Assignment 3: Planar Manipulator Kinematics

MM8803 Mobile Manipulation, 2020 Edition

February 18, 2020

Colab file

The assignment also comes with a Colab file, *Assignment3_Kinematics.ipy*nb. You can copy this file to Colab and run the different sections.

You should see that it skipped 5 unit tests. The assignment consists in fixing them all, and then using the result to do trajectory following for the robot.

1 Forward Kinematics

As discussed in the notes (please read them!) a simple recursive formula for the forward kinematics is given by

$$T_t^s(q) = T_t^s(q) = X_1^s Z_1^1(q_1) \dots X_j^{j-1} Z_j^j(q_j) \dots X_n^{n-1} Z_n^n(q_n) X_t^n.$$

Task 1: remove the skip decorator from the unit test "test_fk", and implement the arm.fk methods using the above formula. You should be able to do this with only two GTSAM functions:

1. $Pose2(x, y, theta)$constructs an $SE(2)$ pose
2. $compose(g0, g1, ...gn)$ composes all its arguments.

Re-run the script as you are debugging: the tests will be run and will only be happy when the FK formula is implemented correctly.

Debugging tips: use print debugging at leisure, but please remove them in the final python script, which has to be submitted. Work things out on paper. Make new unit tests, say, to test the position of the first axis.

2 Product of Exponentials, with Conjugation

Per the notes, for any serial manipulator with n joints, we have the following product of exponentials expression for the forward kinematics,

$$T_t^s(q) = \exp\left(\hat{\bar{\xi}}_1\theta_1\right)\ldots\exp\left(\hat{\bar{\xi}}_j\theta_j\right)\ldots\exp\left(\hat{\bar{\xi}}_n\theta_n\right)T_t^s(0)$$

where we can write each exponential as a simple conjugation, i.e., with a translation to and from the joint axis p_j:

$$\exp\left(\hat{\bar{\xi}}_j\theta_j\right) = \begin{bmatrix} I & p_j \\ 0 & 1 \end{bmatrix}\begin{bmatrix} R(\theta_j) & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} I & -p_j \\ 0 & 1 \end{bmatrix}$$

Task 2: remove the skip decorator from the unit test "test_con", and implement the arm.con methods using the above formula. You should not need any extra machinery, but you might want to create a helper function "expmap" that you call three times. Use the joint axes at rest, i.e., for $q = (\theta_1, \theta_2, \theta_3) = (0, 0, 0)$.

3 Product of Exponentials, for Reals

GTSAM, of course, has the $SE(2)$ exponential map built in, but it expects a finite twist as input. Re-read the notes to learn that in forward kinematics we only work with **unit twists** $\bar{\xi}$, with $\omega = 1$. Indeed, for revolute joints, the finite twist that corresponds to a 1 radian of rotation around the joint axis p is the **unit twist** $\bar{\xi} = (-p^\perp, 1) = (p_y, -p_x, 1)$.

Task 3: remove the skip decorator from the unit test "test_poe", and implement the arm.poe methods using the POE formula, but now using GTSAM. You will need two functions:

1. $vector3(x, y, z)$: little helper that creates a *numpy* three-vector.
2. $Pose2.Expmap(twist)$: takes a three-vector and produces a Pose2

Again, be sure to the joint axes at rest, i.e., for $q = (\theta_1, \theta_2, \theta_3) = (0, 0, 0)$.

4 The Manipulator Jacobian

The manipulator Jacobian is important both for inverse kinematics and trajectory following. The key is to derive a relationship between velocities $(\dot{x}, \dot{y}, \dot{\theta})$ in pose space in response to commanded velocities in joint space $\dot{q}$. This relationship is

locally linear, and hence we have the following expression at a given configuration q:

$$(\dot{x}, \dot{y}, \dot{\theta}) = J(q)\dot{q}$$

The quantity $J(q)$ above is the **manipulator Jacobian**.

For the three-link planar manipulator example, we can read off the pose $T(q)$ components from the forward kinematics equation in the notes, yielding

$$\begin{bmatrix} x(q) \\ y(q) \\ \theta(q) \end{bmatrix} = \begin{bmatrix} -3.5\sin\theta_1 - 3.5\sin\alpha - 2.5\sin\beta \\ 3.5\cos\theta_1 + 3.5\cos\alpha + 3.5\cos\beta \\ \beta + \pi/2 \end{bmatrix}$$

where $\alpha = \theta_1 + \theta_2$ and $\beta = \theta_1 + \theta_2 + \theta_3$. Hence, the 3×3 Jacobian $J(q)$ can be computed as

$$\begin{pmatrix} -3.5\cos\theta_1 - 3.5\cos\alpha - 2.5\cos\beta & -3.5\cos\alpha - 2.5\cos\beta & -2.5\cos\beta \\ -3.5\sin\theta_1 - 3.5\sin\alpha - 2.5\sin\beta & -3.5\sin\alpha - 2.5\sin\beta & -2.5\sin\beta \\ 1 & 1 & 1 \end{pmatrix}$$

Task 4: remove the skip decorator from the unit test "test_jacobian", and implement the arm.jacobian method. You just need some python math functions, and the *np.array* method. Google it.

5 Inverse Kinematics

As we discussed in class, we can implement inverse kinematics using gradient descent. However, that is horribly slow. Please read the notes about second-order methods, like damped least-squares. It turns that we do not even need the damping, we can use straight-up Gauss-Newton optimization, by iterating:

$$q \leftarrow q + \left(J^T J\right)^{-1} J^T \left(fk(q) - p_d\right)$$

starting from some initial estimate q_0. I supplied an initial estimate in the code, because it is important not to start at a singularity (e.g., 0,0,0).

Task 5: remove the skip decorator from the unit test "test_ik", and implement the arm.ik method. You will need three functions:

1. $delta(g0, g1)$: returns the 3-dimensional error $(g1.x - g0.x, g1.y - g0.y, g1.theta - g0.theta)$
2. $np.linalg.norm$: returns the norm of a vector, which you can use to check convergence
3. $np.linalg.pinv$: implements the pseudo-inverse $\left(J^T J\right)^{-1} J^T$

Common mistakes are getting the sign wrong.

6 Trajectory Following

If you have implemented everything correctly (all unit tests pass!), you should be able to uncomment line "run_example()" in the main function at the end of the file. Run the script again, and you should now see the arm executing a straight trajectory in a matplotlib window.

Task 6: take a screenshot of the final result, which should show the movement of the arm. No video needed.

Task 7: create a short write up that provides a line-by-line explanation of the run_example method, omitting any plotting commands. Be sure to properly explain the "q += np.dot(np.linalg.inv(J), error)" line

Deliverables

Please submit 2 separate files (which allows us to grade more easily):

1. Colab file for which all unit tests pass
2. Write-up about trajectory following, in txt, pdf, or md file.