Electronics Design Lab Report #2

Introduction

This week I have learned more about ROS topics and performance of program details since I was faced with strange bugs while was doing my homework. Additionally, I read about the theory of controlling the vehicle that was provided as reference in the lecture note, watched the video about PID controller and read many sources to know more details about it. Finally, I tried to implement all these techniques in the assignment and consequently learnt a lot about parameters tuning. Code for P, PI, Pure pursuit and Stanley method was made and some comments about each case and explanations will be provided here; and lastly, questions will be answered based on my new experience. Note that I made all the codes in the same file, so to switch from one to other you need to uncomment some of the parts which I underlined in the comments. Default is P controller.

Explanation

```
        # P controller
        kp_y   = 0.3 # P gain w.r.t. cross track error
        kp_yaw = 0.3 # P gain w.r.t. yaw error

        # PI controller
        # add new error values to history
        delta_y.append(error_y)
        delta_yaw.append(error_yaw)
        # define proportional constants
        kpi_y = 0 # 0.05 # uncomment for pi controller
        kpi_yaw = 0 # 0.05 # uncomment for pi controller
        # define integral terms in terms of sum
        iterm_y = sum(delta_y)
        iterm_yaw = sum(delta_yaw)
        # the final result will be
        steer = kp_y*error_y + kp_yaw*error_yaw + kpi_y*iterm_y +
kpi_y*iterm_yaw
```

```
        # Pure pursuit method
        L = 0.5 # length of car
        l = 3
        # uncomment line below to see its implementation
        #steer = np.arctan(2*L*math.sin(error_yaw)/l)

        # Stanley method
        k = 1
        # uncomment lines below for Stanley method
        #if (v == 0):
        #       steer = 0
        #else:
        #       steer = error_yaw + np.arctan(error_y/v)
```

```
        """
        # for P controller
        kp_v = 0.7
        # for PI controller
        kpi_v = 0 #0.03 # uncomment for pi controller
        delta_v.append(error_v)
        iterm_v = sum(delta_v)

        return kp_v * error_v + kpi_v*iterm_v
```

Making P controller was easy as the code was already written, the goal was to find the optimal constants for each term which was a challenge. For acceleration term, velocity quickly reaches the maximum so it becomes constant after a while. The constant just decides how fast, however great constant makes it fluctuates a lot and skip its turns by making other errors great, so I found the value when this fluctuation does not happen and to be it large enough, that's why kp_v = 0.7. For constants in error_y and error_yaw it was more complicated, for large values it again fluctuated in turns and in the path where waypoints are far from each other, for small it did not merge with waypoints, or merged very slowly. So I made them 0.3 each which showed a good performance.

In PI controller integral was replaced with sum as it is not the continuous program, but instead the simulation where the motion is performed by each step dt. I created the arrays for each error_y, error_yaw and error_v as global variables and store all these values as a history, than every time the integral term is the sum of all the history which simulates the actual integral. The performance of PI controller is worse than P as velocity changes is strange manner even was decelerating, fluctuations about the waypoints were very often and sometimes car started making circular motion near the turns and then continued its way. That's why I choose very small constant terms when the integral part is noticeable, but motion is still normal.

I made the Pure Pursuit method by following the equation from lecture notes, the length of car was measured in rviz by measurement option and should contain some error, but since the look ahead distance l is tuning parameter the error in L does not matter much. Next, the tuning of l begun, when I made it 1 it oscillated about its path, in very large l it did not merge to waypoints. I choose it 3 and it goes well, but after some time finds some equilibrium at some distance from desired path. Anyway, its behavior at turns is still good.

In Stanley method, I used approximations as variables should be calculated from front gear, but the values taken from nodes possibly from the center of vehicle. There was another challenge as initial velocity is 0 which gave error in controller, that was solved by simple if else statements. Parameter k is tuned as 1 and was good enough. The performance is very satisfiable as it moves very smoothly even at turns.

Discussion

The primary difference between the P and PI controller is that PI has some history, i.e. it considers the previous states of the vehicle. This led to the high sensibility in the changes in the road and therefore more smooth turns (as velocity is also sensible to turns) should result. However, this sensibility created numerous fluctuations and oscillations about the desired path and sometimes the vehicle behaved abnormally which may be either incorrect code, bad parameter tuning or just simplicity of the situation

did not need such careful approach. Therefore, the whole performance of P controller was better in this example.

The large cross track error was primary because when the car moves in the corner is performs some part of circular motion and large velocity at can cause some drifting or very high acceleration, in other words for good turning, car should either lower its velocity especially when the radius of corner is high or start turning more carefully before the corner. In our model the velocity reaches and maintains same value during all path. To minimize it one way is to control velocity value and lower it at corners. Other technique is use look ahead distance, when you predict the turn before making is and start already turning the velocity direction so car turns more smoothly and avoids unnecessary drift from desired path. That's why there is low error in Pure Pursuit model. Also, good approach is in Stanley model as steering is a function of velocity, so it notices when velocity value is high and takes it into the account while making the turn, at high velocity it makes smoother angle change and therefore should be better at sharp turning point.

Thank you