COMP 514 Assignment 3

Inverse Kinematics

Due: Oct 5

In this assignment, you will implement an inverse kinematics controller with redundancy resolution. Your will plan the end-effector positions for the robot using cubic polynomial. The final output is the joint positions of the robot.

1 Installation

1.1 ROS packages

```
> sudo apt-get install ros-noetic-ros-control
> sudo apt-get install ros-noetic-ros-controllers
```

1.2 Kinova Kortex Manipulator

- Download the kinova Kortex simulator from here
  `https://gitlab.cs.mcgill.ca/applied-robotics/robots/ros_kortex`
- Add `ros-kortex` to your catkin workspace and build it
  `> catkin build kortex_gazebo`
- Once it is complied, try
  `> roslaunch kortex_gazebo gen3.launch`

1.3 Eigen3

- Eigen is a library for handling matrices and vectors. Check if you have eigen installed
  `> whereis eigen3`
- If you don't see eigen in your directory
  `> sudo apt-get install libeigen3-dev`
- Pinocchio is a library that computes the kinematics and dynamics parameters for you. Install pinocchio by
  `> sudo apt-get install ros-noetic-pinocchio`

1.4 Example

Download the example code. Feel free to reuse it.
`https://gitlab.cs.mcgill.ca/applied-robotics/examples/robot_model`

- CMakeLists.txt: eigen and pinocchio are not standard ROS library. Therefore, they are specified differently in CMakeLists.txt. The minimum change is the following:

```
find_package(catkin REQUIRED COMPONENTS
    roscpp
    sensor_msgs
    # put whatever more packages you need
)

find_package(Eigen3 REQUIRED)                      # find eigen
find_package(pinocchio REQUIRED)                   # find pinocchio
add_definitions("-DBOOST_MPL_LIMIT_LIST_SIZE=30")  # something for pinocchio
```

- test-pinocchio.cpp: examples on how to use pinocchio
- test-eigen.cpp: examples on how to use eigen

2 How to submit

Create a new ROS package under your Gitlab repository `robotic-coursework-f2022`. Your new ROS package should be called `inverse_kinematic_controller`. When you are ready to submit, push your change to `robotic-coursework-f2022`. Do not create a new repository.

3 Steps

- Read the joint feedback by subscribing to the topic `/gen3/joint_states`. Do not read from gazebo.
- Plan your trajectory for the robot end-effector positions using cubic-polynomial. You can ignore the orientations in this assignment. You are more than welcome to re-use your A2. The desired position is defined by the user through the same service `move_robot`.
- Compute the end-effector positions and Jacobian matrix using `pinocchio`.
- Compute the joint velocity using inverse kinematics. There are many different ways to use planner and inverse kinematics. Here are two options:

  1.

  $$\begin{aligned} \dot{\mathbf{x}}^{ref} &= cubic\_polynomial\_planner(t) && \text{get reference velocity from cubic polynomial} \\ \dot{\mathbf{q}}^{ref} &= \mathbf{J}^{\dagger}\dot{\mathbf{x}}^{ref} && \text{calculate reference joint velocity using inverse kinematics} \end{aligned}$$

  2.

  $$\begin{aligned} \mathbf{x}^{ref} &= cubic\_polynomial\_planner(t) && \text{get reference position from cubic polynomial} \\ \dot{\mathbf{x}}^{ref} &= \frac{\mathbf{x}^{ref} - \mathbf{x}^{fbk}}{\delta t} && \text{calculate reference velocity} \\ \dot{\mathbf{q}}^{ref} &= \mathbf{J}^{\dagger}\dot{\mathbf{x}}^{ref} && \text{calculate reference joint velocity using inverse kinematics} \end{aligned}$$

  Which one is better?
- Add the redundancy resolution. To begin, I suggest that you start without redundancy resolution and only add it if the original inverse kinematics is working.
- Publish joint positions you calculated to `/gen3/joint_group_position_controller/command`

4 Useful tips

- In your header files, if you need to include both `pinocchio` and `ros` header files, put pinocchio first. e.g.,

  ```
  #include <pinocchio/multibody/model.hpp>
  ⋮
  #include <ros/ros.h>
  ```
- If you open gen3.launch, you should see

  ```
  <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model" respawn="false"
            output="screen"
            args="-urdf -param robot_description -model $(arg robot_name)
  ```

```
-x $(arg x0) -y $(arg y0) -z $(arg z0)
-robot_namespace $(arg robot_name)
-J joint_1 1.57
-J joint_2 0.35
-J joint_3 3.14
-J joint_4 -2.00
-J joint_5 0
-J joint_6 -1.00
-J joint_7 1.57"/>
```

This node launch the gen3 manipulator with initial configurations. (i.e., if you change those numbers, the joint positions will be different when you launch gazebo.

5 Evaluation

We will test your implementations as follow:

1. Open a terminal and run
   `> roslaunch kortex_gazebo gen3.launch`
2. Open another terminal and run
   `> roslaunch inverse_kinematic_controller a3.launch`
3. Open another terminal and run
   `> rosservice call /cubic_polynomial_planner/move_robot 0.5 0.2 0.3 5`
   The end-effector should move to position (0.5 0.2 0.3) in 5 seconds. We will try a few different combinations of inputs. To send negative positions, you can do
   `> rosservice call /cubic_polynomial_planner/move_robot '{x: -0.3, y: 0.0, z: 0.3, T: 5.0}'`

5.1 Marks (10 points total)

(2 points) Correct file, node, and package names
(2 points) Read and write to the correct topics
(2 points) Calculate the forward kinematic and jacobian
(2 points) Implement inverse kinematics with redundancy resolution
(2 points) End-effector reaches the target