Control and Trajectory Tracking for Autonomous Vehicle

Proportional-Integral-Derivative (PID)

In this project, you will apply the skills you have acquired in this course to design a PID controller to perform vehicle trajectory tracking. Given a trajectory as an array of locations, and a simulation environment, you will design and code a PID controller and test its efficiency on the CARLA simulator used in the industry.

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`

PID controller coding

To build this project following files are updated :

1) pid_controller.h

In pid class, gains & limits parameters added.

```
class PID {
public:

    /**
    * TODO: Create the PID class
    **/

     /*
     * Errors
     */
      double p_error;
      double i_error;
      double d_error;

     /*
     * Coefficients
     */
     double kpi_i;
     double kii_i;
     double kdi_i;

     /*
     * Output limits
     */
     double output_lim_max_i;
     double output_lim_min_i;
     /*
     * Delta time
     */
      double new_delta_time_i ;
     /*
```

2) pid_controller.cpp

pid class's gains & limits are initialized in this function.

```
void PID::Init(double Kpi, double Kii, double Kdi, double output_lim_maxi, double output_lim_mini) {
  /**
   * TODO: Initialize PID coefficients (and errors, if needed)
   **/
  kpi_i = Kpi;
  kii_i = Kii;
  kdi_i = Kdi;
  output_lim_max_i = output_lim_maxi;
  output_lim_min_i = output_lim_mini;
  p_error = 0;
  i_error = 0;
  d_error = 0;
}
```

in this function we are updating the proportional, derivative & integral errors

```
void PID::UpdateError(double cte) {
  /**
   * TODO: Update PID errors based on cte.
   **/
  double pre_perror = p_error;
  p_error = cte;
  if (new_delta_time_i>0)
    {
      d_error = (cte - pre_perror) / new_delta_time_i;
    }
    else {
      d_error = 0 ;
    }

  i_error = i_error + (cte)*new_delta_time_i;
}
```

Calculating the control values based on errors & pid gains

```
double PID::TotalError() {
  /**
   * TODO: Calculate and return the total error
   * The code should return a value in the interval [output_lim_mini,
   * output_lim_maxi]
   */
  double control;

  control = (-p_error * kpi_i - d_error * kdi_i - i_error * kii_i);

  control = min(output_lim_max_i, max(output_lim_min_i, control));

  return control;
}
```

updating the delta time to public parameter of pid class

```
double PID::UpdateDeltaTime(double new_delta_time) {
  /**
   * TODO: Update the delta time with new value
   */
  new_delta_time_i = new_delta_time;

  return new_delta_time_i;
}
```

3) main.cpp

creating pid class object & initialized the pid gains & limits

```
// initialize pid steer
/**
* TODO (Step 1): create pid (pid_steer) for steer command and initialize values
**/
PID pid_steer = PID();


// initialize pid throttle
/**
* TODO (Step 1): create pid (pid_throttle) for throttle command and initialize values
**/
PID pid_throttle = PID();
pid_steer.Init(0.3, 0.001, 1.2, 1.2, -1.2);
pid_throttle.Init(0.1, 0.0001, 1.2, 1.0, -1.0);
```

Calculating steer error based on current yaw angle & target yaw angle.

```
double steer_output;

/**
* TODO (step 3): compute the steer error (error_steer) from the position and the desired trajectory
**/
 error_steer =  yaw - (angle_between_points(x_position,y_position, x_points[x_points.size()-1], y_points[y_points.size()-1]));

/**
* TODO (step 3): uncomment these lines
```

Calculating throttle error based current velocity & target velocity.

```
// modify the following line for step 2
error_throttle = 0;

error_throttle =   velocity - v_points[v_points.size()-1];
```

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):

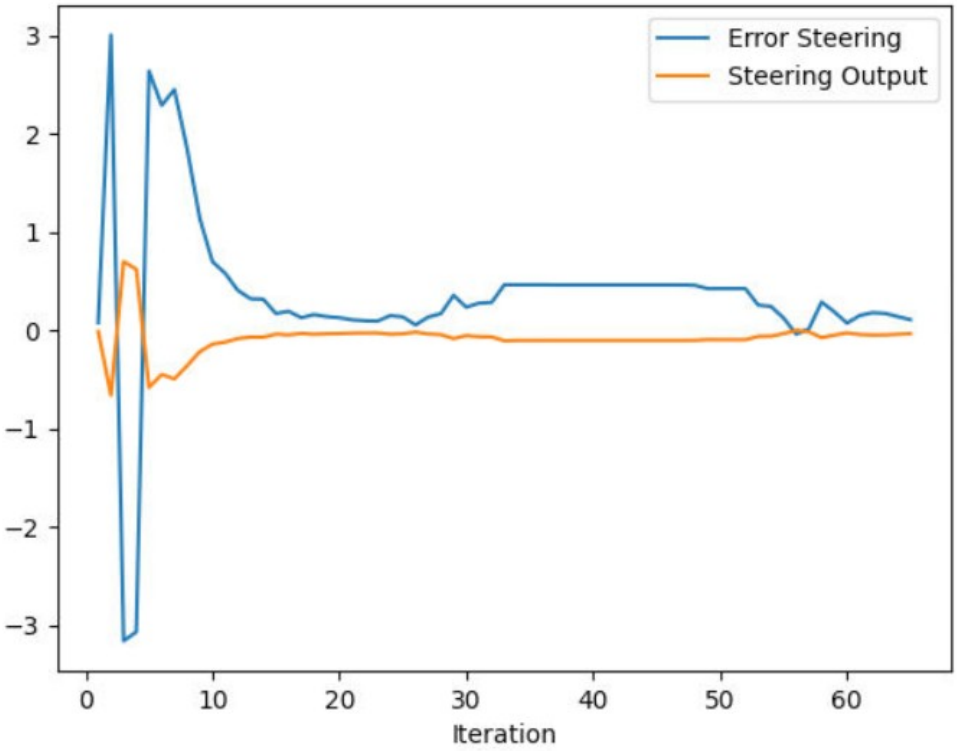
```
python3 plot_pid.py
```

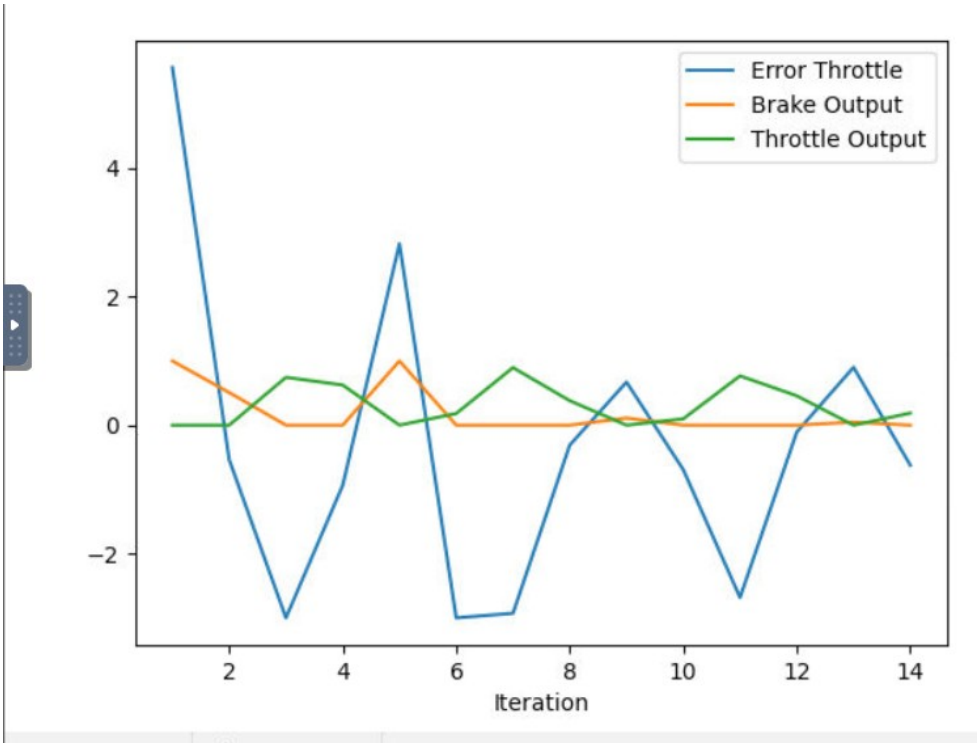
You might need to install a few additional python modules:

```
pip3 install pandas
pip3 install matplotlib
```

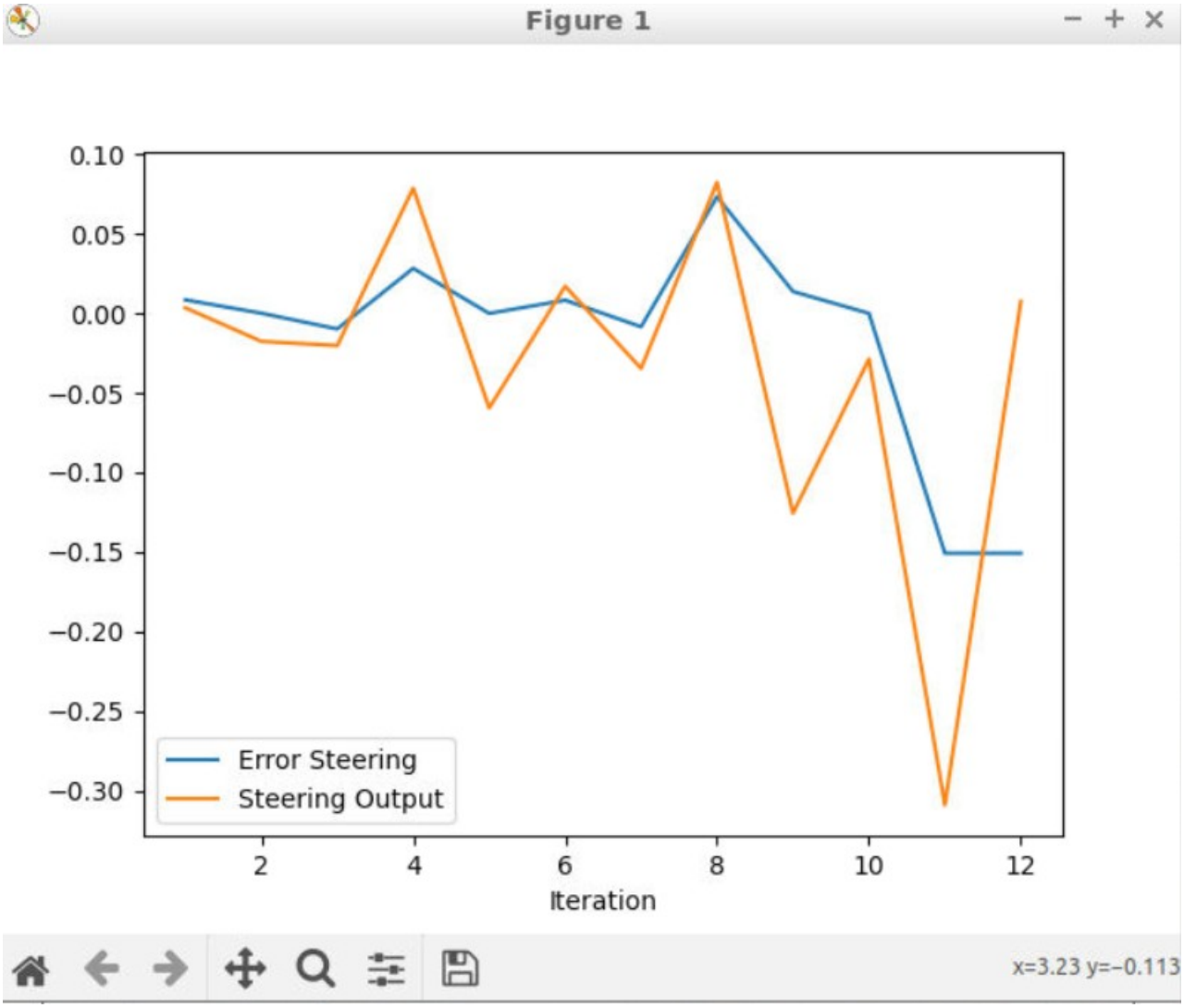
Answer the following questions:

Initially we can see there is large error and after some time error reduces, more tuning is required in the pid gains.





Case2



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

Following are the effect of the PID according to the plots

With the increase of P, we can reduce the error to zero,

I reduces the steady state error

Increasing the derivative gain increases the speed of response.

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

we can use adaptive controller method to tune the PID parameters also with machine learning method, we can tune the gains.

// 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 Controllers : PID controller model is easy to implement & understand.

Cons of Controllers : Not suitable non linear models (aeroplane, cars etc).Some time tuning of PID gains can be problem.

// (Optional) What would you do to improve the PID controller?

Improvement points of PID controllers:

- automatic tuning PID gains
- Make PID controller suitable for non linear models.