MAE 204 FINAL PROJECT

README

March 19th, 2020

Venkatesh Prasad Venkataramanan
PID : A53318036

DESCRIPTION

The objective of the project is to design a planner and controller for the KUKA YouBot (consisting of a mobile base and a 5-joint manipulator) to be able to pick and place a cube.

CODE ORGANISATION

main.m – Main code which has to be run

calcJac.m – Calculates Jacobian and returns its inverse

next_State.m – Calculates next robot states.

reference_trajectory.m - Calculates the reference trajectory

feedbackcontrol.m - Calculatesthe commanded twist.

WORKFLOW

Milestone 1

In Milestone 1 I implemented the youBot simulator, where in, we develop an odometry based motion model to be able to calculate the next position of the robot.

This is required in order to find the next state of the robot, in order to find the error from the reference trajectory, which eventually has to be driven down to 0.

Milestone 2

In milestone 2, I implemented a trajectory planner. In total, the robot has to perform 8 trajectory segments. I implemented this using the Screw Trajectory function which was provided along with the Modern Robotics Library.

This is required for the final step as this gives us the desired trajectory which helps drive the controller.

Milestone 3

Milestone 3 was about implementing the feedforward control. Here, I wrote a function to return the commanded twist for the robot, taking the Ki and Kp values, along with the actual and desired robot states. This was then used to calculate the commanded wheel and joint velocities using the pseudo-inverse of the jacobian. This would then drive the robot along the desired trajectory.

I first tested just the feedforward part, without any proportional and integral error and the robot was able to pick up the cube(although with some difficulty), but there was a considerable offset in setting down the cube at the final position. The y-coordinate was off by 20% error. This was fixed using the PID controller which I designed in the final part of the project.

Putting it all together

In order to put it all together, we need to do a bunch of initialisations.

Setting Kp, Ki values

Setting Kp,Ki values was an experimental procedure. I achieved near perfect control with the Kp,Ki set as 10 and 7 respectively. I also chose a set of bas Kp,Ki values for the overshoot results.

Other initialisations

Other initialisations were set exactly as the website described. I set initial configuration with considerable error and played around with the controls.

I gave error of about 30 degrees in the chassis phi, and a positional error of 0.2m in all wheels. I also set the initial configuration of the robot wheels differently.

I then initialise the time for each trajectory segment. I set it to be 4 seconds for each segment. This gives beautiful results. I also initialise my error matrix, and my configuration matrix.

Main workflow

The sequence of operations we need to follow is thus:

Initialise our current state with the variations as mentioned above. I then calculate the reference trajectory as performed in Milestone 2. I made a slight change in this regard. Earlier, when I went in to grab the cube, my gripper used to be perfectly horizontal to the cube. This time, I made it come in from above, but at an angle of 45 degrees instead of 90 degrees, in order to simplify the pick and place operation. This gave rise to a vastly smoother trajectory. This was inspired by the sample video posted on the website by the course instructors.

I then initialise my X_desired and X_desired_next from the reference trajectory that has been calculated. I also calculate the X_actual using the current state, which is at an offset. I am now all set to get inside the loop.

Once inside the loop, I use all my initialisations and send it to the feedback control function. This gives me the commanded twist. This twist is then converted to the requisite wheel and joint velocity using the Jacobian of the end effector which is calculated each time in the loop using the jacobian function provided in the modern robotics library.

After I extract the control inputs, I then pass it to the next state function to be able to calculate the next state. This next state will be then again be fed to the feedback control function. We also update the desired state and the next desired state. This loop then continues for the entire range of motion.

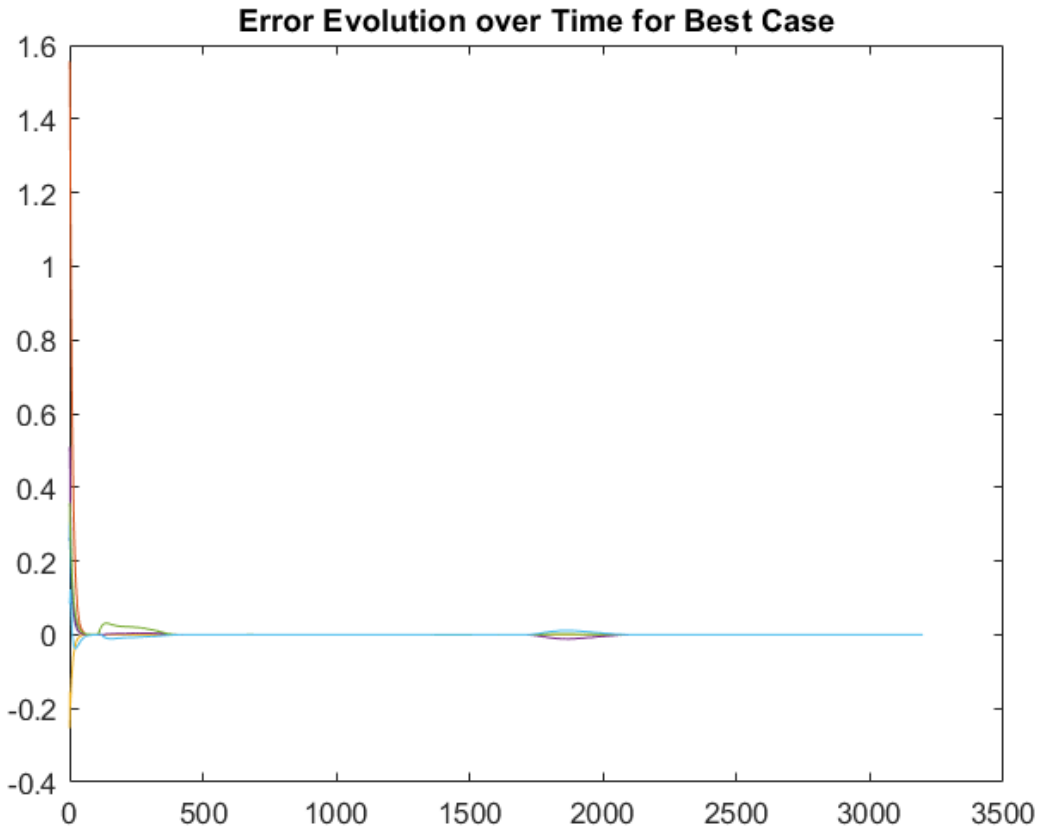
Final Step

The final step is to move in and write out the CSV files for error and configurations. Also, I draw error plots to get me results.

Results

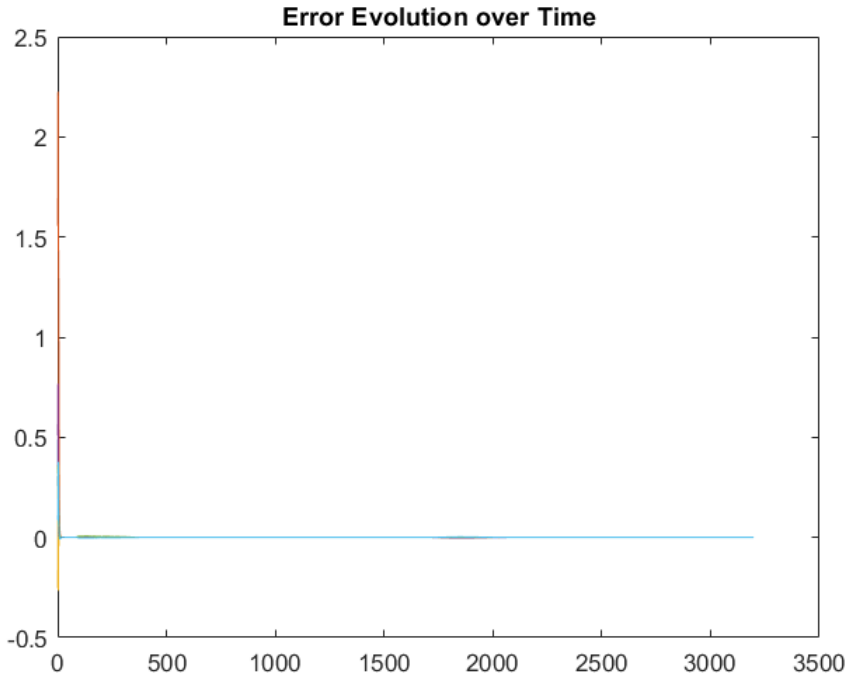
Here, I present the evolution of errors in all three cases i.e, best, next state and overshoot.

BEST(Kp = 10, Ki = 7)



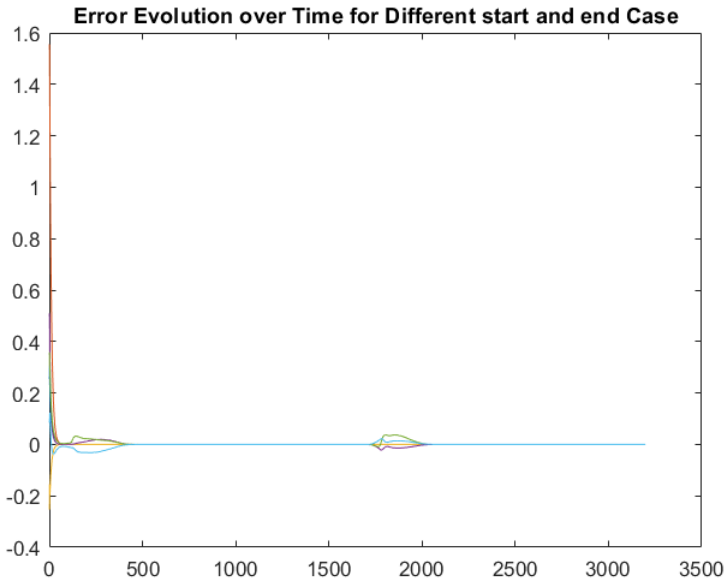
This is the error plot with my best controller tuning parameters. There is actually very less overshoot and achieves error of almost zero. We use the exact same start and end locations for the cube.

Overshoot(Kp = 50, Ki = 7)



In this image, as you can notice, there is a huge overshoot. The controller goes completely in the opposite side and then converges rather quickly. It is extremely jerky, and really bad and unrealistic for the robot.

NEXT STATE(Best Kp,Ki values but different start and end locations)



You can see that the controller performs really well in this case also. The simulation shows that the controller is able to effectively control the robot along the desired trajectory.

Discussion

The enhancement I tried out was experimenting with singularity avoidance. I experimented with various values for the pseudo-inverse. I finally settled on 1e-3 to get me the smoothest, best results.

Another thing that I observed was that Screw Trajectory Planner and Cartesian Trajectory Planner gave vastly different results. In certain scenarios, cartesian trajectory trumped screw trajectory in giving amazing results. But cartesian tends to fail in many cases, whereas screw gives results for any kind of scenario. To be on the safer side, I chose to go ahead with screw trajectory.