/\* FINAL CODE v1.1

Matt Russell, Joshua Ramayrat

This is the code that will guide our robot through the obstacle course.

\*/

#include <QTRSensors.h>

#define MINGUESS 5000

//how many samples to take for calibration

#define CALNUM 100

//how long to wait between calibration

#define CALWAIT 1000

//basespeed of motors

#define BASESPEED 90

//how many data points to take for line-following

#define FOLNUM 10

//arbitrary constant for line following

#define ARBC 500

int goPin = 12, swPin = 30, readyPin = 26, statusPin = 24;

//QTR Sensor Calibration

//Pins

int QTRLEDon = 22, QTRmod1 = 41, QTRmod2 = 42, QTRmod3 = 43, QTRmod4 = 44, QTRmod5 = 45, QTRmod6 = 46, QTRmod7 = 47, QTRmod8 = 48;

//Structures

QTRSensorsRC qtrrc((unsigned char[]) {

41, 42, 43, 44, 45, 46, 47, 48

}, 8);

/\*unsigned int maxVal[] = {0, 0, 0, 0, 0, 0, 0, 0};

unsigned int minVal[] = {MINGUESS, MINGUESS, MINGUESS, MINGUESS, MINGUESS, MINGUESS, MINGUESS, MINGUESS};

volatile unsigned int wMin[] = {0, 0, 0, 0, 0, 0, 0, 0};

volatile unsigned int wMax[] = {0, 0, 0, 0, 0, 0, 0, 0};

volatile unsigned int wAvg[] = {0, 0, 0, 0, 0, 0, 0, 0};

volatile unsigned int bMin[] = {0, 0, 0, 0, 0, 0, 0, 0};

volatile unsigned int bMax[] = {0, 0, 0, 0, 0, 0, 0, 0};

volatile unsigned int bAvg[] = {0, 0, 0, 0, 0, 0, 0, 0};\*/

unsigned int sensorValues[8]; /\*sensor1[100], sensor2[100], sensor3[100], sensor4[100], sensor5[100], sensor6[100], sensor7[100], sensor8[100];\*/

//avg[] = {lValavg, rValavg};

long avg[] = {0, 0}, blackVal[8], whiteVal[8];

//Motors

// Left motor

int enLeft = 5, in1 = 7, in2 = 6;

// Right motor

int enRight = 10, in3 = 8, in4 = 9;

//Encoder

long count = 0;

int encLeft = 3, encRight = 4;

//Line-following

unsigned int newspd = BASESPEED;

int lVal, rVal;

long threshold;

int diff[CALNUM];

int stepPin = 34, dirPin = 35;

void setup() {

digitalWrite(statusPin, LOW);

//Initialize Serial Console only for debugging

Serial.begin(9600);

//Setup for calibration

pinMode(QTRmod1, INPUT);

pinMode(QTRmod2, INPUT);

pinMode(QTRmod3, INPUT);

pinMode(QTRmod4, INPUT);

pinMode(QTRmod5, INPUT);

pinMode(QTRmod6, INPUT);

pinMode(QTRmod7, INPUT);

pinMode(QTRmod8, INPUT);

pinMode(readyPin, OUTPUT);

pinMode(statusPin, OUTPUT);

pinMode(goPin, OUTPUT);

pinMode(QTRLEDon, OUTPUT);

//Setup for line-following

pinMode(swPin, INPUT);

pinMode(in1, OUTPUT);

pinMode(in2, OUTPUT);

pinMode(in3, OUTPUT);

pinMode(in4, OUTPUT);

pinMode(enLeft, OUTPUT);

pinMode(enRight, OUTPUT);

pinMode(encLeft, INPUT);

pinMode(encRight, INPUT);

//Setup for stepper

pinMode(stepPin, OUTPUT);

pinMode(dirPin, OUTPUT);

//calibrate for the line

digitalWrite(statusPin, HIGH);

while (digitalRead(swPin) != LOW) {

}

digitalWrite(statusPin, LOW);

digitalWrite(QTRLEDon, HIGH);

lCal();

digitalWrite(statusPin, HIGH);

rdelay(CALWAIT);

//calibrate for black

while (digitalRead(swPin) != LOW) {

}

digitalWrite(statusPin, LOW);

bCal();

digitalWrite(statusPin, HIGH);

rdelay(CALWAIT);

//Calibrate for white

while (digitalRead(swPin) != LOW) {

}

digitalWrite(statusPin, LOW);

wCal();

//Ready to begin

digitalWrite(readyPin, HIGH);

while (digitalRead(swPin) != LOW) {

}

digitalWrite(readyPin, LOW);

Reverse();

delay(500);

Forward();

delay(500);

}

void loop() {

follow();

}

//Line-Following functions

//Function to compute sensor values

void follow() {

//lVal for 1-3, rVal for 6-8

lVal = 0, rVal = 0;

qtrrc.read(sensorValues);

for (int i = 0; i < 3; i++) {

lVal = lVal + sensorValues[i];

rVal = rVal + sensorValues[i + 5];

}

//Serial.println(rVal - lVal);

//check if drifting right

if (rVal - lVal > threshold + ARBC) {

Serial.println("Drifting Right");

digitalWrite(in1, HIGH);

digitalWrite(in2, LOW);

analogWrite(enLeft, 0);

digitalWrite(in3, HIGH);

digitalWrite(in4, LOW);

analogWrite(enRight, newspd + 10);

}

//check if drifting left

if (rVal - lVal < threshold - ARBC) {

Serial.println("Drifting Left");

digitalWrite(in1, HIGH);

digitalWrite(in2, LOW);

analogWrite(enLeft, newspd + 10);

digitalWrite(in3, HIGH);

digitalWrite(in4, LOW);

analogWrite(enRight, 0);

}

//drifting 'straight'

if (rVal - lVal < threshold + ARBC && rVal - lVal > threshold - ARBC) {

Serial.println("Drifting Straight");

digitalWrite(in1, HIGH);

digitalWrite(in2, LOW);

analogWrite(enLeft, newspd);

digitalWrite(in3, HIGH);

digitalWrite(in4, LOW);

analogWrite(enRight, newspd);

}

//exception handling

//Right Turn

if ((sensorValues[0] > blackVal[0] - ARBC) && (sensorValues[1] > blackVal[1] - ARBC)) {

digitalWrite(in1, HIGH);

digitalWrite(in2, LOW);

analogWrite(enLeft, newspd);

digitalWrite(in3, LOW);

digitalWrite(in4, HIGH);

analogWrite(enRight, newspd + 60);

}

//Left Turn

if ((sensorValues[6] > blackVal[6] - ARBC) && (sensorValues[7] > blackVal[7] - ARBC)) {

digitalWrite(in1, LOW);

digitalWrite(in2, HIGH);

analogWrite(enLeft, newspd + 60);

digitalWrite(in3, HIGH);

digitalWrite(in4, LOW);

analogWrite(enRight, newspd);

}

}

//Calibration for line

void lCal() {

for (int i = 0; i < CALNUM; i++) {

lVal = 0, rVal = 0;

qtrrc.read(sensorValues);

for (int j = 0; j < 3; j++) {

lVal = lVal + sensorValues[j];

rVal = rVal + sensorValues[j + 5];

}

diff[i] = rVal - lVal;

threshold = threshold + diff[i];

}

threshold = threshold / CALNUM;

Serial.println(threshold);

}

//Calibration for black, gives us value when each sensor is looking at black

void bCal() {

for (int i = 0; i < 8; i ++) {

blackVal[i] = 0;

}

for (int i = 0; i < CALNUM; i++) {

qtrrc.read(sensorValues);

for (int j = 0; j < 8; j++) {

blackVal[j] = blackVal[j] + sensorValues[j];

}

}

for (int i = 0; i < 8; i++) {

blackVal[i] = blackVal[i] / CALNUM;

Serial.println(blackVal[i]);

}

}

//Calibration for white, gives us value when each sensor is looking at white

void wCal() {

for (int i = 0; i < 8; i ++) {

whiteVal[i] = 0;

}

for (int i = 0; i < CALNUM; i++) {

qtrrc.read(sensorValues);

for (int j = 0; j < 8; j++) {

whiteVal[j] = whiteVal[j] + sensorValues[j];

}

}

for (int i = 0; i < 8; i++) {

whiteVal[i] = whiteVal[i] / CALNUM;

Serial.println(whiteVal[i]);

}

}

//Motor functions

void Forward() {

//Left motor

digitalWrite(in1, HIGH);

digitalWrite(in2, LOW);

analogWrite(enLeft, BASESPEED);

//Right motor

digitalWrite(in3, HIGH);

digitalWrite(in4, LOW);

analogWrite(enRight, BASESPEED);

}

void Reverse() {

//Left motor

digitalWrite(in1, LOW);

digitalWrite(in2, HIGH);

analogWrite(enLeft, BASESPEED);

//Right motor

digitalWrite(in3, LOW);

digitalWrite(in4, HIGH);

analogWrite(enRight, BASESPEED);

}

void stepper(){

digitalWrite(12,trig);

for(int i=0; i<250; i++){

digitalWrite(13,HIGH);

delay(1);

digitalWrite(13, LOW);

delay(1);

}

delay(1000);

trig = !trig;

digitalWrite(12,trig);

for(int i=0; i<100; i++){

digitalWrite(13,HIGH);

delay(1);

digitalWrite(13, LOW);

delay(1);

}

}

//General-purpose functions

void rdelay(int dt) {

int ct, pt;

pt = millis();

ct = pt;

while (ct - pt < dt) {

ct = millis();

}

}