Foundations of Robotics

Project 2

Inverse Kinematics, PD, Resolved Rate and Impedance Control of a KUKA 7-joint Manipulator

Aditya Wagh
(amw9425)

Notes. If not stated, θ represents joint parameters, and x represents end effector parameters. All plots are provided in the jupyter notebook in the interest of page limit.

Question 1

Part A – Inverse Kinematics

Inverse kinematics is the problem of solving for the joint configuration, given a desired position to be reached by the robot. It's an iterative process, where we update an initial joint configuration guess until the error between the desired position and position obtained after forward kinematics at each step is minimal. My implementation of inverse kinematic is as follows:

The function inverse kinematics takes has the following parameters.

- `desired_position` - The desired position of the end effector.
- `error` - The threshold for the error in computation.
- `step` - The step size used to update the value of theta at each step. It's multiplied with error in joint angles and added to current theta.
- `initial_theta` - The initial guess about the inverse kinematics. Its default value is chosen to be zero. I expect joint angles to be close to zero.
- `max_iterations` - The number of iterations before we conclude that the point is untraceable.

The function returns the following values.

- `best_theta` - The optimal value of joint positions.
- `solution_found` - Boolean denoting whether the solution.

The algorithm for computing the inverse kinematics is as follows:

- Initialize `current_theta` to `initial_theta`, that is zero.
- Initialize the `solution_found` Boolean to `False`.
- Initialize `best_theta` to `None`.
- Iterate for fixed number of iterations.
- At each iteration, compute forward kinematics with current value of joint angles.
- Following that, compute the error in position.
- If error is less than threshold error, set the `solution_found` flag to True, store current theta as the best theta, and break the loop.
- Else, increment theta using the value of current error. The incremental value of theta is calculated using the equation.

$$\Delta\theta = J^{-1}(\mathrm{x}_{des} - x_{curr})$$

- Theta is updated in steps of α

$$\theta_{i+1} = \theta_i + \alpha\Delta\theta$$

Part B – Inverse Kinematics on given joint configurations.

I found that the positions (2, 3, 4, 5, 9, 10) are within the reach of the manipulator.

Part C – Inverse Kinematics with null space term.

We need to modify the inverse kinematics function and pass an additional parameter `theta_zero` to compute the null space term. Its default value is [1, 1, -1, -1, 1, 1, 1] since we need to keep our joint positions close to [1, 1, -1, -1, 1, 1, 1], and the null space accounts for that. So, the following null space term gets added with the error value.

$$(I - J^{+}J)\vec{\theta}$$

Where $\vec{\theta} = \theta_0 - \theta_{curr}$ and $\theta_0 = [1,\ 1,\ -1,\ -1,\ 1,\ 1]$

So, the error value in the joint positions changes to

$$\Delta\theta = J^{-1}(\mathrm{x}_{des} - x_{curr}) + (I - J^{+}J)\vec{\theta}$$

Part D – Inverse Kinematics (with null space term) on given joint configurations.

In this case too, I found that the positions (2, 3, 4, 5, 9, 10) are within the reach of the manipulator.

Question 2

Part A – Joint positions using the null space inverse kinematics function.

End Effector Goal	Joint Angles
[0.7, 0.2, 0.7]	[[0.74892197], [0.77374463], [-1.16849869], [-0.85119068], [0.99962204], [0.88573792], [0.9991405]]
[0.3, 0.5, 0.9]	[[1.38424594], [0.38911807], [-0.86308894], [-0.76334322], [1.06640344], [1.16156525], [0.99922645]]

Part B – Point to Point motion function.

The point-to-point motion function takes in either **initial and final joint positions** OR **initial and final end-effector positions**, the current time step **t** and, a given time duration **T**. It then computes the desired joint or end-effector positions and velocities. This is an interpolation between **initial and final joint positions** OR **initial and final end-effector positions**. The following equations summarize the computation.

$$\theta_{des}(t) = \theta_{init} + \left(10\frac{t^3}{T^3} - 15\frac{t^4}{T^4} + 5\frac{t^5}{T^5}\right)\left(\theta_{final} - \theta_{init}\right)$$

$$\dot{\theta_{des}}(t) = \left(30\frac{t^2}{T^3} - 60\frac{t^3}{T^4} + 60\frac{t^4}{T^5}\right)\left(\theta_{final} - \theta_{init}\right)$$

Part C – Robot Controller (PD)

To design the controller for PD control of manipulator, I computed desired values of joint positions and joint values using the point-to-point motion function. From t = 0 to t = 5, our function calculates the desired joint positions & velocities when the end effector moves from initial position to first goal. Similarly, from t = 5 to t = 10, I computed the desired values.

Then I computed the required torque using the following term

$$\tau = P(\theta_{des} - \theta) + D\left(\dot{\theta_{\text{des}}} - \dot{\theta}\right)$$

Question 3

Part A – Robot Controller Function (Resolved Rate Control)

Similarly, for resolved rate control, I computed the desired values of end effector positions and velocities from t = 0 to t = 5, and t = 5 to t = 10. The initial position in this case would be the zero configuration of the manipulator i.e. [0, 0, 1.301]. Following which, I calculated the desired joint velocity using the equation

$$\dot{\theta_{des}} = J^{+}(P(x_{des} - x_{curr}) + \dot{x_{des}}) + (I - J^{+}J)(\theta_{des} - \theta_{curr})$$

where $(I - J^{+}J)(\theta_{des} - \theta_{curr})$ is the null space term.

Then, we compute the required torque using the following equation, where D is the derivative controller gain. Its value is chosen to be 4 for each joint.

$$\tau = D\left(\dot{\theta_{des}} - \dot{\theta}\right)$$

Question 4

Part A – Robot Controller Function (Impedance Control)

Similarly, for impedance control, I computed the desired values of end effector positions and velocities from t = 0 to t = 5, and t = 5 to t = 10. Again, the position at time t = 0 is [0, 0, 1.301] since it's the home configuration of the robot. While finding the required torque in this case, we need to add gravity compensation since automatic computation of gravity compensation is disabled this time.

There are two cases for damping in this controller design.

A. Damping is not allowed. If damping is not allowed, the final torque equation would be

$$\tau = K(x_{des} - x_{curr}) + d(\dot{x_{des}} - \dot{x_{curr}}) + G(\theta)$$

where $G(\theta)$ is the gravity compensation term that is computed using the function `robot_visualizer.rnea` which takes in joint position, joint velocity and joint acceleration.

B. **Damping is allowed.** If damping is allowed, we need to add an extra term to the torque equation

$$K(x_{des} - x_{curr}) + d(\dot{x_{des}} - \dot{x_{curr}}) + G(\theta) - D\dot{\theta}$$

where $\dot{\theta}$ represents the joint velocities.

Comments on damping. The controller is prone to oscillations when we turn off the damping, and thus it's beneficial to add the damping term. This makes sure out controller is stable.

Comparison of PD, resolved rate and impedance controller.

I found that the PD controller is the fastest and the most accurate of the three controllers. It's also easy to tune, since we only have two parameters to tune. I would rank the resolved rate controller after the PD controller. It showed the 2^{nd} best results, followed by the impedance controller. The resolved rate controller is the most intuitive of all since we are operating in the task space. Also, it's quite hard to find the correct parameters for the impedance controller. To summarize, my ranking for the controllers would be:

$$\text{PD} > \text{Resolved Rate} > \text{Impedance}$$