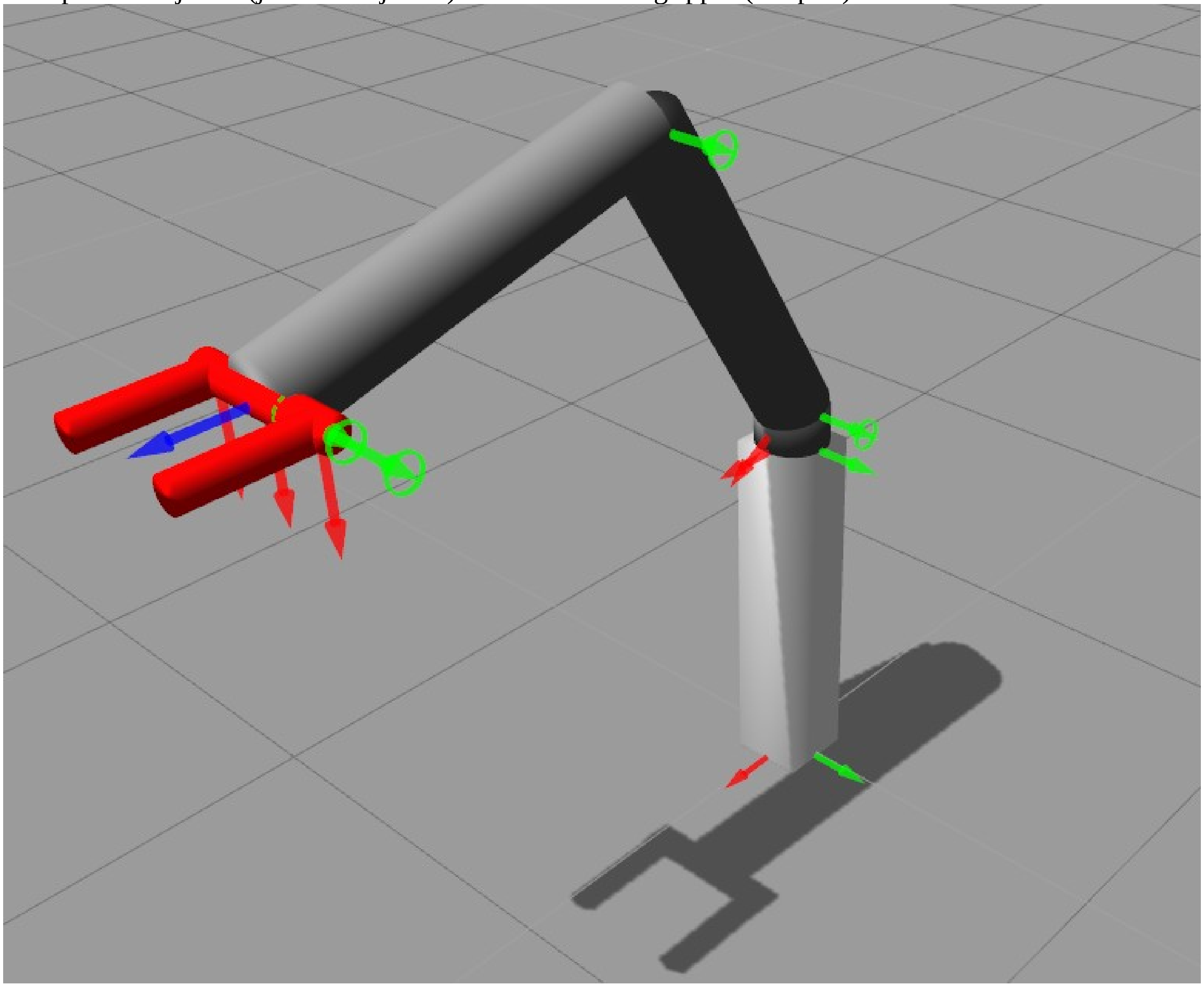
First introduce a new gripper robot I have defined in the “urdf/gripper_robot.urdf” file. As shown below, there are six joints in this robot. The joint properties are listed:

joint1: revolute, along z axis, between base_link and link1;
joint2: revolute, along y axis, between link1 and link2;
joint3: revolute, along y axis, between link2 and link3;
joint4: revolute, along y axis, between link3 and link4;
joint5: prismatic, along y axis, between link4 and link5;
joint6: prismatic, along y axis, between link4 and link6;

Two prismatic joints (joint5 and joint6) combines to the gripper (red part) of this robot.



In ps5_yxl1450/src, there are 4 cpp files:

gripper_robot_controller.cpp	*// controller of this gripper robot*
kpkv_service_client.cpp	*// tune the kp&kv for each joint*
gripper_robot_trajectory_action_server.cpp	*// trajectory action server*
gripper_robot_trajectory_action_client.cpp	*// trajectory action client*

Specifically

1. In the controller (*gripper_robot_controller.cpp*), each joint is treated the same way, no matter revolute or prismatic. A class define joint properties is used (from ps4_yxl1450), an improvement is

that service server to tune kp&kv is also added to joint class. Now this class “Joint” is capable of supporting as many joints as possible.

2. In the trajectory action server (*gripper_robot_trajectory_action_server.cpp*), multiple joint support is added, just a lot of detailed changes. This server interpolates trajectory linearly at sample time of 0.005 second. An improvement (I hope so..., at cpp114~140) is that, when jumping to the next segment trajectory for interpolation, it's possible that the trajectory sent in is more dense in time than the interpolation time rate. At this condition, this action server keeps jumping to the next segment and ends up doing nothing. New code is compatible with this condition, by extending *t_next* and *next_jnts* when condition is meet, so that this action server can interpolates any trajectory at its own time rate.

3. In the trajectory action client (*gripper_robot_trajectory_action_client.cpp*), a simple move task is performed. The task is grasp the beer on the table and place it at the center of the table. To make it happen, this movement is decomposed to ten steps in the following, and each of them will be sent to the action server as a goal.

1.move the gripper to the safe point
2.move the gripper to the top area of the beer
3.move the gripper around the beer
4.clamp the gripper to grasp the beer
5.move the gripper up with the beer
6.move the gripper to the above of target area (center of table)
7.move the gripper down to place the beer on table
8.unclamp the gripper and release the beer
9.move the gripper up from table
10.move the gripper back to the safe point

Some key information are obtained by communicating with Gazebo. Like using service “*/gazebo/get_joint_properties*” to get the current joint positions of the gripper robot, using “*/gazebo/get_model_state*” to get the position of the beer and table.

A video “gripper_robot_grasp_test.mp4” is submitted. In the video, trajectory action client has been run three time. In the first time it shows how this gripper robot moves. In the second time, the beer is moved to another place and action client gets this information . In the third time, the table is also moved to another place and action client gets this information for control.

In the picture “joint_pos_cmd.png”, the changes of joint commands are not clear because they don't change to much.

For running, in the terminal:

```
roscore
rosrun gazebo_ros gazebo
rosrun gazebo_ros spawn_model -file gripper_robot.urdf -urdf -model gripper_robot
```

(in the urdf directory)

```
rosrun ps5_yxl1450 gripper_robot_controller
rosrun ps5_yxl1450 gripper_robot_trajectory_action_server
rosrun ps5_yxl1450 gripper_robot_trajectory_action_client
```

in the Gazebo, before running the action client, insert the table and the beer, and set the gravity to -0.1 (kps and kvs have not been tuned well, this is the solution I have now).