M.E. 530.646 Project

Ryan Keating and Noah Cowan
Johns Hopkins University

Put the ROS package for project in the `src` directory of your catkin workspace.

In this project, you will put to use what you have learned about robot kinematics this semester. You will compare trajectories planned in Cartesian space with Jacobian inverse and transpose methods to trajectories planned in joint space.

To complete this assignment, your team will write code in `project_main.cpp`. You will use rviz to test your code before running it on the actual UR5 hardware. To start rviz with the the UR5 model imported, enter the following command in the terminal:

```
roslaunch ur5 ur5_rviz.launch
```

Assignment

Be sure to comment all of your code!

In your report, show some examples to show each step works! Also, the instructor may test certain standard configurations above and beyond your examples.

1. Denote $T = T_6^0$ (to avoid proliferation of sub- and superscripts). Let $T = (R, p) \in \mathrm{SE}(3)$, where $R \in \mathrm{SO}(3)$ is the rotational part and $p \in \mathbb{R}^3$ is the translational part. Suppose you are given *any* two nontrivial homogenous transformations defining the position of the end-effector of the UR5 with respect to its base. These will be denoted as $T_i = (R_i, p_i), T_f = (R_f, p_f) \in \mathrm{SE}(3)$ for the initial and final configurations, respectively.
2. Write code to perform a test to ensure that these are both within the workspace of the robot.
3. Perform a test for whether or not either of the configurations is singular.
4. Execute a trajectory in joint space from T_i to T_f:
   (a) Using the inverse kinematics solutions for each of the two transformation (initial pose and final pose), choose the pair such that the joint space distance between them is minimized (i.e. minimize $||\vec{q_f} - \vec{q_i}||_2$)
   (b) Linearly interpolate between the initial and final joint values to define the trajectory.
5. Execute two trajectories in Cartesian space from T_i to T_f. In each case, you need to compute a velocity vector that "points" from the current configuration to the final configuraiton.

   One way to do this is to compute the "straight line" between the two poses. Since SO(3) is a very curvy space, there is no perfect "straight line" but one nice way to do this is to calculate the "error" between the two poses:

   $$\tilde{R} = R R_f^T \tag{1}$$

   where R is the *current* pose (not simply the initial pose, although obviously at $t = 0$, $R = R_i$). The special thing about the error rotation $\tilde{R}$ is it can be written as the exponential of some skew symmetric matrix $\hat{\omega}$,

   $$e^{\hat{w}} = \tilde{R} \tag{2}$$

where the vector $\omega \in \mathbb{R}^3$ is the closest thing to a straight line distance!

So, we take

$$\omega = -k_\omega w \tag{3}$$

where ω is the body velocity, and

$$v = -k_v(p - p_f) \tag{4}$$

where the gains k_ω and k_v define the speed of convergence.

(a) Prove that this works, namely that, assuming the body can always follow the ω and v above, that this will eventually arrive at the end goal. This question does NOT involve anything to do with the kinematics of the UR5! It is as if there are little jet packs on the end effector, driving it along the desired ω and v.

Hint: The translational part of this is fairly easy, so try that first (treat it as just moving a point in 3D space).

Hint: Do this in simulation to convince yourself it is right! It is POSSIBLE Prof. Cowan made a mistake in his formulation.

(b) Use the transpose of the Jacobian to map the angular and linear velocities into the generalized coordinates:

$$\dot{q}* = J^T \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{5}$$

This gives a desired joint velocity. Since we are going to implement "small steps" in joint space, you'll need to find a (small) discrete motion based on this desired joint velocity.

Execute the trajectory such that $q_{k+1} = q_k + \epsilon \dot{q}^*$, where ϵ is a small number. You'll have to choose this parameter to be "small enough" to give a good approximation to the derivative and "large enough" that the robot doesn't move too slowly.

(c) Repeat the previous problem, but instead of using J^T as the mapping into joint space, use J^{-1}. Make sure to use your code that checks being too close to a singularity, and abort the experiment and give an error to the user if this happens!

6. Compare the three trajectories executed in the previous questions.

7. Expand on these techniques in a creative way to demonstrate what you've learned.

Suggestions:

- Use the under-actuated gripper on the UR5 to do pick-and-place of some objects.
- Plan a trajectory with path constraints (e.g. workspace obstacles).
- Find a way to deal with passing through or very near to singularities, when using the inverse / transpose Jacobian matrix. In the inverse case, the calculation of the inverse can "blow up", and in the transpose case, you can get stuck in the null space of the transpose Jacobian and never get to your goal.
- Implement a vision-based control system. See Prof. Cowan for references if you are interested in this!
- Use a Kinect and do something cool.