IVR Assignment

A - Robot Vision - Joint State Estimation

Method:

1. To find Joint angles:
    a. Use formula to find the angle between two vector: arccos((v1.v2) / |v1| x |v2|)
    b. Find the sign(+/-) with rotation vector and vectors of two links of the joints sign = signOf ((v1 x v2). vector_rotation)

3. To find Joint velocity:
    a. Joint velocity = current joint angle – previous joint angle

Results:

1. position of joints result:
    Work well in dark environment
2. Joint angles result: (average error)
    [0.023,0.027,0.026,0.014]
3. Joint velocity result: (average error):
    [0.727, 0.902, 1.24]

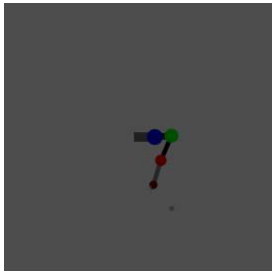
A - Robot Vision - Target Identification/Detection

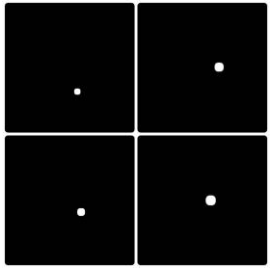
Method:

1. To find position of joints:
   a. Convert RGB graph to HSV
   b. Detect joint with range of h and s and peak value of v
2. To Detect target we used two methods:
   a. extract targets from xy-image and xz-image
   b. use cv2.findContours in OpenCV to find the edge of the target
   c. we used two method to distinguish between valid and invalid target:
      A. SVM classifier
      1.We constructed a SVM classifier and train it by adding train pictures.
      2. Classify the extracted target by the trained SVM classifier
      B. by pixel
      1. By extract the features of target (here we extract difference between white and black)

Results:

1. Accuracy of position of joints:





2. a. Accuracy of SVM classifier:
   1. The accuracy of SVM classifier is very low (56.4%) compare the other method, and to train the data, the whole process is very slow at same time.
   2. The

b. Accuracy of detect by pixels

The accuracy of this method is way more higher (about 100%) at this task, however if the target change, it will be very hard to detect.

B - Robot Control - Inverse Kinematics Velocity Control

Experiments:
Velocity control:

a. Jacobian:

Since the axis of rotation in each joint change in 3D dimension, to calculate the Jacobian in geometrically, we should calculate the rotation axis for each joint by doing:

We begin with the joint close to the base and apply rotation matrix about current joint to the rest of rotation axis.

Since the rotation axis could be neither x, y or z, we should apply Rodrigues' rotation formula

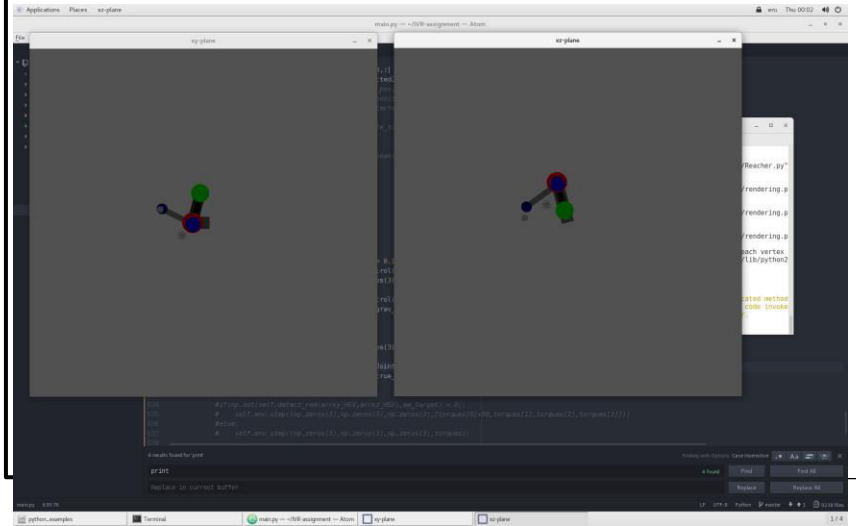
b. Inverse kinematics:

1. Calculate current position and error in position in task space
2. Calculate Jacobian:
    a. If the Jacobian is low rank then use the transpose
    b. Otherwise use pseudo-inverse
3. Apply inverted Jacobian on the position error in task space to get qdot

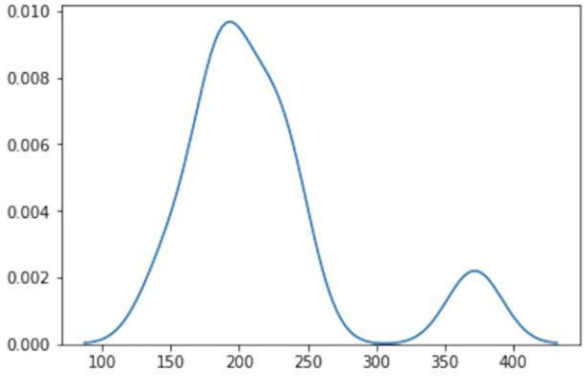
B - Robot Control - Inverse Kinematics Velocity Control

Result:

a. This is the density plot for the result of velocity control mode, according to the graph we can observe that the robotic arm can detect the target around 200ms
b. the graph shows the robot is stuck.



Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x7f3d67f6db38>

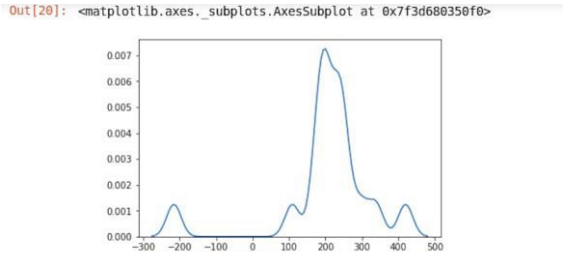


B - Robot Control - Gravity Compensated Torque Control

Experiments

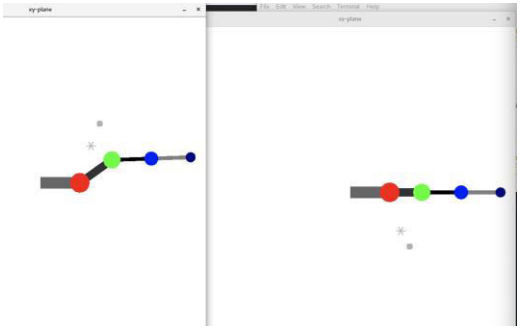
To calculate the torque against gravity, we use the method:

1. Calculate each joint's Jacobian transform
2. Obtain each joint's Cartesian location
3. Calculate each joint's torque by doing matrix rotation in y-axis, z-axis then z-axis.
4. Return each joint's torque.



Results:

1. According to the motion in xy-axis, the robotic arm stable
2. We observed a phenomenon that when we only use gravity against torque, the whole system, especially in the x-y plane still doing motions against time. However we did not find how to solve this.



C - Open Challenge

Open Challenge Details: implementing the Newton-Euler algorithm for inverse dynamics

1, compute joint angles acceleration(ddq) with joint angles velocity(dq) and dt
2, compute the Newton-Euler algorithm with three input (q,dq,ddq)
3, add it to the torques to reduce the affection between the movement of each joints

Problem:
Since there are errors when detecting the joints angles(q) and the dt are quite small, the errors have been amplified in angles velocity. When we calculate the acceleration in such way again, the error are getting much bigger and the direction of it keeps switching. Therefore, what we get from the NE algorithm are values with large errors, which, brings much more negative effect than positive effect.