ME 449 Capstone Project README

Submission File Structure:

The file structure for my submission can be seen below. You are reading the README.pdf in the main folder. Then, there is a folder containing all the code for this project, written in a fully commented and well-organized python notebook. Code for all three sections is found within the code notebook, and the full final project code can be found in the last cell of this notebook. I also consolidated the code just for the final run (including all milestone functions) into a single code.py file for the graders convenience, however, I used the notebook and found it convenient. The results requested are within the results folder, organized into the best, newTask, and overshoot folders. To run my code for each of these sections, simply copy the entry from within the log.txt file and paste into the bottom of the python notebook. Then, the trajectory will be stored into the q_final.csv file, the X_err data will be stored into the X_err.csv file, and the X_err data will be plotted. To run in CopelliaSim simply copy the path to the q_final.csv file and paste into Scene6.

```
JENZ_MICHAEL_FINALPROJECT/
│
├──README.pdf
├── code/
│   ├── code.ipynb
│   ├── code.py
│   ├── configuration.csv
│   ├── q_final.csv
│   ├── reference_trajectory.csv
│   ├── X_err.csv
├── results/
│   ├── best/
│   │   ├── best_log.txt
│   │   ├── best_recording.mp4
│   │   ├── best_trajectory.csv
│   │   ├── best_X_err.csv
│   │   ├── best.png
│   │   └── README.txt
│   ├── newTask/
│   │   ├── newTask_log.txt
│   │   ├── newTask_recording.mp4
│   │   ├── newTask_trajectory.csv
│   │   ├── newTask_X_err.csv
│   │   ├── newTask.png
│   │   └── README.txt
│   └── overshoot/
│       ├── overshoot_joint_limit_comparison_recording.mp4
│       ├── overshoot_log.txt
│       ├── overshoot_recording.mp4
│       ├── overshoot_trajectory.csv
│       ├── overshoot_X_err.csv
│       ├── overshoot.png
│       └── README.txt
```

Software Overview:

The software in this project can perform kinematic task-space feedback control of a mobile manipulator using PI feedback control. The software can be divided generally into three main functions. The first function TrajectoryGenerator defines a desired trajectory for the robot to follow. This trajectory is defined by a set of key points in a pick and place task, and outputs the desired trajectory. Next, the function FeedbackControl takes in the current position of the robot, the desired position, and the next desired position of the robot. Using simple PI feedback loop it generates joint speeds that minimize the twist error of the robot. Finally, the function NextStep uses odometry and a single euler step to calculate the next position of the robot. These three functions work together to simulate a robot control system, and output .csv files of the configuration during the task that can be played on CopelliaSim for visualization.

Joint Limits:

One challenge presented was to use joint limits to avoid self-collisions and singularities. I decided to focus on using the joint limits to prevent self-collisions after testing revealed that some joint limits that prevented singularities posed more serious problems for the accomplishment of the pick and place task. So, I chose joint limits such that the arm does not bend backwards on itself to pick up the block. Meaning, I restricted the joint to around a -2 to 2 radian range (or smaller for other joints). This improved performance since the joint only operated in front of the chassis. During runs without the joint limits the mobile manipulator sometimes picked and placed the block inside of the chassis. The effect of my improvements can be seen in the overshoot directory where severe self-collisions occur without the intervention of my joint limits.

Code Changes and Helper Functions:

I changed the inputs and outputs to the FeedbackControl() function quite a bit from the suggested ones in the project description. This was for both practical reasons and for convenience. First, I had the function input and output the integrated error, so the value was carried through between iterations. Secondly, I changed the input of the current state to be the configuration rather than the transformation matrix X. I found it to be more elegant to input the current configuration (similar to the output of NextStep and the input to the .csv file) and then have this data converted to the X transformation matrix using my helper function get_transformation(). I also added a helper function get_jacobian() which retrieved the mobile manipulator Jacobian for each run of FeedbackControl(). Another helper function testJointLimits() takes in the current next step configuration and identifies joints that violate the joint limits.This data is then used to modify the mobile manipulator Jacobian to prevent such violations and recalculate the joint speeds before finishing a run of FeedbackControl(). One last helper function odometry() performs all the odometry for the NextStep() function.

Conclusion:

This project was an interesting culmination of the work in ME 449. I would like to thank Prof. Lynch and the TA’s for an intellectually stimulating quarter.