 # Control and Trajectory Tracking for Autonomous Vehicle

 # Proportional-Integral-Derivative (PID)

In this project I have implemented a PID controller to control the steering and throttle of a vehicle in Carla simulator. The trajectory is given by the path planner and PID controller is responsible for following this trajectory. Implementation was done by first creating a PID class, creating PID objects for steering and throttle and finally tuning the proportional, derivative and integral parameters so as to follow the trajectory given by the path planner. The controller was tested in Carla simulator and the outputs were plotted using a python script.

Installation

Run the following commands to install the starter code in the Udacity Workspace:

Clone the repository:

`git clone https://github.com/udacity/nd013-c6-control-starter.git`

Run Carla Simulator

Open new window

* `su - student`

// Will say permission denied, ignore and continue

* `cd /opt/carla-simulator/`

* `SDL_VIDEODRIVER=offscreen ./CarlaUE4.sh -opengl`

Compile and Run the Controller

Open new window

* `cd nd013-c6-control-starter/project`

* `./install-ubuntu.sh`

* `cd pid_controller/`

* `rm -rf rpclib`

* `git clone https://github.com/rpclib/rpclib.git`

* `cmake .`

* `make` (This last command compiles your c++ code, run it after every change in your code)

Testing

To test your installation run the following commands.

* `cd nd013-c6-control-starter/project`

* `./run_main_pid.sh`

This will silently fail `ctrl + C` to stop

* `./run_main_pid.sh` (again)

Go to desktop mode to see CARLA

If error bind is already in use, or address already being used

* `ps -aux | grep carla`

* `kill id`

Project Instructions

In the previous project you built a path planner for the autonomous vehicle. Now you will build the steer and throttle controller so that the car follows the trajectory.

You will design and run the a PID controller as described in the previous course.

In the directory [/pid_controller](https://github.com/udacity/nd013-c6-control-starter/tree/master/project/pid_controller) you will find the files

[pid_controller.cpp](https://github.com/udacity/nd013-c6-control-starter/blob/master/project/pid_controller/pid_controller.cpp) and [pid_controller.h](https://github.com/udacity/nd013-c6-control-starter/blob/master/project/pid_controller/pid_controller.h). This is where you will code your pid controller.

The function pid is called in [main.cpp](https://github.com/udacity/nd013-c6-control-starter/blob/master/project/pid_controller/main.cpp).

Step 1: Build the PID controller object

Complete the TODO in the [pid_controller.h](https://github.com/udacity/nd013-c6-control-starter/blob/master/project/pid_controller/pid_controller.h) and [pid_controller.cpp](https://github.com/udacity/nd013-c6-control-starter/blob/master/project/pid_controller/pid_controller.cpp).

Run the simulator and see in the desktop mode the car in the CARLA simulator. Take a screenshot and add it to your report. The car should not move in the simulation.

Step 2: PID controller for throttle:

1) In [main.cpp](https://github.com/udacity/nd013-c6-control-starter/blob/master/project/pid_controller/main.cpp), complete the TODO (step 2) to compute the error for the throttle pid. The error is the speed difference between the actual speed and the desired speed.

Useful variables:

- **nx_pt** – stores the index of waypoint ahead of current position.

- The last point of **v_points** vector contains the velocity computed by the path planner.

- **velocity** contains the actual velocity.

- The output of the controller should be inside [-1, 1].

2) Comment your code to explain why did you computed the error this way.

3) Tune the parameters of the pid until you get satisfying results (a perfect trajectory is not expected).

Step 3: PID controller for steer:

1) In [main.cpp](https://github.com/udacity/nd013-c6-control-starter/blob/master/project/pid_controller/main.cpp), complete the TODO (step 3) to compute the error for the steer pid. The error is the angle difference between the actual steer and the desired steer to reach the planned position.

Useful variables:

- The variable **y_points** and **x_point** gives the desired trajectory planned by the path_planner.

- **yaw** gives the actual rotational angle of the car.

- The output of the controller should be inside [-1.2, 1.2].

- If needed, the position of the car is stored in the variables **x_position**, **y_position** and **z_position**

2) Comment your code to explain why did you computed the error this way.

3) Tune the parameters of the pid until you get satisfying results (a perfect trajectory is not expected).

Step 4: Evaluate the PID efficiency

The values of the error and the pid command are saved in thottle_data.txt and steer_data.txt.

Plot the saved values using the command (in nd013-c6-control-refresh/project):

```

python3 plot_pid.py

```

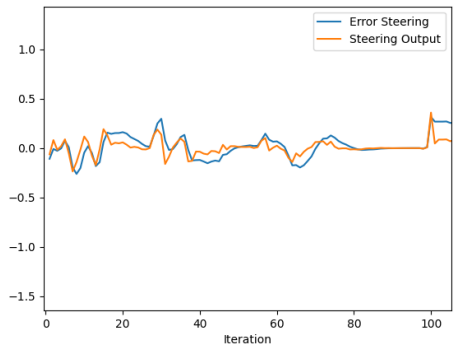
You might need to install a few additional python modules:

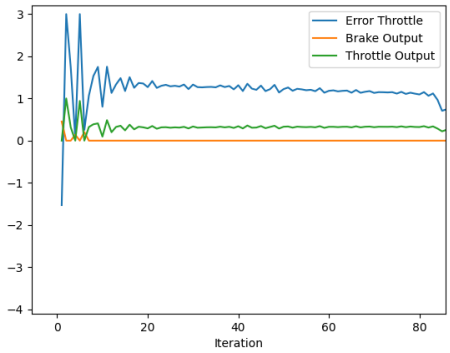
```

pip3 install pandas

pip3 install matplotlib

```

// Answer the following questions:

// Add the plots to your report and explain them (describe what you see)

Answer:





- The first plot shows Steering error and Steering output vs Iterations. We can see that the output closely follows the error and also error stays close to zero. PID parameters can be further tuned to get perfectly overlapping error and output.
- The second plot shows Throttle error and Throttle output vs Iterations. We can see that the output curve follows the error curve but there is a bias in the error and it does not go to zero. This is mainly due to limitations in lateral acceleration. The PID parameters mainly integral parameter can be further tuned to bring the bias close to zero.

// What is the effect of the PID according to the plots, how each part of the PID affects the control command?

Answer:

- Proportional parameter: As seen in the first plot this part helps us to bring the error down to zero based on how big the error is. The bigger the error the higher is the output of proportional part of PID.
- Derivative Parameter: As seen in the second plot this part helps us to respond fast to changing errors. The bigger the rate of change of error the higher is the output of derivative part of PID
- Integral Parameter: This part helps us to bring the bias in error down to zero. The integral part of throttle can be increased to bring down the bias closer to zero.

// How would you design a way to automatically tune the PID parameters?

Answer:

- The first method would to use twiddle algorithm that we learnt in our course but the complexity of the simulation makes it difficult to implement for the given project.
- The second method could be by creating a simple mathematical model of the simulation for example bicycle model of the car and collecting error data and input data such as steering input, velocity and timestamp from the real simulation. Now the simple model can be run for many iterations and an algorithm like twiddle can be implemented to find the best PID parameters.

// PID controller is a model free controller, i.e. it does not use a model of the car. Could you explain the pros and cons of this type of controller?

Pros of model free controller:

- The system(car) can be controlled without any knowledge of the inner workings of the system. The throttle can be controlled without the need of knowing how the motor works.
- Less computation is required.
- Much cheaper and easier to implement.

Cons:

- A model-based controller can provide a much robust control when compared to a model free PID controlled.
- PID controller's parameters have to be changed and computed again when there is a small change in the system. But in a model-based controller only the value of the variable associated with the change have to be tweaked to get the controller working. For example, if a new steering wheel is installed then only some of the parameters of the steering model have to be changed based on the new steering wheel, but in case of PID controller new PID parameters have to be computed which can be challenging.
- Tuning and finding the best control parameters is always a challenging task for a model free controller.

Tips:

- When you wil be testing your c++ code, restart the Carla simulator to remove the former car from the simulation.

- If the simulation freezes on the desktop mode but is still running on the terminal, close the desktop and restart it.

- When you will be tuning the PID parameters, try between those values: