Robot Devices, Kinematics, Dynamic and Control (EN.530.646) Final Project

Huan Min, Ruijie Zhu, Shengni Xu

Abstract

The project purpose is to move the UR5 robot from the start position to the target location, which will both be inside the UR5's workspace and near to the bottom surface. As a result, the job is the same as the pick-and-place problem. Before traveling to the start position, the robot should be in its home configuration, or a defined intermediate configuration that is equidistant from both places (this might be the robot's "home" configuration). We create a full program that uses the UR5 and the end-effector to move the robot end-effector from the start place to the stop location. The program has three main methods: inverse kinematics, resolved -rate control using differential kinematics, and transpose jacobian. After testing in ROS, all the testing errors are less than 10^{-4}, which is close to 0 and demonstrates the success of the project.

Introduction

The object of this project is to use three distinct ways in the R-VIZ platform to emulate the UR5 robot's entire pick-and-place operation. Inverse kinematic technique, differential kinematic method, and transpose Jacobian method are three of these approaches. Users can enter the technique they want to utilize, as well as the start and target locations, in this project. The UR5 will then travel to 0.25m above the start point and descend directly to the start spot. Then repeat the process with the destination location. Any portion of the robot will not contact the surface ($z = 0$) or go below the surface throughout this phase. In addition, because robots arise in singularity, the system will return -1 in the algorithm. The system will finally return the orientational and positional faults. The work is now complete, and the robot returns to its original position.

Method

1 Inverse Kinematics

By using the function "ur5InvKin.m" which calculated possible eight sets of six joints angle solutions for the UR5 robot by using the inverse kinematics method, the main task for the final project team is finding the optimal solution from these eight possible angle solutions.

1. Input: $g_{06,}$ the 4x4 homogeneous transformation matrix with respect to the base link. In the TA free input mode, the team wrote one simple function generating one simple homogeneous transformation matrix with identical 3x3 matrix as the rotation matrix and the input 3x1 position matrix as p part. Then, used it as the input for "ur5InvKin.m" function
2. Output: A 6x8 matrix, each column represents one possible solution of the UR5 robot joints angle arrangement.

The main task of the team is choosing the optimal joints angle option generated from the "ur5InvKin.m" function, with the following criterions no singularity among robot links, minimal joint movement, and no collision between the robot links with the ground.

Algorithm

Step 1: Input $q = [q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8]$, eight groups of joints angle from q_1 to q_8
Step 2: Each element, q_1 for example, with have 6 angles from Θ_1 to Θ_6 of the UR5 angles.

- Use the function "ur5BodyJacobian.m" to calculate the body Jacobian of our UR5 robot J_b^i
- Then Use the function "manipulability.m" to calculate the singularity of our input angles' body Jacobian with the "detjac" method.

- Then the team set the criterion that abs(mu)> 0.0001 , and removed unqualified q from the optimal solution.
- By observing the UR5 model, the team found out the Θ_2 has tremendous influence on the collision of end-effector or other links of out UR5 robot. Therefore, the team decide to set the range of Θ_2 from 0 to -π to avoid potential collision.
- Use the forward kinematics function to get the g_{06} transformation matrix of this joint angle combination.
- Check whether $g_{06}(3,4) > 0$, avoid collision between the end-effector and the ground and remove the unqualified joints angle combination from the optimal solution
- q_{best} should have the smallest angels movement, the team needs to introduce one reference angle to compare as $\mathbf{q_r}$. For i = 1 we initialize the q_{best} to the first input joints angle set:

  $$q_{best} = q_1$$

  When i > 1, we need to check whether new candidate of d have smaller total moving angles compared to the old optimal selection by this following equation:

  $$\mathrm{norm}(q_i - q_r) < \mathrm{norm}(q_{best} - q_r)$$

  If this equation is satisfied, the new q_i will replace the old optimal selection. Otherwise, the q_{best} will remain the same.

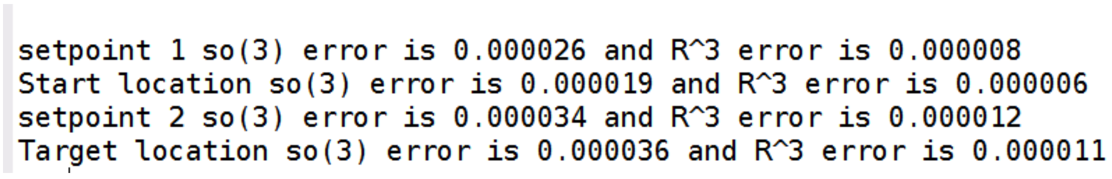
Error Result

By using the provided equation to calculate the error between the rotation matrix and position errors.

$$d_{SO(3)} = \sqrt{\mathrm{Tr}\left((R - R_d)(R - R_d)^T\right)}$$

$$d_{\mathbb{R}^3} = \|\mathbf{r} - \mathbf{r}_d\|.$$

We can get the following error result calculation from matlab window command.



```
setpoint 1 so(3) error is 0.000026 and R^3 error is 0.000008
Start location so(3) error is 0.000019 and R^3 error is 0.000006
setpoint 2 so(3) error is 0.000034 and R^3 error is 0.000012
Target location so(3) error is 0.000036 and R^3 error is 0.000011
```

Figure 1. Error Result Calculation

Simulation results

Because there is no access to the real UR5 robot, the team decided to do the simulation in the RVIZ simulation environment. The UR5 clearly moves much faster than other two control methods and the following pictures are the demonstration of the UR5 tool0 frame precisely coincide with the preset frames setpoint1, start location, setpoint2, target location.

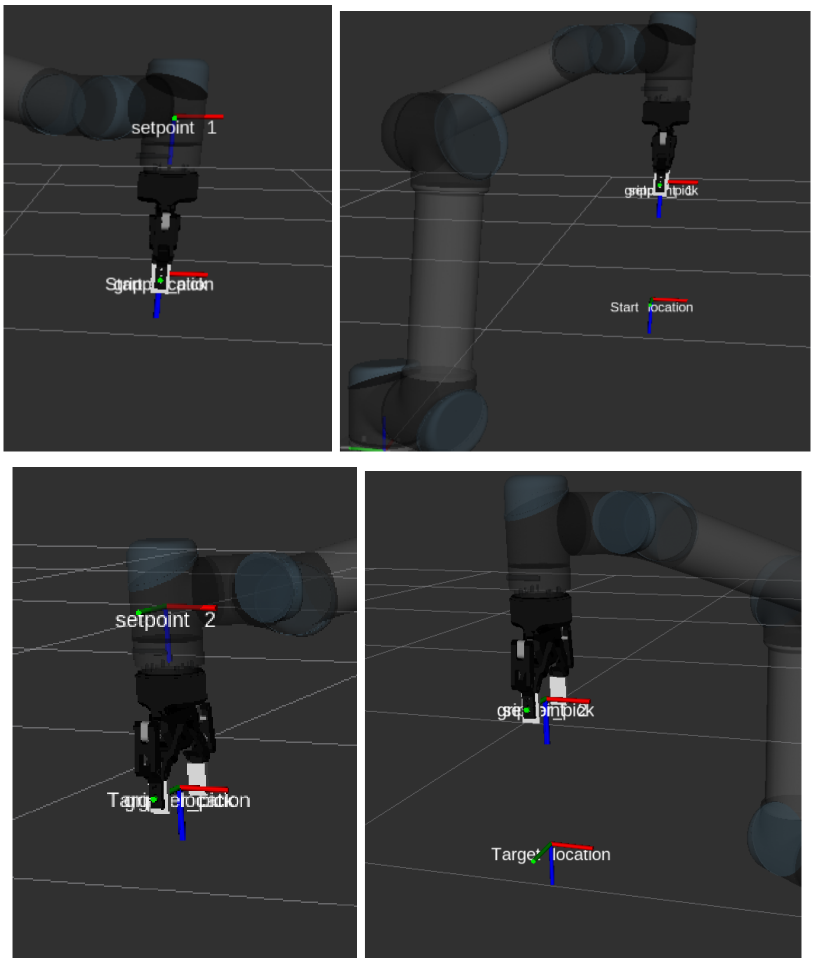


Figure 2. RVIZ simulation Tool0 frame coincide with frame setpoint1, start_location, setpoint2, target_location in order for IK method

2 Resolved -rate control using differential kinematics

Algorithm

To implement the pick-and-place task in a differential method, the first step is to understand what we are doing in Lab 3. As insert the desired point and gain, we control the ur5 approach to goal position in discrete time. To do so, we will use the following equation:

$$q_{k+1} = q_k - K\,T_{\text{step}}\left[J^b_{st}(q_k)\right]^{-1}\xi_k$$

Where:

q_k+1 = next state joint angles;
q_k = current state joint angles;
K = gain;
Tstep = time step;
Jb_st(qk) = current state body jacobian ;
ξ_k = velocity;

Where, at each time step, xi_k is get from the following equation and get conducted by function of getXi (include in the code package):

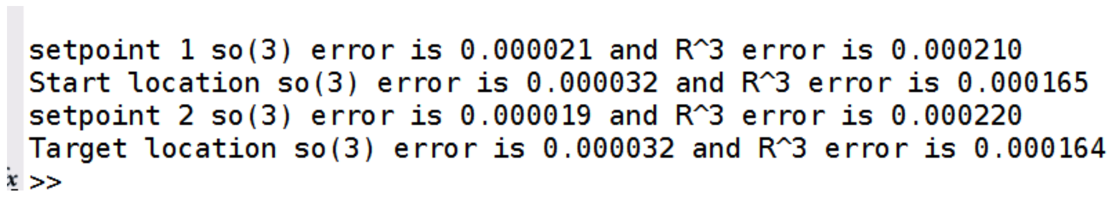
$$\exp(\hat{\xi}_k) = g_{t^*t} = g_{st^*}^{-1} g_{st}$$

As the next joint angle is updated by the previous joint angle state, the end effector of ur5 will start to approach the desired location and adjust its rotation angle to overlap with desired state's orientation.

Implementation

Step 1 : initialized the ur5 let it back to home position ;
Step 2 : move to a suitable home configuration ;
What we set is [pi/4 -pi/3 pi/4 -pi/2 -pi/2 0]'
Step 3 : As the user inserts the start and target location, the system automatically sets the start location at left side and target location at right side.
Step 4 : The user needs to press the button in the GUI to let ur5 do the next step, each button-press it's a safety check which makes sure that ur5 is moved to a desired orientation and it has finished the last movement.
Step 5 : The robot will move to the top of the start location after it finishes pick-up, it will go back to home position and move forward to the location above the target. Then move straight down to the target location.
Step 6: After ur5 moves to the target location, the matlab will give the feedback of location(R^3) and orientation(so(3)) error for each movement.

Error Results



```
setpoint 1 so(3) error is 0.000021 and R^3 error is 0.000210
Start location so(3) error is 0.000032 and R^3 error is 0.000165
setpoint 2 so(3) error is 0.000019 and R^3 error is 0.000220
Target location so(3) error is 0.000032 and R^3 error is 0.000164
>>
```

From the result we can see the errors are less than 10^{-4} , which is reasonable.

Simulation Result

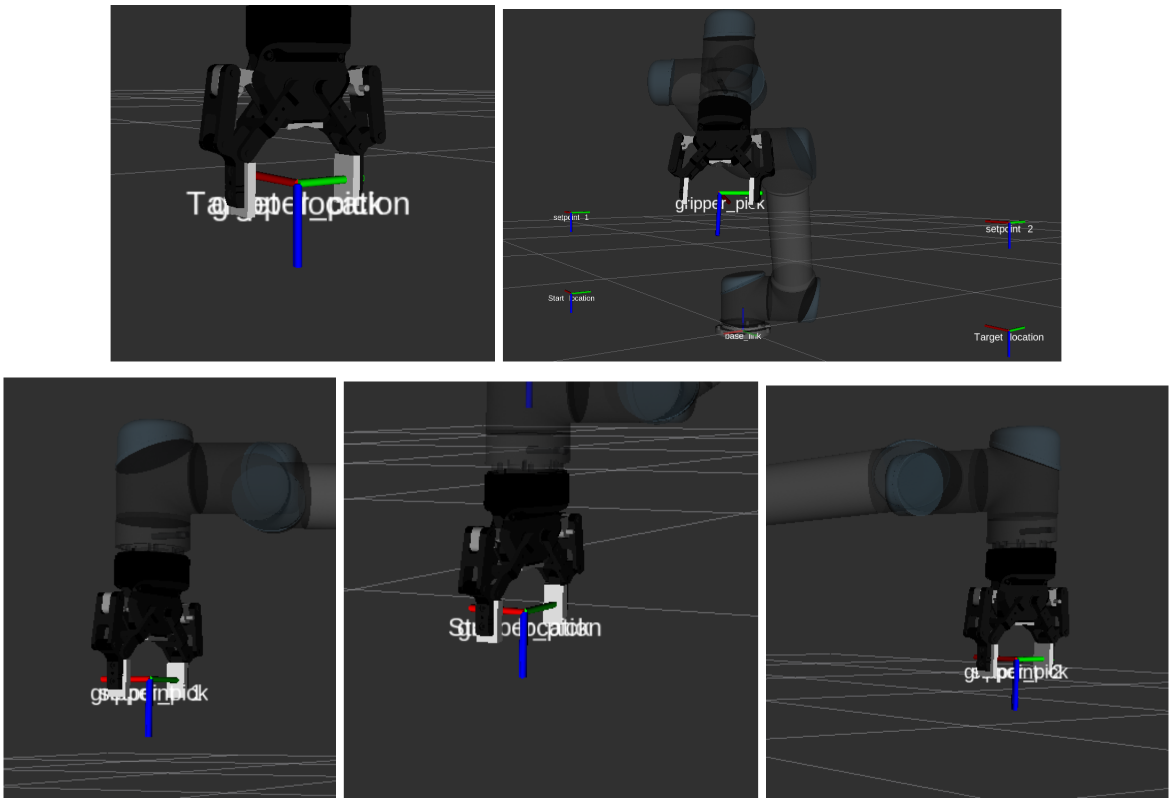


Figure 3. RVIZ simulation Tool0 frame coincide with frame setpoint1, start_location, setpoint2, target_location in order for RR method

3 Transpose - Jacobian Control:

Use transpose on jacobian calculation, let the robot move along a specified Cartesian path by finding the corresponding path in joint space. Most of this algorithm is like the inverse kinematics, but the way to calculate jacobian.

Equation

The core equation of this method is, which applies transpose of body jacobian

$$q_{k+1} = q_k - K * T_{step} * {J_b}^T * \xi$$

Where:

q_k+1 = next state joint angles;

q_k = current state joint angles;

K = gain;

Tstep = time step;

Jb = current body jacobian ;

ξ = velocity.

And Xi_k is calculated by the following equation with the MATLAB method getXi():

$$\exp(\hat{\xi}_k) = g_{t^*t} = g_{st^*}^{-1} g_{st}$$

Algorithm

Step 1: The input includes three parts. g_desired, is a homogeneous transform that is the desired end-effector vector pose, i.e. gst0. K, the gain on the controller. And ur5, the ur5 interface object.
Step 2: The output includes a variable 'finalerr', which is short for final error. It gives out a value of -1 if a failure occurs. If convergence is achieved, it gives the nal positional error in a unit of cm.
Step 3: In the order of the code, firstly we set hard code Tstep = 0.1. Then we call ur5.get_current_transformation method to get the current position of the base link and name it as g. And we read the current joint angle with ur5.get_current_joints and name it q. Then, we can get gst, the current rigid transformation using our previous function ur5FwdKin by input q.
Step 4: By using the previous function getXi, we get the current error for xi. From this variable, we get the current linear distance error from its row 1 to row 3, and the current angle error from row 4 to row 6. And here we initialize the final error as 0.
Step 5: While current linear distance error is smaller than 5cm and current angle error less than 15 degrees, we use the function ur5BodyJacobian to get body jacobian. And we check the singularity, we use function manipulability if the singularity is close to zero.
Step 6: Then we implement the control for q_k+1 with the equation:

q = q - K*Tstep*transpose(Jb)*xi

Here is the most difference with inverse kinematics control. And we control the range of Θ between $-\pi$ to π.
Step 7: Then we get a new gst. For this gst, check the safety for the bottom surface. After checking the safety, we get the new error matrix by calling getXi with inputting g_desired and new gst.
Step 8: Finally, we use g_desired and g_actual to get the final error.

Error Result

```
setpoint 1 so(3) error is 0.000022 and R^3 error is 0.000009
Start location so(3) error is 0.000035 and R^3 error is 0.000002
setpoint 2 so(3) error is 0.000013 and R^3 error is 0.000009
Target location so(3) error is 0.000035 and R^3 error is 0.000002
```

From the result we can see the errors are less than 10^{-4} , which is reasonable.

Simulation Result

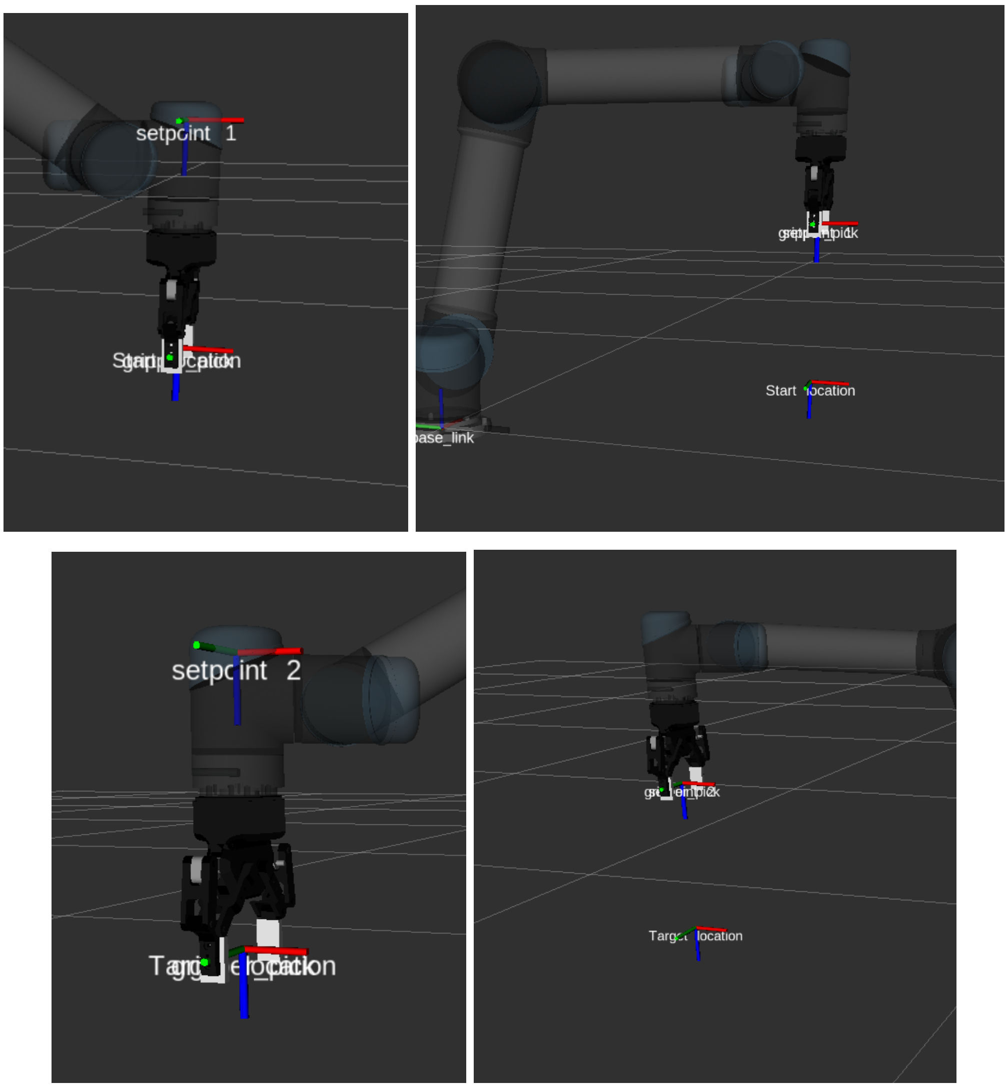


Figure 4. RVIZ simulation Tool0 frame coincide with frame setpoint1, start_location, setpoint2, target_location in order for TJ method

Future Improvement

When we run the main program of the three methods, there is a small problem related to the frame. When we first run the script with the tf_frame function, the tf_frame will create relative frames but will not display in Ros. Thus, you need to re-run the subsections in MATLAB, then the frames will display.

Conclusion

In this project, we wrote the program and moved UR5 to the appointed destinations. The program is written in three main methods: Inverse Kinematics, Resolved - Rate control with differential Kinematics, and Transpose - Jacobian Control. We reported errors between the specified location and the actual location during the simulation, and all errors are within the range of $1*10^{-4}$ and $9.9*10^{-4}$. In Ros we noticed that we have the highest running speed when we use the inverse kinematics method and the lowest running speed when we use the transpose Jacobian method. Therefore, in practice we made better use of inverse kinematics in route planning.

Work distribution:

Huan Min :

1. DK - based code implementation and simulation ;
2. wrote ur5_project
3. wrote the DK-based part in the report.

Shengni Xu:

1. TJ - based code implementation and simulation;
2. wrote the TJ-based part in the report;
3. edited report format.

Ruijie Zhu:

1. IK - based code implementation and simulation;
2. wrote the IK-based part in the report;
3. video production.