UDACITY

‹ Return to "Self-Driving Car Engineer" in the classroom

DISCUSS ON STUDENT HUB

PID Controller

REVIEW

CODE REVIEW 2

HISTORY

src/main.cpp 1

```
#include <math.h>
#include <uWS/uWS.h>
#include <iostream>
#include <string>
#include "json.hpp"
#include "PID.h"

// for convenience
using nlohmann::json;
using std::string;

// For converting back and forth between radians and degrees.
constexpr double pi() { return M_PI; }
double deg2rad(double x) { return x * pi() / 180; }
double rad2deg(double x) { return x * 180 / pi(); }

// Checks if the SocketIO event has JSON data.
// If there is data the JSON object in string format will be returned,
// else the empty string "" will be returned.
string hasData(string s) {
  auto found_null = s.find("null");
  auto b1 = s.find_first_of("[");
  auto b2 = s.find_last_of("]");
  if (found_null != string::npos) {
    return "";
  }
  else if (b1 != string::npos && b2 != string::npos) {
    return s.substr(b1, b2 - b1 + 1);
  }
  return "";
}

int main() {
  uWS::Hub h;

  PID pid;
  double init_Kp = -0.15;
  double init_Ki = -0.0;
  double init_Kd = -0.8;
  pid.Init(init_Kp, init_Ki, init_Kd);

  h.onMessage([&pid](uWS::WebSocket<uWS::SERVER> ws, char *data, size_t length
                     uWS::OpCode opCode) {
    // "42" at the start of the message means there's a websocket message even
    // The 4 signifies a websocket message
    // The 2 signifies a websocket event
    if (length && length > 2 && data[0] == '4' && data[1] == '2') {
      auto s = hasData(string(data).substr(0, length));

      if (s != "") {
        auto j = json::parse(s);

        string event = j[0].get<string>();

        if (event == "telemetry") {
          // j[1] is the data JSON object
          double cte = std::stod(j[1]["cte"].get<string>());
          double speed = std::stod(j[1]["speed"].get<string>());
          double angle = std::stod(j[1]["steering_angle"].get<string>());
          double steer_value;

          pid.UpdateError(cte);
          steer_value = pid.TotalError();
          double throttle = 0.3 + 0.40 * (0.05 - abs(steer_value)) / (0.05 + a

```

AWESOME

Great implementation on the steering control! This actually made your car drive really in a smooth way. Ar
extra challenge!

```
          // DEBUG
          std::cout << "CTE: " << cte << " Steering Value: " << steer_value <<
                    << std::endl;

          json msgJson;
          msgJson["steering_angle"] = steer_value;
          msgJson["throttle"] = throttle;
          auto msg = "42[\"steer\"," + msgJson.dump() + "]";
          std::cout << msg << std::endl;
          ws.send(msg.data(), msg.length(), uWS::OpCode::TEXT);
        }  // end "telemetry" if
      } else {
        // Manual driving
        string msg = "42[\"manual\",{}]";
        ws.send(msg.data(), msg.length(), uWS::OpCode::TEXT);
      }
    }  // end websocket message if
  }); // end h.onMessage

  h.onConnection([&h](uWS::WebSocket<uWS::SERVER> ws, uWS::HttpRequest req) {
    std::cout << "Connected!!!" << std::endl;
  });

  h.onDisconnection([&h](uWS::WebSocket<uWS::SERVER> ws, int code,
                         char *message, size_t length) {
    ws.close();
    std::cout << "Disconnected" << std::endl;
  });

  int port = 4567;
  if (h.listen(port)) {
    std::cout << "Listening to port " << port << std::endl;
  } else {
    std::cerr << "Failed to listen to port" << std::endl;
    return -1;
  }

  h.run();
}

```

▸ src/PID.cpp 1

▸ src/PID.h

▸ cmakepatch.txt

▸ README.md

▸ CMakeLists.txt

RETURN TO PATH

Rate this review