Project Report

Yangqi Su 800957989

Context

Building a robot that is capable of catching objects can translate naturally into many areas of engineering and even entertainment. The goal of this project was to implement 2 robot arms that can pass a ball between one another and the user. Such a system can be utilized in areas such as packaging and delivery, where the robot can help load packages in place of humans and yet still guaranteeing humans to work with freedom since the robot does not require the packages to be placed in a fixed way. Furthermore, such a system can also be used specifically in athletic training for sports that involve inanimate moving objects (i.e. basketball, soccer, etc.). Using a robotic arm in these cases can give accurate readouts for the athletes involved and provide personalized feedback and training for athletes to improve their accuracy (i.e. shooting a ball, throwing a pitch, etc.). These are just a few of the areas that such a model can be of use.

Task Completion:

As of the current deadline, one 6 dof robot arm has been implemented using kinematics only. The robot arm currently can only catch objects moving towards at a constant velocity.

Technical Description

The simulated robot arm contains 6 revolute joints as described in the accompanied URDF file. In order to acquire an analytical inverse kinematics solution for the joints, the last 3 revolute joints are at a single point connected to the end-effector (last 3 links are of length 0). However, if needed the program can be modified to accommodate links of any length, but would only use numerical inverse kinematics to solve the joints. Both the numerical inverse kinematics and analytical kinematics solutions were adopted from [1]

A ball was drawn with the ROS **visualization_marker** package, and is currently programmed to move at a constant velocity from an arbitrary starting point.

The link lengths and initial joint angles for the robot are read from an initialization file. Furthermore, the velocity and starting point of the ball are also read from the same file. Finally, the file contains an additional **alpha** parameter indicating the type of trajectory for the robot arm to generate in order to catch the ball with values of **alpha** indicated below:

- **0: Cartesian Straight-Line Trajectory**
- **1: Screw Trajectory**
- **2: Joint-space Straight-Line Trajectory**

The trajectories can utilize both quantic time-scaling and cubic time-scaling, and both are shown in the demos. All implementations were adopted from [1].

In order to catch the ball, the program first calculates whether the path of the ball intersects with workspace of the robot. If the ball is found not to cross the workspace, the arm does not move and a message that the ball is un-catchable is shown. If the path of the ball crosses the workspace, the arm calculated the entry and exit point (points in which the ball enters and exits the workspace) of the ball, and finds a close point to the exit to catch the ball. Upon catching the ball, the orientation of the end-effector is selected as follows: **the z axis points in the direction opposite the ball's velocity and the ball's velocity vector resides in the z-y frame.** Once the orientation and position of the end-effector is known, a trajectory from the robot arm's current position to the final configuration iss generated using one of the above 3 trajectory generation methods and the joint configurations are calculated with analytical inverse kinematics, the arm then tries to move to the position and orientation in half the time it takes for the ball to arrive. After the arm arrives at the final configuration, it calculates the distance between the ball and the arm. If such a distance is under **1mm**, the movement of the ball is stopped, and the ball is caught.
All of the simulations were done using Rviz.

Demos of the project are shown in the webpage:
https://sites.google.com/uncc.edu/ysu13-robotics-project/home

The file containing the initialization parameters are located in:
/path/to/Robotics_Project/IR_catkin_ws/src/spatial6r_description/launch/init_config.yaml

Command to run the program:
$ cd /path/to/Robotics_Project/IR_catkin_ws
$ catkin_make
$ source devel/setup.bash
$ roslaunch spatial6r_description rviz.launch

Future Work and Improvements:

Currently, the second and third link of the arm are of the same length so that the end-effector can reach all points within the spherical workspace created by the second and third link. However, if they were of different length, there would exist a smaller sphere within the larger sphere that is unattainable, yet the non-joint trajectory generation methods do not take this into consideration. Thus, a definite improvement would be to plan out several trajectories within the workspace to accommodate for shortcomings of a single trajectory.
Secondly, one could implement gravity and a parabolic curve for the ball, which should be straightforward since the end-effector's position and orientation only requires the ball's final instant velocity and position.
Throwing the ball would require a generating a different trajectory, but should also be straightforward as well. Once the robot can both throw and catch, adding another robot with the same capabilities would be simple. The final part would involve constructing interfaces for the user to launch the ball from arbitrary position and receiving the ball from the robot.

Reference
[1] Lynch, Kevin M., and Frank C. Park. Modern Robotics. Cambridge University Press, 2017.