/\* FINAL CODE

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

//basespeed of motors

#define BASESPEED 45

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

#define FOLNUM 10

//Arbitrary 'bound constant' used in line-following to determine drift condition

#define DRIFTC 1000

//QTR Sensor Calibration

//Pins

int calStart = 26, QTRLEDon = 22, statusPin = 24, 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];

volatile long avg[] = {0, 0, 0, 0, 0, 0, 0, 0};

//Motors

// motor one

int enA = 10, in1 = 9, in2 = 8;

// motor two

int enB = 5, in3 = 7, in4 = 6;

//Encoder

long count = 0;

int encA = 3, encB = 20;

//Line-following

unsigned int fArray[8 \* FOLNUM];

int adjC = 5; //used to readjust motor speed

int driftState = 0; //used to determine adjC

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(calStart, OUTPUT);

pinMode(statusPin, OUTPUT);

pinMode(QTRLEDon, OUTPUT);

attachInterrupt(digitalPinToInterrupt(19), wCal, RISING);

attachInterrupt(digitalPinToInterrupt(18), bCal, RISING);

//Setup for line-following

pinMode(in1, OUTPUT);

pinMode(in2, OUTPUT);

pinMode(in3, OUTPUT);

pinMode(in4, OUTPUT);

pinMode(enA, OUTPUT);

pinMode(enB, OUTPUT);

pinMode(encA, INPUT);

pinMode(encB, INPUT);

//Indicate ready for calibration

digitalWrite(statusPin, HIGH);

}

void loop() {

// put your main code here, to run repeatedly:

}

//Calibration Functions

void wCal() {

digitalWrite(calStart,HIGH);

digitalWrite(statusPin,LOW);

digitalWrite(QTRLEDon, HIGH);

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

qtrrc.read(sensorValues);

//Store raw data

sensor1[i] = sensorValues[0];

sensor2[i] = sensorValues[1];

sensor3[i] = sensorValues[2];

sensor4[i] = sensorValues[3];

sensor5[i] = sensorValues[4];

sensor6[i] = sensorValues[5];

sensor7[i] = sensorValues[6];

sensor8[i] = sensorValues[7];

}

//Data processing: 4000 reading is noise in white case, give it nonsensical value

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

if (sensor1[i] == 4000) {

sensor1[i] = 5000;

}

if (sensor2[i] == 4000) {

sensor2[i] = 5000;

}

if (sensor3[i] == 4000) {

sensor3[i] = 5000;

}

if (sensor4[i] == 4000) {

sensor4[i] = 5000;

}

if (sensor5[i] == 4000) {

sensor5[i] = 5000;

}

if (sensor6[i] == 4000) {

sensor6[i] = 5000;

}

if (sensor7[i] == 4000) {

sensor7[i] = 5000;

}

if (sensor8[i] == 4000) {

sensor8[i] = 5000;

}

}

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

if (sensor1[i] != 5000) {

avg[0] = avg[0] + sensor1[i];

}

if (sensor2[i] != 5000) {

avg[1] = avg[1] + sensor2[i];

}

if (sensor3[i] != 5000) {

avg[2] = avg[2] + sensor3[i];

}

if (sensor4[i] != 5000) {

avg[3] = avg[3] + sensor4[i];

}

if (sensor5[i] != 5000) {

avg[4] = avg[4] + sensor5[i];

}

if (sensor6[i] != 5000) {

avg[5] = avg[5] + sensor6[i];

}

if (sensor7[i] != 5000) {

avg[6] = avg[6] + sensor7[i];

}

if (sensor8[i] != 5000) {

avg[7] = avg[7] + sensor8[i];

}

}

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

avg[i] = avg[i] / CALNUM;

}

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

if (maxVal[0] < sensor1[i] && sensor1[i] != 5000) {

maxVal[0] = sensor1[i];

}

if (maxVal[1] < sensor2[i] && sensor2[i] != 5000) {

maxVal[1] = sensor2[i];

}

if (maxVal[2] < sensor3[i] && sensor3[i] != 5000) {

maxVal[2] = sensor3[i];

}

if (maxVal[3] < sensor4[i] && sensor4[i] != 5000) {

maxVal[3] = sensor4[i];

}

if (maxVal[4] < sensor5[i] && sensor5[i] != 5000) {

maxVal[4] = sensor5[i];

}

if (maxVal[5] < sensor6[i] && sensor6[i] != 5000) {

maxVal[5] = sensor6[i];

}

if (maxVal[6] < sensor7[i] && sensor7[i] != 5000) {

maxVal[6] = sensor7[i];

}

if (maxVal[7] < sensor8[i] && sensor8[i] != 5000) {

maxVal[7] = sensor8[i];

}

if (minVal[0] > sensor1[i]) {

minVal[0] = sensor1[i];

}

if (minVal[1] > sensor2[i]) {

minVal[1] = sensor2[i];

}

if (minVal[2] > sensor3[i]) {

minVal[2] = sensor3[i];

}

if (minVal[3] > sensor4[i]) {

minVal[3] = sensor4[i];

}

if (minVal[4] > sensor5[i]) {

minVal[4] = sensor5[i];

}

if (minVal[5] > sensor6[i]) {

minVal[5] = sensor6[i];

}

if (minVal[6] > sensor7[i]) {

minVal[6] = sensor7[i];

}

if (minVal[7] > sensor8[i]) {

minVal[7] = sensor8[i];

}

}

//Store all min, average, and max calibrated values into appropriate variables

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

wMin[i] = minVal[i];

wMax[i] = maxVal[i];

wAvg[i] = avg[i];

}

Serial.println("Background Values");

Serial.print("Min");

Serial.print('\t');

Serial.print("Max");

Serial.print('\t');

Serial.println("Average");

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

Serial.print(wMin[i]);

Serial.print('\t');

Serial.print(wMax[i]);

Serial.print('\t');

Serial.println(wAvg[i]);

}

digitalWrite(QTRLEDon, LOW);

digitalWrite(calStart,LOW);

digitalWrite(statusPin,HIGH);

}

void bCal() {

digitalWrite(QTRLEDon, HIGH);

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

qtrrc.read(sensorValues);

//Store raw data

sensor1[i] = sensorValues[0];

sensor2[i] = sensorValues[1];

sensor3[i] = sensorValues[2];

sensor4[i] = sensorValues[3];

sensor5[i] = sensorValues[4];

sensor6[i] = sensorValues[5];

sensor7[i] = sensorValues[6];

sensor8[i] = sensorValues[7];

}

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

if (sensor1[i] != 5000) {

avg[0] = avg[0] + sensor1[i];

}

if (sensor2[i] != 5000) {

avg[1] = avg[1] + sensor2[i];

}

if (sensor3[i] != 5000) {

avg[2] = avg[2] + sensor3[i];

}

if (sensor4[i] != 5000) {

avg[3] = avg[3] + sensor4[i];

}

if (sensor5[i] != 5000) {

avg[4] = avg[4] + sensor5[i];

}

if (sensor6[i] != 5000) {

avg[5] = avg[5] + sensor6[i];

}

if (sensor7[i] != 5000) {

avg[6] = avg[6] + sensor7[i];

}

if (sensor8[i] != 5000) {

avg[7] = avg[7] + sensor8[i];

}

}

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

avg[i] = avg[i] / CALNUM;

}

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

if (maxVal[0] < sensor1[i] && sensor1[i] != 5000) {

maxVal[0] = sensor1[i];

}

if (maxVal[1] < sensor2[i] && sensor2[i] != 5000) {

maxVal[1] = sensor2[i];

}

if (maxVal[2] < sensor3[i] && sensor3[i] != 5000) {

maxVal[2] = sensor3[i];

}

if (maxVal[3] < sensor4[i] && sensor4[i] != 5000) {

maxVal[3] = sensor4[i];

}

if (maxVal[4] < sensor5[i] && sensor5[i] != 5000) {

maxVal[4] = sensor5[i];

}

if (maxVal[5] < sensor6[i] && sensor6[i] != 5000) {

maxVal[5] = sensor6[i];

}

if (maxVal[6] < sensor7[i] && sensor7[i] != 5000) {

maxVal[6] = sensor7[i];

}

if (maxVal[7] < sensor8[i] && sensor8[i] != 5000) {

maxVal[7] = sensor8[i];

}

if (minVal[0] > sensor1[i]) {

minVal[0] = sensor1[i];

}

if (minVal[1] > sensor2[i]) {

minVal[1] = sensor2[i];

}

if (minVal[2] > sensor3[i]) {

minVal[2] = sensor3[i];

}

if (minVal[3] > sensor4[i]) {

minVal[3] = sensor4[i];

}

if (minVal[4] > sensor5[i]) {

minVal[4] = sensor5[i];

}

if (minVal[5] > sensor6[i]) {

minVal[5] = sensor6[i];

}

if (minVal[6] > sensor7[i]) {

minVal[6] = sensor7[i];

}

if (minVal[7] > sensor8[i]) {

minVal[7] = sensor8[i];

}

}

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

bMin[i] = minVal[i];

bMax[i] = maxVal[i];

bAvg[i] = avg[i];

}

Serial.println("Line Values");

Serial.print("Min");

Serial.print('\t');

Serial.print("Max");

Serial.print('\t');

Serial.println("Average");

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

Serial.print(bMin[i]);

Serial.print('\t');

Serial.print(bMax[i]);

Serial.print('\t');

Serial.println(bAvg[i]);

}

digitalWrite(QTRLEDon, LOW);

}

//Line-Following functions

//Function to compute sensor values

void follow() {

//Zero avg[]

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

avg[i] = 0;

}

//Get 10 data points from each sensor

//(How long does this for-loop take?)

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

qtrrc.read(sensorValues);

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

fArray[10 \* i + j] = sensorValues[i];

}

}

//Compute average sensor value

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

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

avg[i] = avg[i] + fArray[10 \* i + j];

}

}

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

avg[i] = avg[i] / FOLNUM;

}

}

//Function to determine robot drift and adjust accordingly

void adjust() {

//Figure out whether Robot is drifting and in which direction

//Check if it has drifted to right and, if so, how severely

//First check sensor6 to see if it has drifted

if (avg[5] > wMax[5] + DRIFTC) {

//check to see if robot was last seen drifting to right

if (driftState == 1) {

adjC++;

}

//drifted to right, need to run rightmost motor slightly faster and leftmost slightly slower to correct

analogWrite(enA, BASESPEED + adjC);

analogWrite(enB, BASESPEED - adjC);

driftState = 1; //store that we last drifted to the right

}

//Now check to see if it has drifted to left

if (avg[3] > wMax[3] + DRIFTC) {

//check to see whether robot was last seen drifting to left

if (driftState == -1) {

adjC++;

}

//drifted to left, need to run rightmost motor slightly slower and leftmost slightly faster to correct

analogWrite(enA, BASESPEED - adjC);

analogWrite(enB, BASESPEED + adjC);

driftState = -1;

}

else {

driftState = 0;

//this might need to be moved or just straight up removed

adjC--;

}

}

//Motor functions

void Forward() {

digitalWrite(in1, HIGH);

digitalWrite(in2, LOW);

analogWrite(enA, BASESPEED);

digitalWrite(in3, HIGH);

digitalWrite(in4, LOW);

analogWrite(enB, BASESPEED);

}

void Reverse() {

digitalWrite(in1, LOW);

digitalWrite(in2, HIGH);

analogWrite(enA, BASESPEED);

digitalWrite(in3, LOW);

digitalWrite(in4, HIGH);

analogWrite(enB, BASESPEED);

}

//General-purpose functions

void rdelay(int dt) {

int ct, pt;

pt = millis();

ct = pt;

while (ct - pt < dt) {

ct = millis();

}

}