Assignment 5
Orientation Control
Nov 2

In this assignment, you will add orientation control to your controller. You can choose any controllers and planners from the previous assignments.

1 Steps

1.1 Warning Messages

Write a ROS package that detects

- velocity limits (if you are doing position control)
- torque limits (if you are doing torque control)
- singularities

If the soft E-stop is triggered, display ROS_WARN message This component is not marked, but highly recommanded to have.

1.2 Highlevel Controller

Create a new ROS package called `highlevel_controller`.

- create a new launch files called `a5.launch`, which launches your planner, controller, and gazebo simulation of the arm. You are allowed to use either planner (potential field or cubic polynomial) and either controller (inverse kinematics or inverse dynamics). This can be done by using `include file` in your launch file), e.g.,

```
<include file="$(find kortex_gazebo)/launch/gen3.launch" >
    <!-- pass arguments here if you want -->
</include>
```

- create a new yaml files called `config/default-target.yaml`, which contains the default end-effector target positions. You will read this default target positions for your planner.

1.3 Task-space Orientation Controller

There are three ways to represent orientations: (1) euler angles, (2) rotation matrix, (3) quaternion. One main difference is how you handle $\mathbf{x}^{ref} - \mathbf{x}^{fbk}$. There are many ways to implement this; e.g., directly compute the angular error between two euler angles, or compute the pose error between two homogeneous transformation matrix and then convert it to a vector.

1.3.1 Euler Angles

If the reference orientation is $= (0, 0, 0)$

$$\boldsymbol{\theta}^{ref} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

The orientation error is calculated like other assignments

$$\boldsymbol{\theta}^{err} = \boldsymbol{\theta}^{ref} - \boldsymbol{\theta}^{fbk}$$

```
#include <pinocchio/math/rpy.hpp>

/* Variables */
```

```
Eigen::VectorXd pose_err_ ;            // pose error vector (6D) in global frame

/* Useful functions */

data_.oMi[hand_id_]                    // feedback pose
data_.oMi[hand_id_].translation()      // feedback translation
data_.oMi[hand_id_].rotation()         // feedback orientation
Eigen::MatrixXd R   = pinocchio::rpy::rpyToMatrix(r,p,y)  // convert euler to rotation
Eigen::MatrixXd rpy = pinocchio::rpy::matrixToRpy(R)      // convert rotation to euler

/* calculations */

// get the feedback translation
pose_fbk_.head<3>() = data_.oMi[hand_id_].translation() ;

// convert feedback orientations into euler angles
pose_fbk_.tail<3>() = pinocchio::rpy::matrixToRpy (data_.oMi[hand_id_].rotation()) ;

// pose error
pose_err_ = pose_ref_ - pose_fbk_ ;
```

1.3.2 Rotation Matrix

If the reference orientation is $= (0, 0, 0)$, the equivalent rotation matrix is

$$\mathbf{R}^{ref} = \mathbf{I}$$

The difference between two homogeneous transformation matrix is

$$\mathbf{R}^{err} = (\mathbf{R}^{fbk})^{-1}\mathbf{R}^{ref}$$

Here is how to do it with pinocchio:

```
/* Variables */
Eigen::VectorXd orientation_ref ;      // reference orientations (from yaml)

pinocchio::SE3  pose_err_mat_   ;      // position error in matrix
Eigen::VectorXd pose_err_vec_local_ ;  // pose error vector in local frame
Eigen::VectorXd pose_err_ ;            // pose error vector in global frame

data_.oMi[hand_id_]                    // feedback pose

/* calculations */

// calculating translation error is the same as other assignments
pose_err_.head<3>() = ...

// compute the error between two homogeneous transformation matrix
pose_err_mat_       = data_.oMi[hand_id_].inverse() * pose_ref_ ;

// convert it to euler angles in local farme
pose_err_vec_local_ = pinocchio::log6(pose_err_mat_).toVector() ;
```

```
// convert the euler angles from local frame to world frame
pose_err_.tail<3>() = data_.oMi[hand_id_].rotation() * pose_err_vec_local_.tail<3>() ;
```

1.4 Switch to "Release" Mode

Catkin allows you to run in "Debug Mode" or in "Release Mode". Once you are certain that your code is bug-free, you can switch to "Release Mode" by running

```
catkin config --cmake-args -DCMAKE\_BUILD\_TYPE=Release
catkin build xxxxxxx
```

This will significantly boost the performance of your code. If you need to switch back to "Debug Mode", you can do it by

```
catkin config --cmake-args -DCMAKE\_BUILD\_TYPE=Debug
catkin build xxxxxxx
```

2 Evaluation

2.1 Test

We will test your implementations as follow:

1. Open a terminal and run
   `> roslaunch highlevel_controller a5.launch`
   Once Gazebo is running, your robot should move toward the end-effector position specified in `default-targe.yaml`.
2. We will change the values in this yaml file and restart.

2.2 Marking Scheme (10 points total)

(5 points) Correct orientations
(3 points) Arrives at the target specified in `default-target.yaml`
(2 points) Arrives at the target for random positions