RBE-500

Final Assignment Part 3

Group 9: Abhishek Jain, Paurvi Dixit, Surya Murugavel Ravishankar

Objective:

1. Velocity Level Kinematics: Modify your inverse kinematics node, such that it has two services: One takes joint velocities and converts them to end effector velocities, and the second one takes end effector velocities and converts them to joint velocities.
2. Extend the position controller in Part 2 to all the joints. (don't forget to revert the joint types.)
3. Move the robot to a position that is significantly away from singular configurations using you position controllers
4. Write velocity controllers for all the joints. For tuning the controller gains, you might need to fix the joints rather than the joint of consideration. Don't forget to revert the joint type to movable ones once you are done.
5. Give a constant velocity reference in the positive 'y' direction of the Cartesian space. Convert this velocity into the joint space using your Jacobian and feed it as a reference to your velocity controllers.This should make the robot move on a straight line in the +y direction.
6. Record the generated velocity references together with the actual velocity of the system over time, and plot via Matlab.

Implementation:

Que 1:

The objective is to modify the inverse kinematic code so that we are able to calculate both forward kinematics

- This was completed by changing the joint type in the robot's URDF file.
- Also, It was necessary to make changes in the control **yaml file** and **launch file** as previously the controller was defined for 3 movable joints.
- To calculate end effector velocities from given joint velocities, pseudo inverse of Jacobian is multiplied with end-effector velocities.
- Pseudo inverse is calculated using pinv() function.

```
def callback(data):
    values = data.v
    J = Jacobian(values[0], values[1], values[2])
    J = np.linalg.pinv(J)
    print(J)
    Qdot = np.matmul(J, [values[3], values[4], values[5],
                         values[6], values[7], values[8]])
    return inv_vResponse(Qdot)


def inv_v_server():
    rospy.init_node('inv_v_server')
    s = rospy.Service('inv_v', inv_v, callback)
    rospy.spin()
```

Fig: Calculating the pseudo inverse of the Jacobian

Que 2:

The objective is to extend position controller to the other two joints apart from the last joint.

- This is done by making changes in yaml. File and URDF file.
- We change the joint type from 'fixed' to 'revolute' for other 2 joints and we define the position controllers for the same in the yaml file

```
<joint name="joint4" type="revolute">
    <parent link="link2"/>
    <child link="link3"/>
    <origin xyz = "0 0 0" rpy = "0 0 0" />
    <axis xyz=" 0 1 0 " />
    <limit effort = "10000.0" lower = "-3.14" upper = "3.14" velocity = "1000"/>
    <hardwareInterface>EffortJointInterface</hardwareInterface>
</joint>
```

Fig: Changing the joint type to revolute

```
# Position Controllers ---------------------------------------
joint1_position_controller:
  type: effort_controllers/JointPositionController
  joint: joint1
  pid: {p: 150.0, i: 0.01, d: 80.0}

joint2_position_controller:
  type: effort_controllers/JointPositionController
  joint: joint4
  pid: {p: 150.0, i: 0.01, d: 80.0}

joint3_position_controller:
  type: effort_controllers/JointPositionController
  joint: joint6
  pid: {p: 150.0, i: 0.01, d: 80.0}
```

Fig: Defining the position controllers for all joints

Que 3:

The objective is to move our robot in a position which is away from singular positions.

- This is done by accessing the position controllers and giving the position command via a service named "control_p_service".
- The service was used in the previous assignments to use the position controller for the last joint

```
killswitch@Legion:~/catkin_ws$ rosservice call /controlp -- [0,1.7,0]
```

Fig: Giving position command

Ques 4:

The objective is to write the velocity controllers for the three joint. We do so by the following steps:-

- We define velocity controllers in the control yaml file of the type: effort_controller/ JointVelocityController. Here we will be controlling joint velocities by virtue of the effort to the joints.
- We define the PID parameters for the controller and tune them one by one using RQT interface.

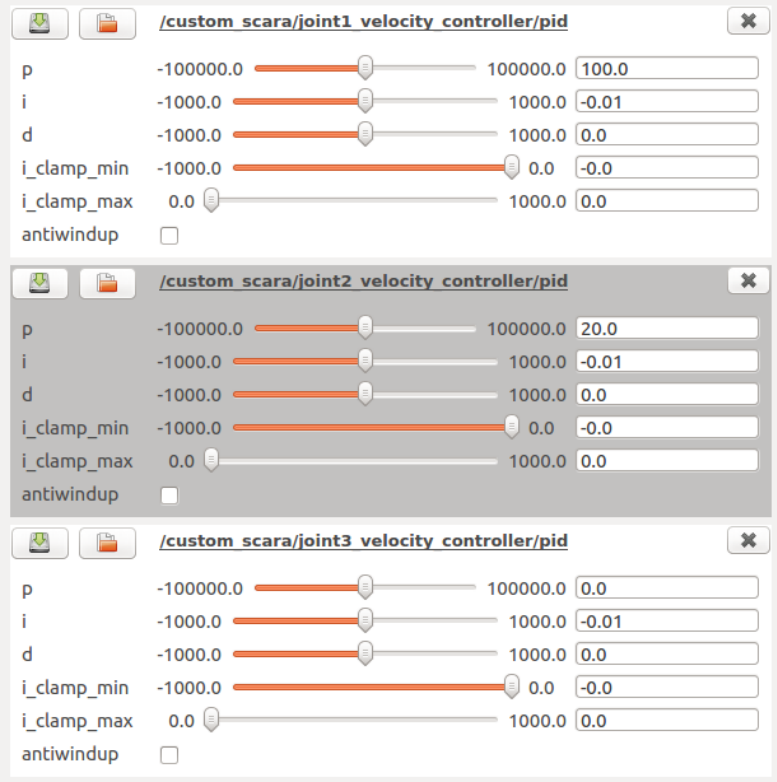

Fig: PID tuning via RQT interface

Ques 5:

The objective is to provide a constant velocity reference to the respective joints by converting the given cartesian reference to joint space using our modified inverse kinematics node.

- Here, We execute the same by defining services, publishers and subscribers.
- A 'switch controller' is used to switch from a position controller to a velocity controller.
- Node control_v_service subscribes to joint state topic of custom_scara.
- Service **'controlv'** is defined here which calls callback function. Postion from data is given to **'Inv_v'** service using proxy service which gives q_dot (inverse velocity kinematics).
- Data coming from joint States is published to joint_velocity_controller of the respective joints to reach the desired velocity reference.
- Then the 'frwd_v' service is used to convert the joint space velocities to cartesian space.
- All these joint velocities and reference values are then written in a file to plot graphs

```
def callback(data):
    global time_start
    time_start = rospy.get_rostime().secs
    time_start_n = rospy.get_rostime().nsecs
    rospy.wait_for_service(
        '/custom_scara/controller_manager/switch_controller')
    try:
        switch_controller = rospy.ServiceProxy(
            '/custom_scara/controller_manager/switch_controller', SwitchController)
        ret = switch_controller(['joint1_velocity_controller', 'joint2_velocity_controller', 'joint3_velocity_controller'],
                                ['joint1_position_controller', 'joint2_position_controller', 'joint3_position_controller'], 2)
    except rospy.ServiceException, e:
        print("Service call failed: %s" % e)

    global i
    global r
    global ref
    ref = data.q
    i = 1
    r = 1
    return controlvResponse()


def control_server():
    rospy.init_node('control_v_server')
    s = rospy.Service('controlv', controlv, callback)
    rospy.Subscriber('/custom_scara/joint_states', JointState, callback1)
    rospy.spin()
```

Fig: Defining services and subscribers and getting the input velocity reference

```
def callback1(data):
    global i
    global time_start
    global r
    global k
    if(i == 1):
        # print(ref)

        inverse = rospy.ServiceProxy('inv_v', inv_v)
        t1 = list(data.position)
        t1.extend(list(ref))
        resp = inverse(tuple(t1))
        pub1.publish(resp.q_dot[0])
        pub2.publish(resp.q_dot[1])
        pub3.publish(resp.q_dot[2])

        forward = rospy.ServiceProxy('frwd_v', frwd_v)
        t2 = list(data.position)
        t2.extend(list(data.velocity))
        # print(t2)
        resp1 = forward(tuple(t2))
```

Fig: Using subscriber callback function to feed values to velocity controller

```
if(r == 1):
    file1 = open(
        "/home/killswitch/catkin_ws/src/scara/frwd_kin/scripts/myfilev.txt", "w")
    file1.write(
        str(resp1.v[0])+' ' + str(resp1.v[1])+' ' + str(resp1.v[2])+' ' + str(k)+'\n')
    file1.close()
    r = 0
else:
    file1 = open(
        "/home/killswitch/catkin_ws/src/scara/frwd_kin/scripts/myfilev.txt", "a")
    file1.write(
        str(resp1.v[0])+' ' + str(resp1.v[1])+' ' + str(resp1.v[2])+' ' + str(k)+'\n')
    file1.close()
if(rospy.get_rostime().secs-time_start > 5):
    file1 = open(
        "/home/killswitch/catkin_ws/src/scara/frwd_kin/scripts/myfilev.txt", "r")
    y = file1.readlines()
```

Fig: Saving positions in a .txt file

Ques 6:

Once we store the values of the reference velocity and current velocity, We plot them against time (smaple time: 5s) using Matplotlib.

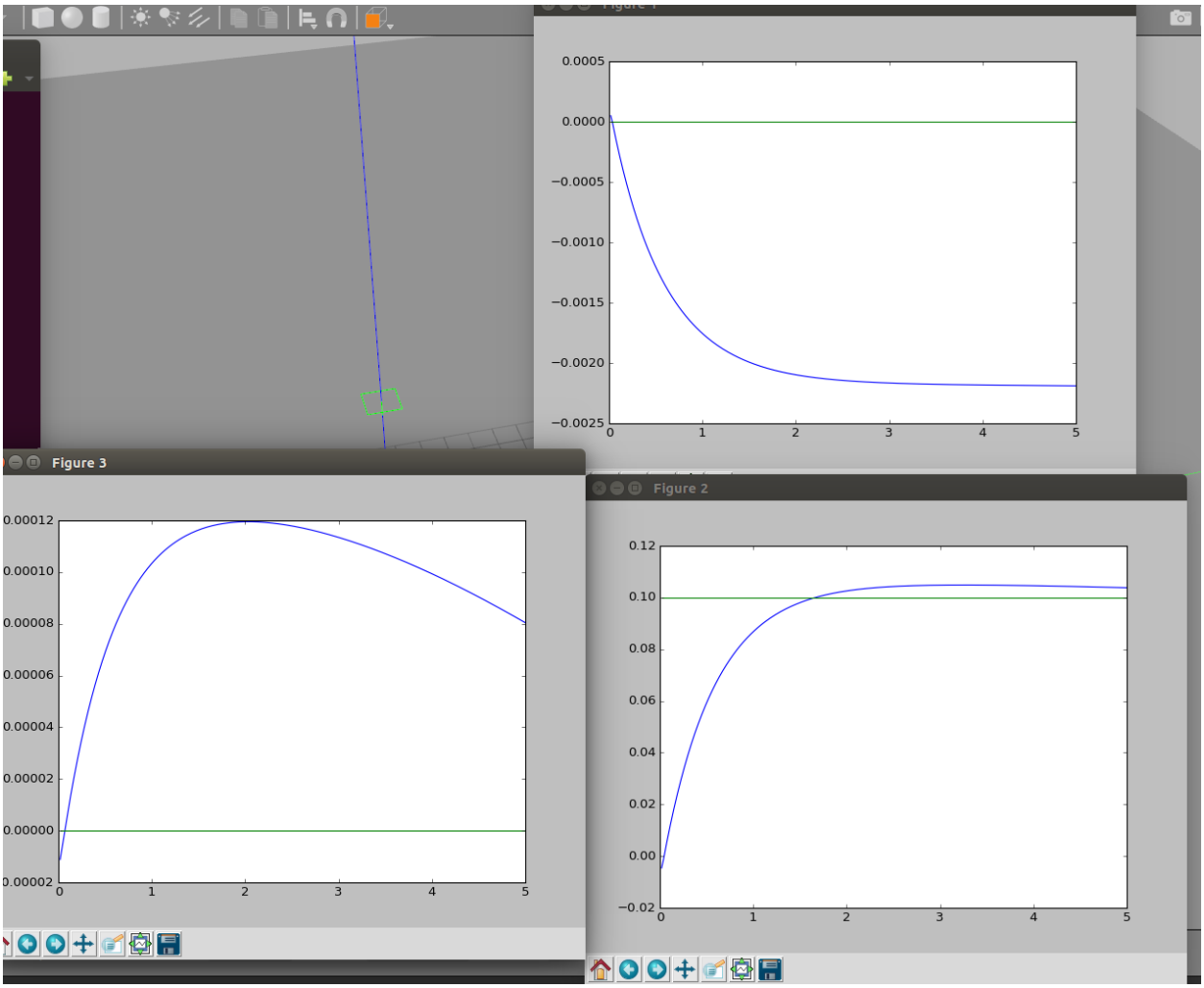

Fig: X, Y, Z graphs of the tip velocity with repute to time

```
killswitch@Legion:~/catkin_ws$ rosservice call /controlv -- [0,0.1,0,0,0,0]
```

Fig: Velocity command given for reference velocity

- Here, it is observed that the velocity along '+y' axis is achieved along with some very small variations along other 2 axis (possibly due to random variations).
- Figure1: 'X' velocity wrt. Time
- Figure2: 'Y' velocity wrt. Time
- Figure3: 'Z' velocity wrt. Time

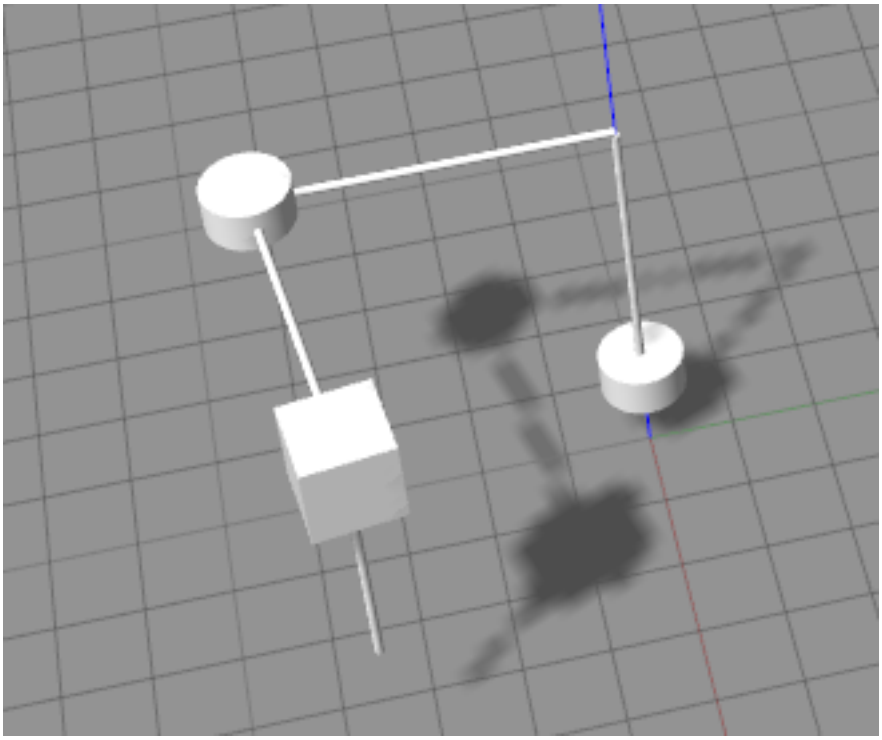

Fig: Initial Position given by position control

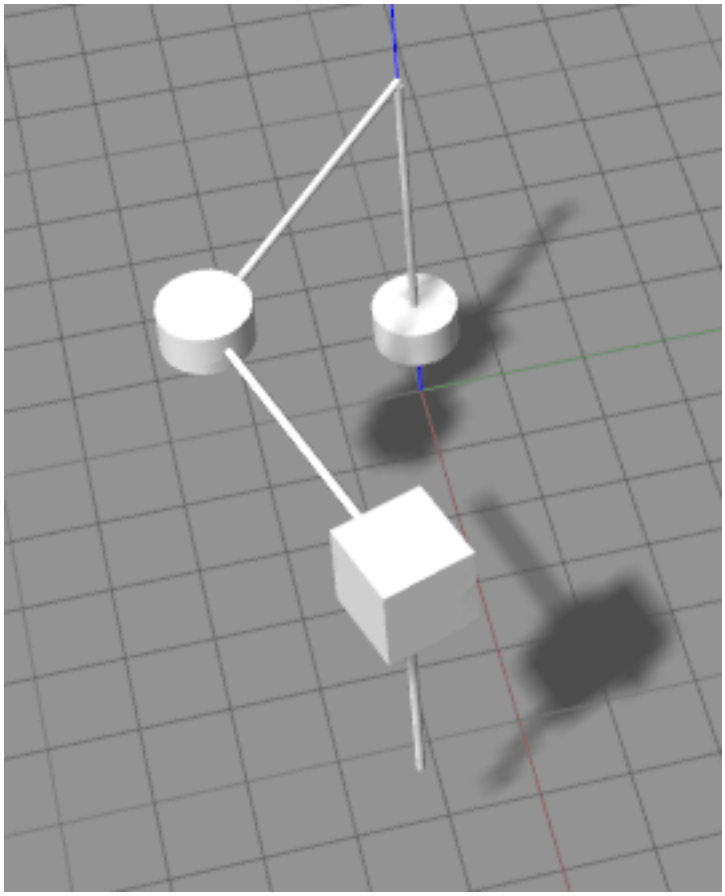

Fig: Final Position achieved by velocity control