Vehicle Following Task Memo – Phillip Arab:

Overview of Task:

As per Vehicle Following task assigned by RMAL, I designed a PID controller via C++ script to govern the motion of a drone following a car. See "TMU_PID.cpp" for the source code, and see "TMU Task Results.xlsx" for the results.

Assumptions:

- Assumed drone height does not changing, and is not part of the desired control.
- Assumed drone should always face car itself, and not the orientation of the car. Therefore no lateral movement, only forward/backward motion in the horizontal (XY) plane and rotation in XY plane about the Z (vertical) axis.
- Assumed, on each timestep, drone does all its rotating before its forward motion. The smaller the change over each timestep, the more reasonable this assumption.
- Assumed initial conditions for entire system for sake of producing general controller and specific output.

Solution:

As per the assumptions listed, my solution to this problem was to create a controller to move and rotate the drone in the XY (horizontal) plane towards the car. This effectively simplifies the problem to 2 dimensions. I wrote two PID controller functions, one for rotation and one for translation, which were each called in my main function. My main function initiates all the necessary variables, and only passes the error values to the PID controller functions. These PID controller functions contain within them the control gains (kp, ki, kd) to be tuned.

Calculating all poses and errors, as well as calling the PID controller functions to produce output velocities, all happens in a loop in my main function where each iteration of the loop represents a timestep for the controller. As well in each loop, the coordinates and time are appended to arrays for post processing/plotting.

In excel each (x,y) coordinate was plotted on a 2D scatterplot. To indicate the third dimension, time, a gradient line is shown where blue represent start time and red represents the end time of the simulation.

The PID gains were tuned such that the output results show the drone reaching, and continuing to closely follow the car. This is achieved for a straight line motion of the car, and with the car executing a turn. It's worth noting that both of these results shown are with the same PID gains of the drone controller.

Conclusions:

A few conclusions were drawn from this design, most significantly from the PID tuning. As the controller tracks a moving target, steady state error has generally low impact. The same is true of overshoot due to the dynamic and less precise nature of this problem. Because of this, for optimal performance the proportional gain was set to be the highest gain for both translational and rotational controllers. It is also worth noting that the rotational gains were set generally higher than the translational gains. This is because in the 2D plane, if the drone is pointing away from its target, any forward motion is only increasing error. The optimal design was to have the drone's rotation very responsive and fast, and the forward motion to be relatively slower.