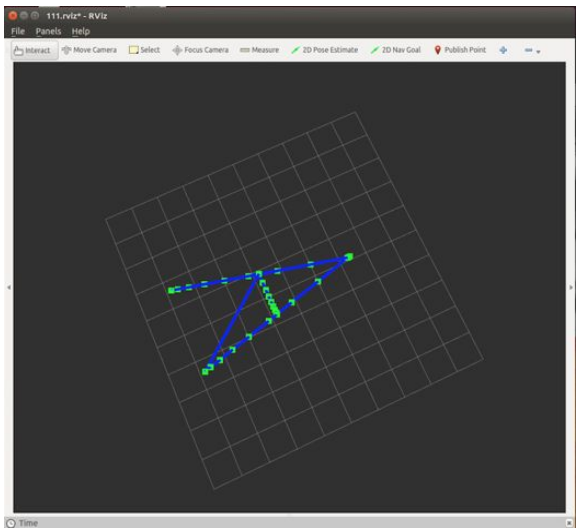
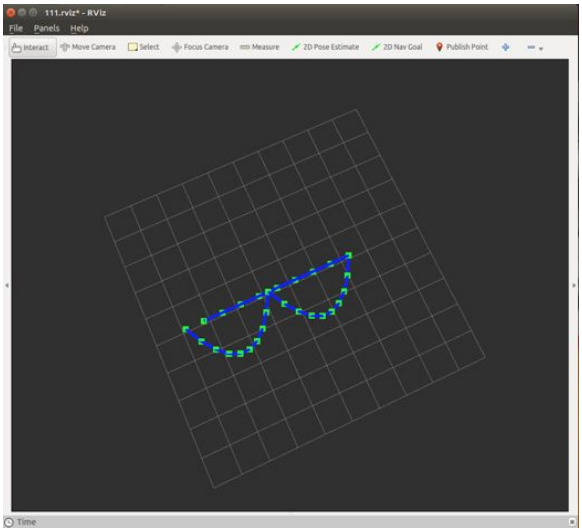
Result

Our robot - PR2, can hold the pen, which we use a blue rectangle to replace, pretty steadily.



By combining the Bezier Curve algorithm with the data from the motion planning part, the robot can write all 26 letters.

The pictures below shows the result trajectory of letter “A” and “B” written by our robot:

Division of Labor

Cheng Zhang (cz2398)
Robot preparing movement (Base movement and grasping)

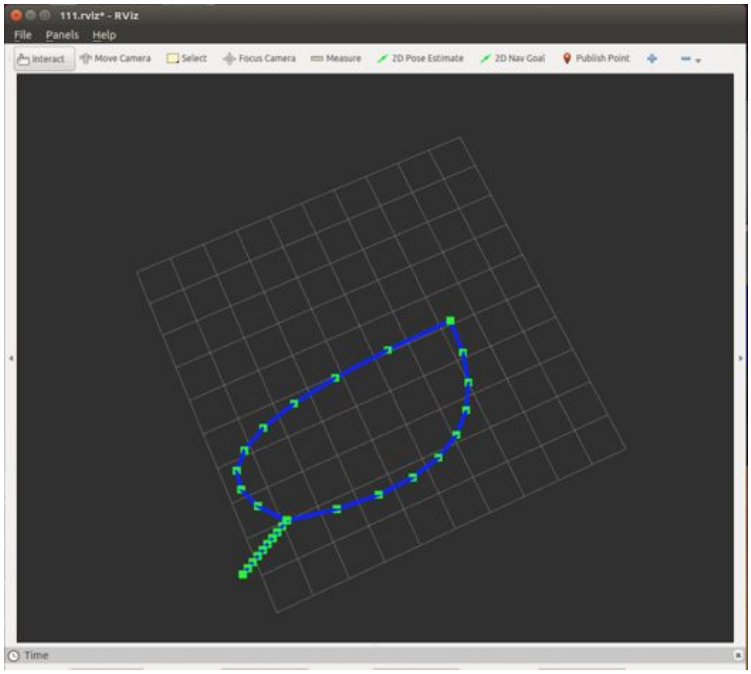
Yixing Chen (yc3094)
Motion planning and word trajectory display.

Di Zhu (dz2311)
Bezier curve algorithm implementation and trajectory tracking.

Limitation

Even though the Bezier Curve algorithm shows good simulation on straight lines and curves, it's not so good at drawing circles. When we were trying to draw the letter "Q" with a bunch of control points, the trajectory is not so natural as straight lines and curves.

Besides, we don't have enough time to implement the vision function. Since we spent a lot of time to adjust the motion plan data so that the Moveit! Could reach the target position. (Moveit! has limitation on movement ranges.)



Reference

1. Rviz - ROS wiki: http://wiki.ros.org/rviz
2. Move Group Interface/C++ API: http://docs.ros.org/indigo/api/pr2_moveit_tutorials/html/planning/src/doc/move_group_interface_tutorial.html
3. Pr2_controller tutorial: http://wiki.ros.org/pr2_controllers/Tutorials
4. Bezier Curve: https://en.wikipedia.org/wiki/B%C3%A9zier_curve