#include <Wire.h>

//Motor setup

const int wheel\_d = 65; // Wheel diameter (mm)

const float wheel\_c = PI \* wheel\_d; // Wheel circumference (mm)

const int counts\_per\_rev = 1920; // (8 pairs N-S) \* (120:1 gearbox) \* (2 falling/rising edges)

const float count\_to\_dist = (wheel\_c / counts\_per\_rev) ; // (1 tick = mm)

const int pwm\_l = 11 ;

const int mot\_l\_1 = 9 ;

const int mot\_l\_2 = 10 ;

const int pwm\_r = 6 ;

const int mot\_r\_1 = 7 ;

const int mot\_r\_2 = 8 ;

const int encoder1PinA = 2;

const int encoder1PinB = 4;

const int encoder2PinA = 3;

const int encoder2PinB = 5;

//Variables

long dist1;

long dist2;

//volatile long avg\_dist;

volatile float avg\_dist\_float;

volatile int avg\_dist;

long prev\_avg\_dist=0;

volatile unsigned long encoder1Pos = 0;

volatile unsigned long encoder2Pos = 0;

volatile int speed1;

volatile int speed2;

volatile bool forward = true;

int currentSpeed = 0;

long startTime = 0;

//const int offset = 5;

//I2C setup

volatile bool action = false ;

volatile bool send\_to\_pic = false;

volatile uint8\_t incomingByte; //unsigned integer of length 8 bits (1 byte)

volatile uint8\_t prev\_incomingByte;

void setup(){

Wire.begin(8); // Join I2C bus with address 8 (slave address)

// Register callback functions

Wire.onReceive(receiveEvent); // Called when this slave device receives a data transmission from master

Wire.onRequest(requestEvent); // Called when master requests data from this slave device

//Motor pin assignments

pinMode(pwm\_l,OUTPUT) ;

pinMode(pwm\_r,OUTPUT) ;

pinMode(mot\_l\_1,OUTPUT) ;

pinMode(mot\_l\_2,OUTPUT) ;

pinMode(mot\_r\_1,OUTPUT) ;

pinMode(mot\_r\_2,OUTPUT) ;

pinMode (encoder1PinA, INPUT);

pinMode (encoder1PinB, INPUT);

pinMode (encoder2PinA, INPUT);

pinMode (encoder2PinB, INPUT);

attachInterrupt(digitalPinToInterrupt(encoder1PinA), count1, RISING);

attachInterrupt(digitalPinToInterrupt(encoder2PinA), count2, RISING);

startTime = millis();

// Open serial port to PC (hardware UART)

Serial.begin(9600);

}

void loop(){

// If we should send to the PIC, then we wait to receive a byte from the PC

if (action) {

set\_speed();

}

if (send\_to\_pic && !action && !incomingByte) { //if sending to PIC, not moving, and the sending byte is not empty

incomingByte = avg\_dist;

Serial.println(incomingByte);

}

}

//void set\_incomingByte

void requestEvent(void){

// prev\_incomingByte = incomingByte;

incomingByte = avg\_dist;

Wire.write(incomingByte); // Respond with message of 1 byte for sensor distance

incomingByte = 0; // Clear output buffer

send\_to\_pic = false;

}

/\*\* @brief Callback for when the master transmits data \*/

void receiveEvent(int){

static uint8\_t buf[3] = {0};

static uint8\_t counter = 0;

uint8\_t x = Wire.read(); // Receive byte

Serial.println((char)x); // Print to serial output as char (ASCII representation)

if (x == '1'){

action = true;

forward = true;

prev\_incomingByte =0;

}

if (x == '5'){

action = true;

forward = false;

prev\_incomingByte =0;

}

if (x == '9'){

action = false; //brake

send\_to\_pic = true;

}

}

void set\_speed() {

//Constrain speed

speed1 = constrain(speed1, 0, 255);

speed2 = constrain(speed2, 0, 255);

//Sample encoders

volatile unsigned long ticks1 = encoder1Pos;

volatile unsigned long ticks2 = encoder2Pos;

Serial.print("Ticks: ");

Serial.print(ticks1);

Serial.print("\t");

Serial.println(ticks2);

Serial.print("Dist: ");

Serial.print(dist1);

Serial.print("\t");

Serial.println(dist2);

int diff = abs(ticks1-ticks2);

int offset = abs(dist1-dist2); ;

if (diff != 0){

if (ticks1>ticks2){

speed1 = speed1 - offset;

speed2 = speed2 + offset;

// Serial.println(" Unveven: ");

}

if (ticks2>ticks1){

speed1 = speed1 + offset;

speed2 = speed2 - offset;

// Serial.println(" Uneven ");

}

}

// else if (ticks1==ticks1){

if (diff == 0){

speed1 = 255;

speed2 = 255;

// Serial.println(" wtf ");

}

Serial.print("Speed: ");

Serial.print(speed1);

Serial.print("\t");

Serial.println(speed2);

// if (forward == true){

drive(speed1, speed2);

// }

// if (forward == false){

//

// }

}

void drive(int speed1, int speed2){

if ((action)&&(forward==true)){

digitalWrite(mot\_l\_1,LOW) ;

digitalWrite(mot\_l\_2,HIGH) ;

digitalWrite(mot\_r\_1,LOW) ;

digitalWrite(mot\_r\_2,HIGH) ;

analogWrite(pwm\_l,speed2) ;

analogWrite(pwm\_r,speed1) ;

}

if ((action)&&(forward==false)){

digitalWrite(mot\_l\_1,HIGH) ;

digitalWrite(mot\_l\_2,LOW) ;

digitalWrite(mot\_r\_1,HIGH) ;

digitalWrite(mot\_r\_2,LOW) ;

analogWrite(pwm\_l,speed2) ;

analogWrite(pwm\_r,speed1) ;

}

if (!action){ //If action is false, brake

digitalWrite(mot\_l\_1,LOW) ;

digitalWrite(mot\_l\_2,LOW) ;

digitalWrite(mot\_r\_1,LOW) ;

digitalWrite(mot\_r\_2,LOW) ;

// Serial.println("brake");

// avg\_dist\_float = (((dist1 + dist2 )/2)/1000); //avg distance is cm

// avg\_dist = (int)(avg\_dist\_float + 0.5); //casting avg distance into an int and rounding

// // avg\_dist = (dist1 );

// Serial.print("Avgdist: ");

// Serial.print(avg\_dist\_float);

// Serial.print("\t");

// Serial.println(avg\_dist);

}

// dist1 = 100;

// dist2 = 23;

// avg\_dist = 90;

}

void count1() {

if (action){

encoder1Pos++;

dist1 = (encoder1Pos \* count\_to\_dist ); //distance in mm

}

}

void count2() {

if (action){

encoder2Pos++;

dist2 = (encoder2Pos \* count\_to\_dist ); //distance in mm

}

}