Package overview:

- Inside `catkin_ws/src`, the main package is `scara_robot`. It does not directly contain any nodes or launch files, but is a way to organize all of the other nodes.
  - New package:
    - The `scara_pd_controller` package implements a proportional and derivative controller for the three controllable joints: joint 1 (revolute), 2 (revolute), and 5 (prismatic joint). The controller functions by reading the current joint position using the `gazebo/get_joint_properties` service, calculating the necessary input into the joint, and applying the input force using the `gazebo/apply_joint_effort` service. The controller receives the desired reference position using a custom service message under `scara/JointControlReference`.
  - Old packages used (from PA #1):
    - The `scara_gazebo` package includes the launch files for the gazebo world.
    - The `scara_description` package includes the URDF files for the robot as well as the rviz launch files.

Problems:

1. Velocity Level Kinematics: Implement a node with 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.) Move the robot to a position that is significantly away from singular configurations using you position controllers

   The PD position controller from PA2 was expanded to control the revolute joints (referred to as joints 1 and 3 in gazebo because of the fixed joints). The function was already made to take in arbitrary current positions, desired positions, gains, and the joint name, but minor changes were made to the way the error is passed into the function:

```python
def pd_control(joint, pos_cur, pos_des, kp, kd, err_old):
    err = pos_des - pos_cur
    d_err = (err - err_old)/(1/rate)
    f = -(kp*err + kd*d_err)

    if debug == True:
        print("\nerr = %f,  d_err = %f" % (err, d_err))
        print("\npos_des = %f, pos_cur = %f" % (pos_des, pos_cur))
        print("\nSending joint force f = [%f]" % (f)) # printing calculated values to terminal

    je_service = rospy.ServiceProxy('/gazebo/apply_joint_effort', ApplyJointEffort)
    zero_time = rospy.Time()
    tick = rospy.Duration(0, int((1/rate)*10**9))
    je_service(joint, f, zero_time, tick)

    if print_to_file == True:
        file1.write("%f,%f,%f,%f,%f\n" % (joint,pos_cur, pos_des,f,1/rate))

    return err
```

   The function created to request the joint position was also expanded to work for all three joints. This also calls the PD controller function using the current joint position, desired joint position, gains, and previous loop error (for the derivative calculation):

```
def request_joint_status(joint):
        global joint_pos

        joint_stauts = rospy.ServiceProxy('/gazebo/get_joint_properties',
            GetJointProperties)
        resp = joint_stauts(joint)

        if joint == 'joint1':
                joint_pos = resp.position[0]
                E_old[0] = pd_control('joint1', joint_pos, th1_des, kp[0], kd
                    [0], E_old[0])

        if joint == 'joint3':
                joint_pos = resp.position[0]
                E_old[1] = pd_control('joint3', joint_pos, th2_des, kp[1], kd
                    [1], E_old[1])

        if joint == 'joint5':
                joint_pos = -resp.position[0]
                E_old[2] = pd_control('joint5', joint_pos, d3_des, kp[2], kd[2],
                    E_old[2])

        if debug == True:
                print("\n\nReceived %f position: [%f] (meters)" % (joint,
                    joint_pos)) # printing received data to terminal

        return resp
```

The reference position service message file was updated to include the positions of the revolute joints:

```
float64 th1_des
float64 th2_des
float64 d3_des
---
float64 success
```

The service handler function was also updated to store the desired positions for the revolute joints:

```
        def service_handle(data):
        global th1_des
        global th2_des
        global d3_des

        th1_des = data.th1_des
        th2_des = data.th2_des
        d3_des = data.d3_des

        if debug == True:
                print("\nReceived reference positions [th1,th2,d3] = [%f,%f,%f]"
                    % (th1_des,th2_des,d3_des)) # printing converted values to
                    terminal

        if d3_des >= 0 or d3_des <=1:
                success = True
        else:
                success = False

        return success
```

Lastly, the server loop was updated to poll the positions of the revolute joints (which in turns calls the controller of each joint each cycle):

```
while not rospy.is_shutdown():
        request_joint_status('joint1')
        request_joint_status('joint3')
        request_joint_status('joint5')
        r.sleep()
```

****insert controller results for revolute joints****

3. 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.

4. Give a constant velocity reference in the positive y direction of the Cartesian space. Convert this velocity in to 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. Record the generated velocity references together with the actual velocity of the system over time, and plot via Matlab.