Final Project
530.646 Robot DKDC
Fall 2019

October 19, 2020

Team members: Erica Tevere(etevere1), Hui-Yun Lin(hlin71), Zhiyi Jiang(zjiang38)

1 General Solution

For the final project, we had the goal of implementing a movement sequence detailed in figure 1 on the ur5 robot using an inverse kinematics based, differential kinematics based, and gradient descent based control method.

Our code for this project is module; we are able to run all three control schemes from our main final ur5_project.m. This file outputs to the command window the instructions to the user and prompts user when necessary. The steps of this main file are detailed below:

1. Prompt user to move robot to some user chosen home configuration - it is recommended that this be half-way between the two desired positions

2. User presses mouse buttons to record the home position joint angles

3. Prompt user to move robot to specified start configuration

4. User presses mouse buttons to record the start position joint angles

5. Prompt user to move robot to specified target configuration

6. User presses mouse buttons to record the target position joint angles

7. Move ur5 robot to home configuration

8. Compute an appropriate gesture move based on which desired configuration is input as the start position

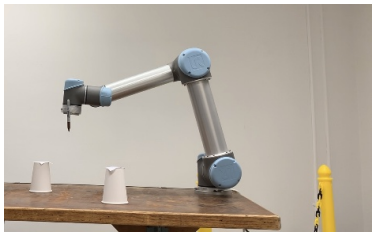
9. Move robot to gesture position to indicate which cup the robot will be moving towards first

10. Prompt user for the control scheme requested

11. User enters value corresponding to desired control scheme

12. Control scheme specified is executed with user input home, start, and target configurations

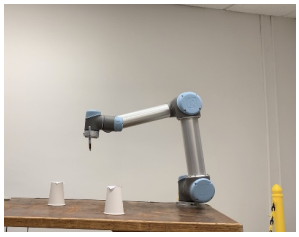
Further in the report, we will go into detail as to how each of these control schemes are achieved.



(a) 0.show gesture

(b) 1.move to the left

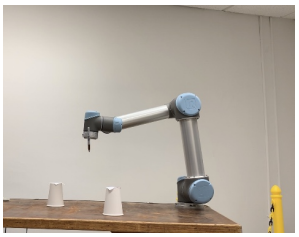
(c) 2.move straight down to mark



(d) 3.move back to the previous pose(home)

(e) 4.move to the opposite direction

(f) 5.move straight down to mark



(g) 6.move back to the previous pose(home)

Figure 1: place-and-mark movement sequence

2 Inverse Kinematics

Using inverse kinematics, move the robot along the desired Cartesian path by finding the corresponding path in joint space.

Step 1: We use **waitforbuttonpress** and **get_current_joints** to get the joint position of home, start and target position, storing them as $q_{home}, q_{start}, q_{target}$.

Step 2: We set weight as 0.3. So during this session, the arm will attempt to move to the start position but stop halfway, and then go back to home.

Step 3: Use forward kinematics function to get the transformation matrix g_{start} and g_{target}, which are the inputs for our inverse kinematics.

$*$ Note that the FwdKin function we used here in step 3 is derived from formula (1) and (2), where we get our T_j^i for the given DH.m file and input with the D-H parameters listed in Figure 2 such that q $\in$ {InvKin(FwdKin(q))}. Noted that q is our input joint position$[\theta_1\theta_2\theta_3\theta_4\theta_5\theta_6]$ which corresponds to the position of joint 1 to 6 of the ur5 robot respectively.

$$T_6^0(\theta_1,\theta_2,\theta_3,\theta_4,\theta_5,\theta_6) = T_1^0(\theta_1)T_2^1(\theta_2)T_3^2(\theta_3)T_4^3(\theta_4)T_5^4(\theta_5)T_6^5(\theta_6) \tag{1}$$

$$T_j^i = \begin{bmatrix} R_j^i & \vec{P}_j^i \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} x_x & y_x & z_x & (P_j^i)_x \\ x_y & y_y & z_y & (P_j^i)_y \\ x_z & y_z & z_z & (P_j^i)_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

Step 4: We also generated g_{start1} and $g_{target1}$, which are also the inputs for inverse kinematics to derive the right-above joint position of our start and target points.

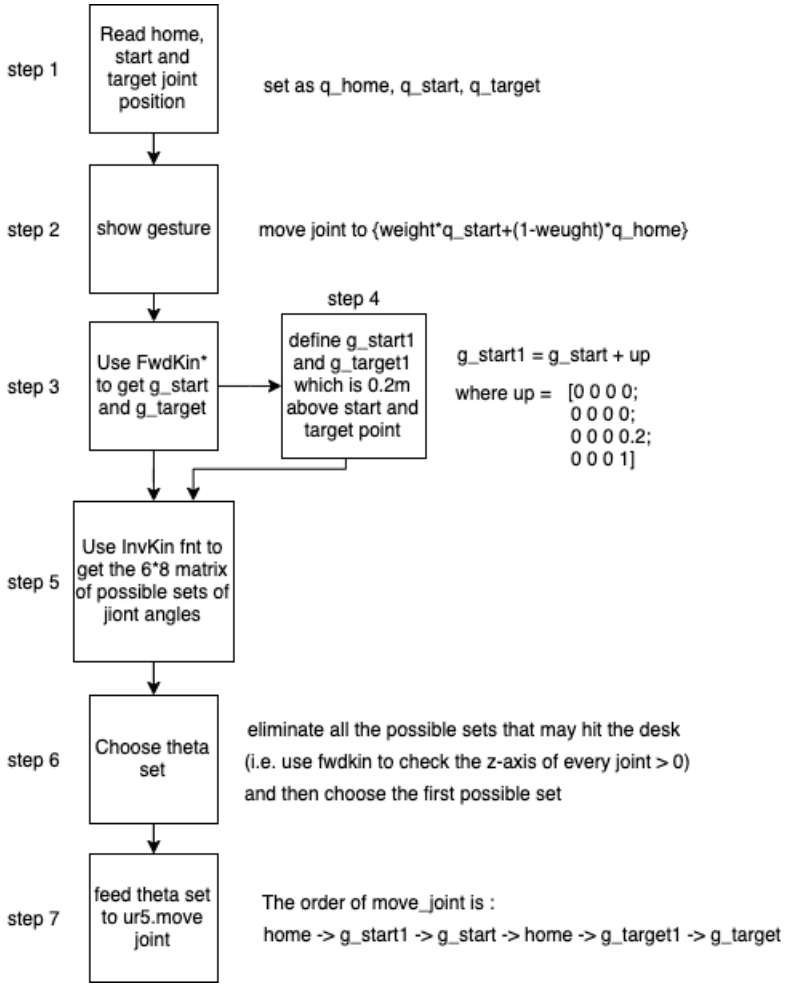


Figure 2: work flow for inverse kinematics

Step 5: Feed each of our transformation matrix we have into the giving InvKin function to get eight possible joint position.

Step 6: Choose the first joint position that won't hit the desk or hit itself. So basically we calculated the absolute z position of joint 3,4,5,6 to make sure they are all larger the 0. Take joint 3 for example, we times T_2^0 with $[0; 0; L1$(as shown in fig 4$); 1]$ so that the relative z position of joint 3 changed from body frame(joint 2) to space frame. We also make sure that the angle of joint 4 is limited between -135°to 135°so that joint 6 won't hit the upper arm.

Step 7: Feed the joint position we get from previous step into **move_joint** function.

By repeating Step 5 to 7 with different transformation matrices as input, we can move the robot in the following order: home → start1 → start → home → target1 → target

Step 8: Move the robot back home.

Experimental Results

Since the static pictures look all the same for three different control methods, we attached the link of our demo video instead. This method amongst all is the smoothest one because all we commanded to the robot is a start point and and end point, all of the movements in between are then generated by its inherent codes.

Demo video: https://youtu.be/UcNBqB17LLU

Joint	a	α	d	θ
1	0	$\pi/2$	0.089159	θ_1
2	-0.425	0	0	θ_2
3	-0.39225	0	0	θ_3
4	0	$\pi/2$	0.10915	θ_4
5	0	$-\pi/2$	0.09465	θ_5
6	0	0	0.0823	θ_6

Figure 3: D-H parameters

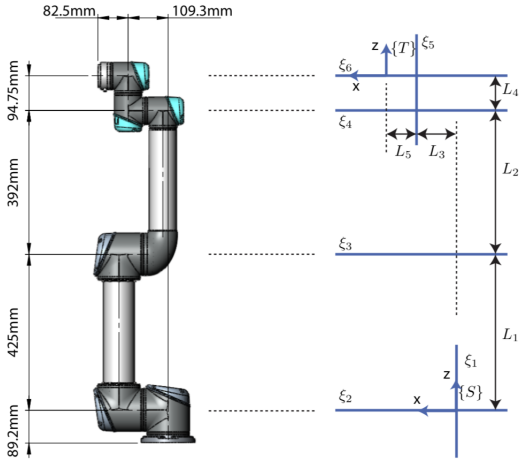


Figure 4: D-H parameters

3 Rate control using differential kinematics

Find the desired velocity (or small, differntial Carte- sian offset), and "pull that back" through the inverse of the Jacobian to find the joint space velocity (increment) that moves you in the right direction.The coding for this part utilized the function from Lab3. The main purpose is trying to make twist coordinate for forward kinematics between start position and target position zero. The input for this control scheme is inverse of body jacobian multiply body velocity which is equal to minus k multiply twist coordinate.

Algorithm steps
Step 1: Read home, start, and target position from user.
Step 2: Use FwdKin to get desired forward kinematics matrix. At the same time, a matrix was built to produce position above the target or above the start.
Step 3: Define all the useful parameters
Step 4: Enter the control loop. Use getXi to get twist coordinate Xi. At the same time the current joint angle was obtained from the robot.
Step 5: Utilize bodyJacobian function to get the body jacobian matrix. Before going to next step, the computer will check the singulairy at the current point by using determinant of body jacobian. Then, the resolved rate control was implemented by using the equation $[q_k p = q_k - dt * K * transpose(Jstb) * \xi_k]$
Step 6: Get a new position. Use getXi to get new twist coordinate. Divide it into two 3x1 vector and compare each vecotr's norm to the desired accuracy. The go back to step 4 until the accuracy was achieved

Overall Procedure:
Step 1:The user input home position, start position, and target position.The user decides to use resolved rate control. The robot shows the gesture of movement.
Step 2:The robot will start from home position. The control loop will help robot to move from home to the position above start point.

Step 3:The robot moves from position above start point to start point.
Step 4:The robot moves from start point to home.
Step 5:The robot moves from home to position above target point.
Step 6:The robot moves from target point to target.
Step 7:The robot goes back to home.

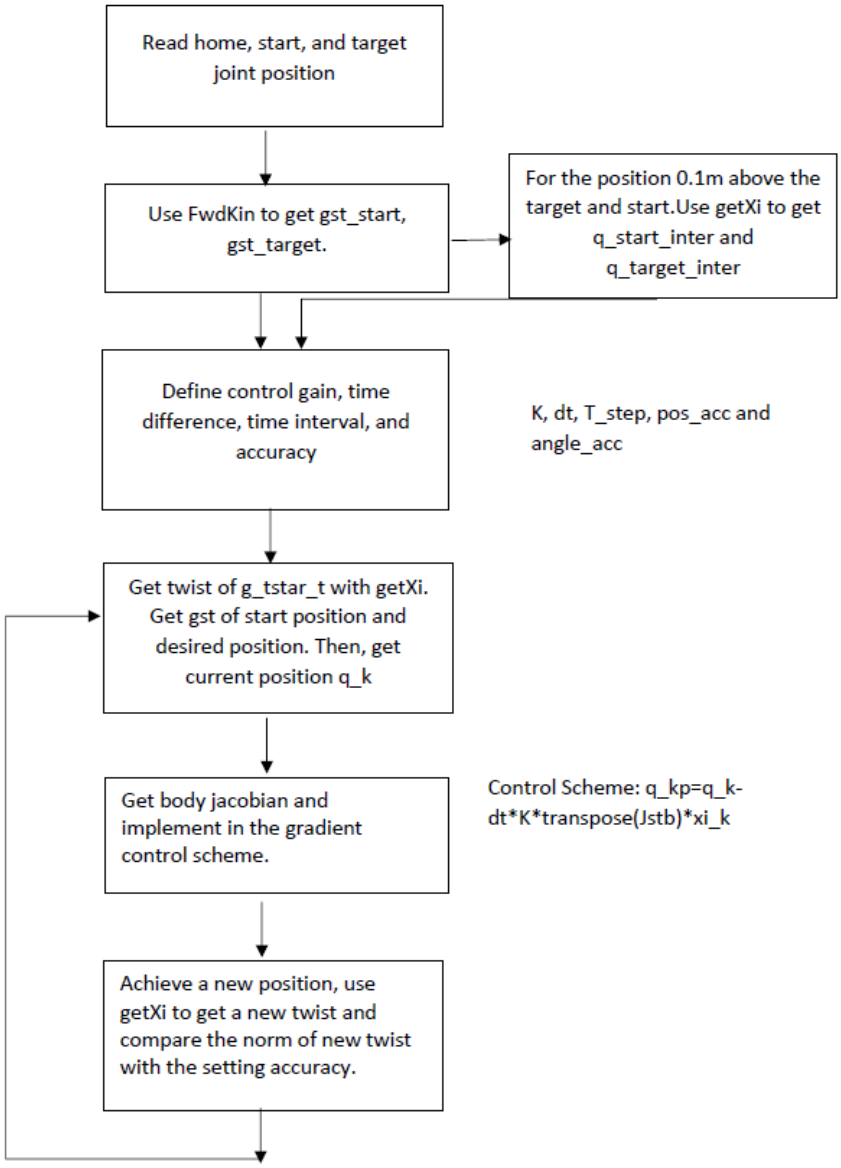
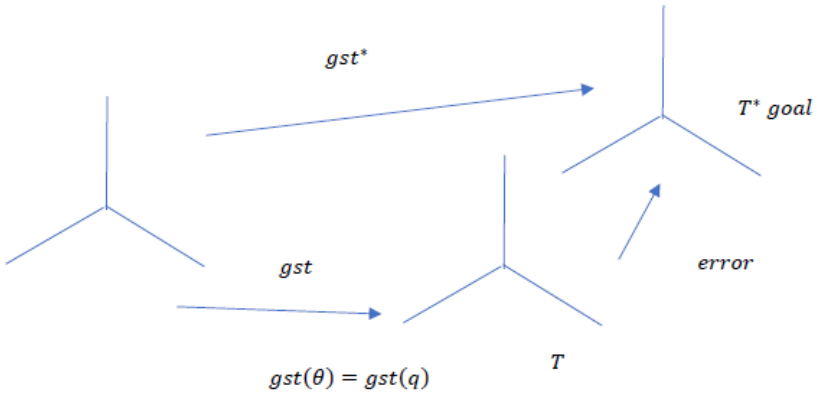


Figure 5: Rate Control Work Flow



$$gst(\theta) = gst(q)$$

$$error = g_{t^*t} = gst^{-1} * gst$$

At time $t = 0, g_{t^*t} = e^{\xi hat_0 * \theta_0}$ the goal for this method is to make g_{t^*t} stay along the same screw but when time is close to infinite $\theta(t)$ *is* 0.

For $V^b_{t^*t} = J^b_{st} * q_{dot}$,

The control input was set to be $-k\zeta$. Therefore, resolved rate control is:

$$q_{dot} = \left(J^b_{st}\right)^{-1}(-k\zeta)$$

For discretize form:

$$q_{k+1} = q_k - \delta t * k * \left(J^b_{st}\right)^{-1}\zeta_k$$

Figure 6: Explanation

Experimental Results

This method converges quickly at first few steps, but when its closer to target, it becomes harder to converge. So we tried to tune up our dt that times with joint velocity and tuned down T_{step}(time period we set for move joint) when joint velocity gets smaller to make our robot moves more smoothly.

Demo video: https://youtu.be/HLhyPWLH9u4

4 Gradient-based control

In this part, control scheme is similar. Instead of using inverse of body jacobian, the transpose of body jacobian is used to multiply body velocity, which is still negative k multiply twist coordinate. This method led to a slow robot movement. Therefore, all parameters need to be tuned again. To solve the problem, the team used dynamic equation to calculated time interval between each step. The equation use two parameters c and d. By multiply the system input with weight c, the change of time interval could be adjusted. To keep the time interval from too small, d was added to make sure its minimum value.

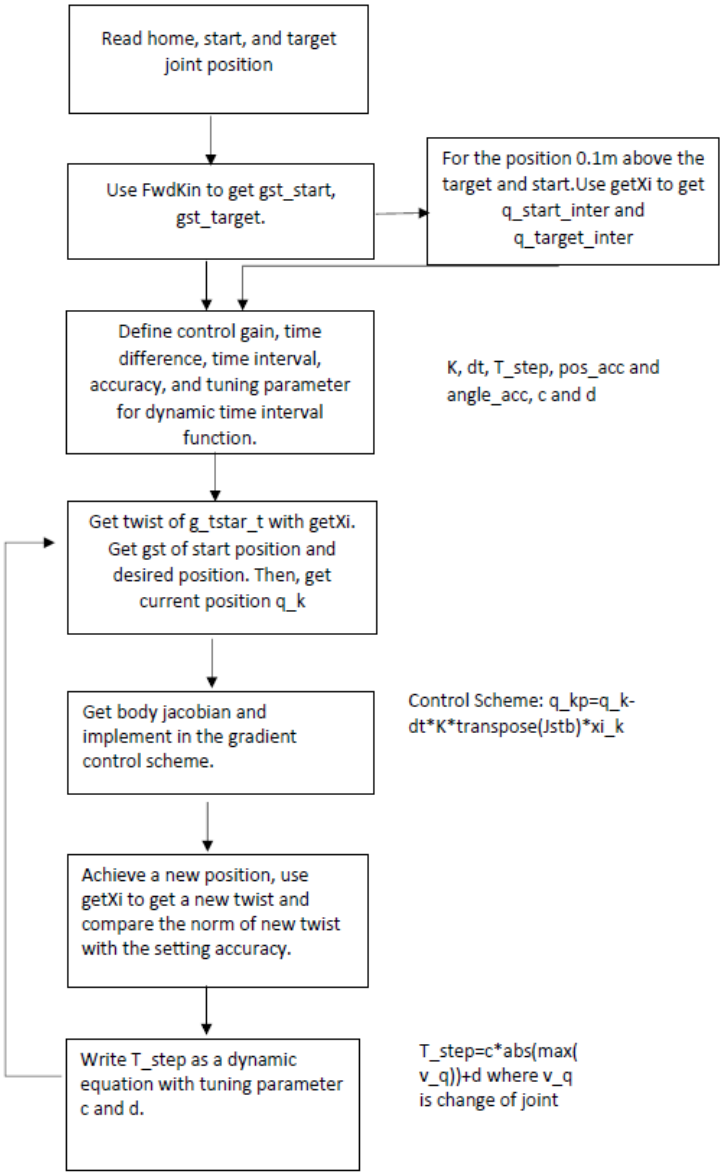


Figure 7: Gradient-base Control Work Flow

Experimental Results

We tried to tune T_{step} and dt separately in each stage to try to find the optimized sets of hyper-parameters, but it still took the robot more than 8 minutes to finish our series of motion.

Demo video: https://youtu.be/4CY9j3Mk1yk

5 Contribution of each team member

	Erica	Hui-Yun	Zhiyi
Inverse Kinematics	v	v	v
Differential Kinematics			v
Gradient-based Control	v	v	v
Code Modularization	v		
Running Test	v	v	v
Report	v	v	v