README

November 21, 2021

1 Final Project Robotics Manipulation: Milestone 2

By: *Ryan King-Shepard*

1.1 Contents

The following contents of the assignment are:

1. code/
    - Directory of the code files
        - `mile2.py`: Contains the TrajectoryGenerator function used to craft the trajectory for milestone2
        - `run.py`: Main executable for milestone 2
2. output.csv
    - Example csv used to produce coppelia sim animation with scene 8
3. KingShepard_Ryan_milestone2.mp4
    - Video of Coppelia Sim Animation
4. README.pdf
    - TIS I, THE DOCUMENT BEFORE YOU!

1.2 Code Dependencies

The code for milestone2 depends on the modern_robotics package, numpy, argparse and csv. The Python Standard Libraries should already contain argparse and csv; if they are not present, use `pip install [package]` to add them to your python libs. You will need to install modern_robotics(instructions here) and numpy(instructions here)

1.3 Code Execution

The main execution script for this milestone is `rain.py`. This file interfaces with the user for arguments and calls the TrajectoryGenerator in `mile2.py` to produce the outputs

```
./run.py -h
usage: run.py [-h] [--ani-file ANI_FILE] [--only_mile1] [--only_mile2]
              [--only_mile3]

Calculate and output csv for final project animations

optional arguments:
  -h, --help           show this help message and exit
  --ani-file ANI_FILE  output csv for animations. Currently only supports
                       milestone 2. Defaults to "ani_file.csv"
  --only_mile1         produce only the outputs for milestone 1
  --only_mile2         produce only the outputs for milestone 2
  --only_mile3         produce only the outputs for milestone 3
```

To produce a file called `output.csv` with the transformation matrices and gripper commands run:

```
python3 run.py --only_mile2 --ani-file 'output.csv'
```

1.4 Solution Summary

The goal of milestone 2 is to craft a series of end-effector trajectories to transport a 5cm3 cube from it's initial orientation to the goal orientation. This is achieved by calculating four end-effector poses and forming trajectories between them. The four poses are the robot's standoff position for when its about to pick up and drop off the cube, the initial pose of the cube, and the desired pose for the cube. We define these poses as matrices in the SE(3) group and are given the initial position of the end-effector in the space frame(Tse), the initial cube position in the space frame (Tsc_init) and the desired cube position(Tsc_goal).

I chose matrices Tce_standoff and Tce_grasp as representation of the end-effector's poses in the cube's frame when the robot is in the standoff pose or about to grab/release the cube.

```
# This is defined in `run.py` as well
T_ce_standoff = [[-sqrt(2)/2,  0,  sqrt(2)/2,  0.0085]
                 [0,           1,          0,       0]
                 [-sqrt(2)/2,  0, -sqrt(2)/2,     .03]
                 [0,           0,          0,       1]]

T_ce_grasp = [[-sqrt(2)/2,  0,  sqrt(2)/2,  0.0085]
              [0,           1,          0,       0]
              [-sqrt(2)/2,  0, -sqrt(2)/2,       0]
              [0,           0,          0,       1]]
```

We can then multiply the matrices to grab the four positions:

- Tsc_init*Tce_standoff = standoff pose to pick up cube
- Tsc_init*Tce_grasp = pose to pick up cube
- Tsc_goal*Tce_standoff = standoff pose to drop off cube
- Tsc_goal*Tce_grasp = pose to drop off cube

For trajectory planning, I used ScrewTrajectory planning for getting the robot's end-effector into the standoff position since these movement involved a rotation and translation of the end-effector's frame. Cartesian Trajectory was used to plan the tiny movements of picking up or placing the cube on the ground. These movements only involve a translation of the end-effector's frame so the plan compartmentalized to avoid any unnecessary rotations.