Mobile Manipulation Capstone

ME 449 YouBot KUKA Manipulation.

INTRODUCTION

This project contains Python 3 scripts that plans a trajectory for the end-effector of the youBot mobile manipulator, performs odometry as the chassis moves, and performs feedback control to drive the youBot to pick up a block at a specified position and deliver it to the desired location in the CoppeliaSim simulation.

RESULTS

The result contains three scenarios which are:

- Best case: well-tuned controller by adjusting Ki, proportional integral

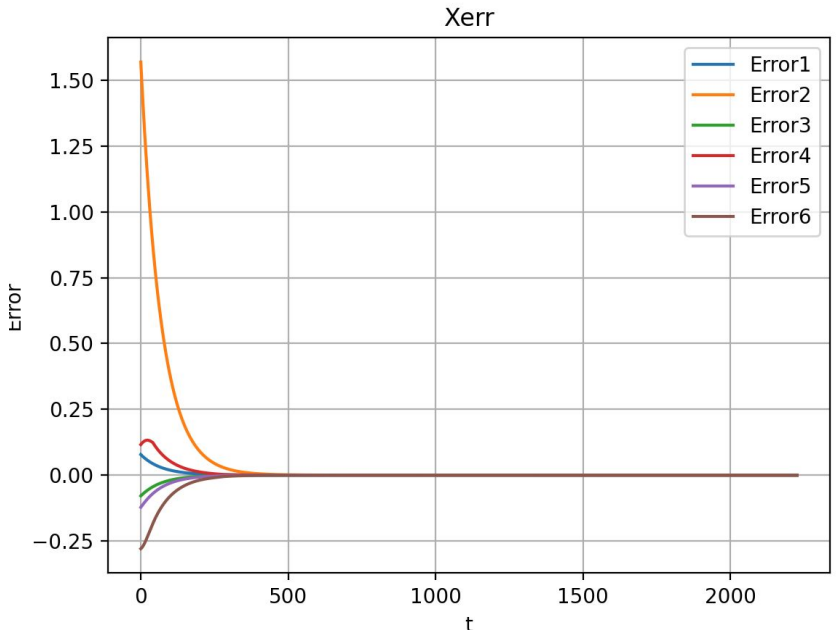

Figure 1: Best Case Xerr plot

- Overshoot: controller that experiences overshoot

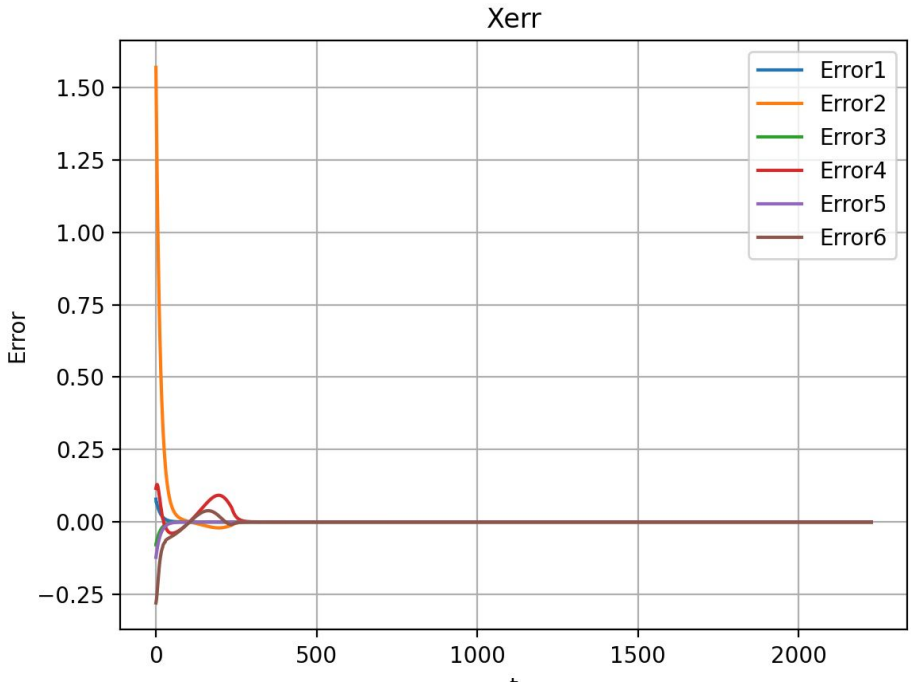

Figure 2: Overshoot Xerr plot

- New Task: well-turned controller with different initial and final block configurations from the above cases.

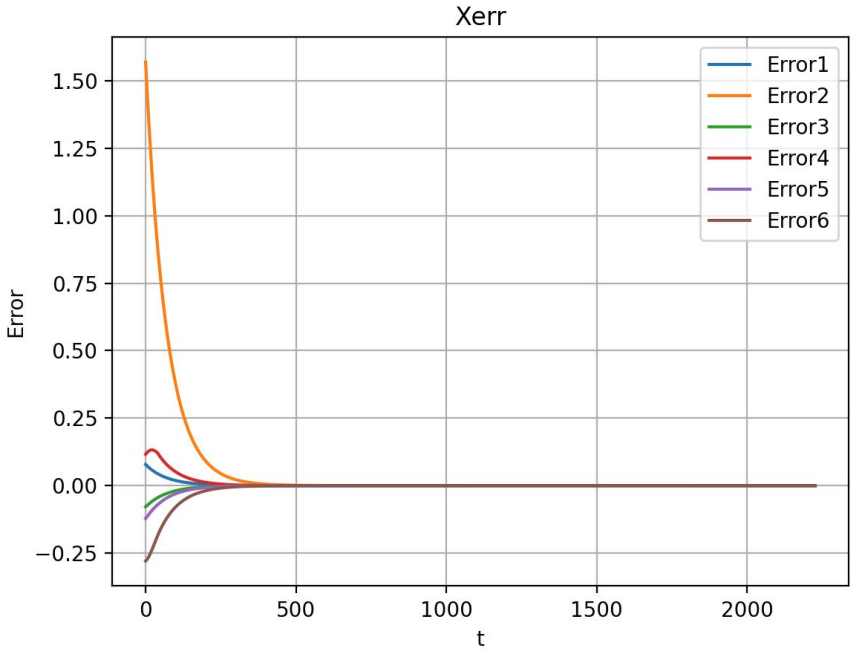

Figure 3: New Task Xerr plot

DISCUSSION

Important Points Neglected

In this project, I did not consider implemented singularity avoidance or joint limit avoidance since the robot executes a simple task. However, a real robot in more complicated tasks may experience self-collision while a robot arm ignores and goes through the chassis in the simulation.

Future Improvements

For better performance of the robot, a motion planning can be implemented that will help the robot to avoid obstacles. In addition, applying and implementing SLAM will allow the robot to navigate more detailed and complicated environments rather than a simple planar field.

CODES

There are a total 4 Python scripts which are milestone1:NextState, milestone2:TrajectoryGenerator, milestone3:FeedforwardControl, and Run. Three milestones define functions that manipulate the configuration of the robot, and the Run script combines those three functions, plot Xerr, and generates a csv file that can be used in the CoppeliaSim simulation. All codes can be found in the `code` directory.

Milestone 1: NextState

The main function in the simulator, called NextState takes

Inputs:

- 12-vector representing the current configuration of the robot
- 9-vector of controls indicating the arm joint speeds
- timestep Δt
- positive real value indicating the maximum angular speed of the arm joints and the wheels.

Outputs:

- new arm joint angles = (old arm joint angles) + (joint speeds) * Δt
- new wheel angles = (old wheel angles) + (wheel speeds) * Δt
- new chassis configuration is obtained from odometry

Milestone 2: TrajectoryGenerator

TrajectoryGenerator generates the reference trajectory for the end-effector frame {e}. It takes

Inputs:

- The initial configuration of the end-effector in the reference trajectory: $T_{se,\text{initial}}$.
- The cube's initial configuration: $T_{sc,\text{initial}}$.
- The cube's desired final configuration: $T_{sc,\text{final}}$.
- The end-effector's configuration relative to the cube when it is grasping the cube: $T_{ce,\text{grasp}}$.
- The end-effector's standoff configuration above the cube, before and after grasping, relative to the cube
- The number of trajectory reference configurations per dt seconds

Output:

- A representation of the *N* configurations of the end-effector along the entire concatenated eight-segment reference trajectory.

Milestone 3: FeedbackControl

This function calculates the kinematic task-space feedforward plus feedback control law taking

Inputs:

- The current actual end-effector configuration X (also written Tse).
- he current end-effector reference configuration Xd (i.e., Tse,d).
- The end-effector reference configuration at the next time step in the reference trajectory, Xd,next
- The PI gain matrices Kp and Ki.
- The timestep Δt between reference trajectory configurations.

Output:

- The commanded end-effector twist expressed in the end-effector frame {e}.

Run

This script combines three functions above and executes a function called `run` which plots Xerr and generates a csv file for the CoppeliaSim animation and log file.