**Name:** Aryan Gupta
**Partner:** Anthony Samagaio

**Video Link:** https://youtu.be/gGr8TEWXEHc

**Objective:** The video will demonstrate a program to drive a vehicle following a line with a 45° bend then stop the vehicle at a perpendicular line. The vehicle must not deviate more than 2 cm from the center of the line.

**Reflection:** The first iteration of the program consisted of a simple if else if statement on the magnitude of the line position, however it was determined this algorithm was not sufficient in accurately driving the vehicle on the line. A PID tuned system was consequently implemented. Due to the fact that a line with a single 45° is not complex enough to tune a full PID system, only the P and I parts were tuned. Time and motivation permitting, a more complex line could be printed to further test a PID system.

**Code:**

```
/*
* File header excluded for brevity
*/

#include "SimpleRSLK.h"

uint16_t sensorVal[LS_NUM_SENSORS];
uint16_t sensorCalVal[LS_NUM_SENSORS];
uint16_t sensorMaxVal[LS_NUM_SENSORS];
uint16_t sensorMinVal[LS_NUM_SENSORS];

/// PID Const Variables
#define PID_OUT_MIN -10
#define PID_OUT_MAX 10
#define PID_Kp 0.004
#define PID_Ki 0.0001
#define PID_Kd 0 // D is not used currently
#define PID_SETPOINT 3500

/// Binds a variable to a lower or upper bind
/// I thought it would be used more often, I guess not
#define LOWER_BIND(X, Y) if ((X) < (Y)) { (X) = (Y); }
#define UPPER_BIND(X, Y) if ((X) > (Y)) { (X) = (Y); }

/// Update the PID values for the current system
/// and return the new output. If a value of
/// (-1, -1, -1) is passed in as parameters, it resets
/// the internal values to 0. This is useful if he vehicle
/// is moved and there are bad values in the integration
/// history.
///
/// @ref https://gist.github.com/bradley219/5373998
///
/// @param dt change in time from last func call
/// @param cur current PID input value
/// @param setpoint value PID input should be
```

```
int8_t update_pid(float dt, float cur, float setpoint = PID_SETPOINT) {
   static float integral = 0;
   static float pre_error = 0;

   if (dt == -1 and cur == -1 and setpoint == -1) {
       integral = 0;
       pre_error = 0;
   }

   // Calculate error
   float error = setpoint - cur;

   // Proportional term
   float Pout = PID_Kp * error;

   // Integral term
   integral += error * (dt / 1000); // millis -> seconds
   Serial.print(" DT: ");
   Serial.print(dt);
   Serial.print(" ");
   Serial.print(" ER: ");
   Serial.print(error);
   Serial.print(" ");
   Serial.print(" IG: ");
   Serial.print(integral);
   Serial.print(" ");
   float Iout = PID_Ki * integral;

   // Derivative term
   float ddt = (error - pre_error) / dt;
   float Dout = PID_Kd * ddt;

   // Calculate total output
   float output = Pout + Iout + Dout;

   // Restrict to max/min
   UPPER_BIND(output, PID_OUT_MAX);
   LOWER_BIND(output, PID_OUT_MIN);

   // Save error to previous error
   pre_error = error;

   return output;
}

bool isCalibrationComplete = false;
void setup()
{
   Serial.begin(115200);

   setupRSLK();
   /* Left button on Launchpad */
   setupWaitBtn(LP_LEFT_BTN);
   /* Red led in rgb led */
   setupLed(RED_LED);
   clearMinMax(sensorMinVal,sensorMaxVal);

   // Some default values so I dont have to calibrate
   // it every time during PID tuning
   sensorMinVal[0] = 830;
   sensorMinVal[1] = 904;
   sensorMinVal[2] = 643;
   sensorMinVal[3] = 721;
   sensorMinVal[4] = 656;
   sensorMinVal[5] = 804;
   sensorMinVal[6] = 751;
   sensorMinVal[7] = 873;

   sensorMaxVal[0] = 944;
   sensorMaxVal[1] = 1033;
   sensorMaxVal[2] = 733;
   sensorMaxVal[3] = 829;
   sensorMaxVal[4] = 732;
   sensorMaxVal[5] = 908;
   sensorMaxVal[6] = 860;
   sensorMaxVal[7] = 975;

   /* Run this setup only once */
   if(isCalibrationComplete == false) {
       floorCalibration();
       isCalibrationComplete = true;
   }

   Serial.println("Finished with setup");
}

void floorCalibration() {
   /* Place Robot On Floor (no line) */
   delay(2000);
   String btnMsg = "Push left button on Launchpad to begin calibration.\n";
   btnMsg += "Make sure the robot is on the floor away from the line.\n";
   /* Wait until button is pressed to start robot */
   waitBtnPressed(LP_LEFT_BTN,btnMsg,RED_LED);

   delay(1000);

   Serial.println("Running calibration on floor");
   simpleCalibrate();
   Serial.println("Reading floor values complete");

   btnMsg = "Push left button on Launchpad to begin line following.\n";
   btnMsg += "Make sure the robot is on the line.\n";
   /* Wait until button is pressed to start robot */
   waitBtnPressed(LP_LEFT_BTN,btnMsg,RED_LED);
   delay(1000);

   enableMotor(BOTH_MOTORS);
}

void simpleCalibrate() {
   /* Set both motors direction forward */
   setMotorDirection(BOTH_MOTORS,MOTOR_DIR_FORWARD);
   /* Enable both motors */
   enableMotor(BOTH_MOTORS);
   /* Set both motors speed 20 */
   setMotorSpeed(BOTH_MOTORS,20);

   for(int x = 0;x<100;x++){
       readLineSensor(sensorVal);
       setSensorMinMax(sensorVal,sensorMinVal,sensorMaxVal);
   }
   /* Disable both motors */
   disableMotor(BOTH_MOTORS);
}

void loop() {
   static bool done = false;

   // Wait for button press after a run so we can run again
   if (done) {
       disableMotor(BOTH_MOTORS);
       waitBtnPressed(LP_LEFT_BTN,"Waiting for Button Press",RED_LED);
       delay(2000);
       enableMotor(BOTH_MOTORS);
       update_pid(-1, -1, -1);
       done = false;
   }

   static unsigned long last_time = 0;
   uint8_t normalSpeed = 10;

   // read sensor values
   readLineSensor(sensorVal);
   readCalLineSensor(sensorVal,sensorCalVal,sensorMinVal,sensorMaxVal,DARK_LINE);

   // If more than 5 sensors read a black line, then assume that we are
   // at the end of the track and stop. This assumes a calibrated sensor
   // value of more than 750 (values go from 0 to 1000) is a line.
   int count = 0;
   for (int i = 0; i < LS_NUM_SENSORS; ++i) {
       if (sensorCalVal[i] > 750) {
           ++count;
       }
   }

   if (count >= 5) {
       Serial.println("End Found");
       done = true;
       return;
   }

   // Get line position
   uint32_t linePos = getLinePosition(sensorCalVal,DARK_LINE);
   Serial.print(linePos);

   // if the position is 0 then no line is under the vehicle
   // and line is lost
   if (linePos == 0) {
       Serial.println("Line Lost");
       done = true;
   }

   // calculate delta time for PID
   auto current_time = millis();
   auto dt = current_time - last_time;
   last_time = current_time;

   // PID
   int8_t change = update_pid(dt, linePos);
   Serial.print("\t");
   Serial.print(change);
```

```
    // calculate wheel speeds from PID values
    int8_t leftMotorSpeed = normalSpeed - change;
    int8_t rightMotorSpeed = normalSpeed + change;

    // If PID value makes one of the wheel speed go
    // out of bounds, add it to the opposite wheel
    if (leftMotorSpeed < 0) {
        rightMotorSpeed += -leftMotorSpeed;
        leftMotorSpeed = 0;
    }

    if (rightMotorSpeed < 0) {
        leftMotorSpeed += -rightMotorSpeed;
        rightMotorSpeed = 0;
    }

    Serial.print("\t");
    Serial.print(leftMotorSpeed);
    Serial.print("\t");
    Serial.print(rightMotorSpeed);

    // update wheel speed
    setMotorSpeed(LEFT_MOTOR, leftMotorSpeed);
    setMotorSpeed(RIGHT_MOTOR, rightMotorSpeed);

    Serial.println();
}
```