|  |
| --- |
| #include "MeMCore.h" |
|  | #include <PID\_v1.h> |
|  |  |
|  | /\*\* |
|  | \* Musical notes definition based on values found online |
|  | \*/ |
|  | #define NOTE\_C3 131 |
|  | #define NOTE\_CS3 139 |
|  | #define NOTE\_D3 147 |
|  | #define NOTE\_DS3 156 |
|  | #define NOTE\_E3 165 |
|  | #define NOTE\_F3 175 |
|  | #define NOTE\_FS3 185 |
|  | #define NOTE\_G3 196 |
|  | #define NOTE\_GS3 208 |
|  | #define NOTE\_A3 220 |
|  | #define NOTE\_AS3 233 |
|  | #define NOTE\_B3 247 |
|  | #define NOTE\_C4 262 |
|  |  |
|  | /\*\* |
|  | \* Ultrasonic Sensor definitions |
|  | \* This is used to aid the colour sensor challenge when we are required to do two |
|  | \* successive left turns (orange) or two successive right turns (blue) in two |
|  | \* grids. After turning the first time, mBot will rely on the ultrasonic sensor |
|  | \* and detects if the wall is within a certain distance before turning again |
|  | \* |
|  | \* We have defined the sensing distance ULTRADISTANCE to be 12cm |
|  | \* Initialised the ultrasonic sensor under the variable ultrasonicSensor reading from PORT\_1 |
|  | \*/ |
|  | #define ULTRADISTANCE 12 |
|  |  |
|  | MeUltrasonicSensor ultrasonicSensor(PORT\_1); |
|  |  |
|  |  |
|  |  |
|  | /\*\* |
|  | \* Functions for Music |
|  | \* Store musical notes in an array based on defined numbers found online |
|  | \* Duration that each note is played is also stored in an array |
|  | \* Chosen song: Thug Life, Snoop Dog LOL |
|  | \* |
|  | \* Initialise the on baord buzzer as variable buzzer |
|  | \* melody[] is the array containing the notes of our victory tune |
|  | \* noteDurations[] is the array holding the duration of each note |
|  | \* noteDuration is defined as 1 second divided by the duration of that note. |
|  | \* E.g. A quartet will be 1000/4 |
|  | \* Between each note there is a delay that we experimented to be \*1.1 of the noteDuration |
|  | \* for best effect |
|  | \*/ |
|  | MeBuzzer buzzer; |
|  |  |
|  | int melody[] = {NOTE\_DS3, NOTE\_AS3, NOTE\_AS3, NOTE\_GS3, NOTE\_AS3, NOTE\_GS3, NOTE\_FS3, NOTE\_GS3, NOTE\_GS3, NOTE\_FS3, NOTE\_DS3, NOTE\_FS3}; |
|  | int noteDurations[] = {4, 4, 8, 8, 4, 8, 8, 4, 8, 8, 8, 8}; |
|  |  |
|  | void play() { |
|  | for (int thisNote = 0; thisNote < 12; thisNote++) { |
|  | int noteDuration = 1000 / noteDurations[thisNote]; |
|  | buzzer.tone(8, melody[thisNote], noteDuration); |
|  | int pauseBetweenNotes = noteDuration \* 1.10; |
|  | delay(pauseBetweenNotes); |
|  | // End of music |
|  | buzzer.noTone(8); |
|  | } |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*\* |
|  | \* Functions related to movement corrections |
|  | \* Utilized for movement correction so that robot can stay in a straight line. |
|  | \* Utilizes PID for minor motion adjustments. |
|  | \* However, should IR sensor detect the wall to be too near (<300) |
|  | \* we will utilize another function called extremeIR to correct our movement back |
|  | \* within safe regions |
|  | \*/ |
|  |  |
|  | #define LEFT\_IR A2 |
|  | #define RIGHT\_IR A3 |
|  |  |
|  | /\*\* |
|  | \* Definitions for the PID function to work |
|  | \* setpointLeft and setpointRight is the initial values sensed by the |
|  | \* left and right IR sensor respectively |
|  | \*/ |
|  | double setpointLeft, inputLeft, outputLeft; |
|  | double setpointRight, inputRight, outputRight; |
|  |  |
|  | // Values for Proportional, Integral and Derivative were found via trial and error |
|  | // P = 0.5 accounts for present errors |
|  | // I = 0.01 accounts for past errors |
|  | // D = 0 accounts for future errors |
|  | PID leftPID(&inputLeft, &outputLeft, &setpointLeft, 0.5, 0.01, 0, DIRECT); |
|  | PID rightPID(&inputRight, &outputRight, &setpointRight, 0.5, 0.01, 0, DIRECT); |
|  |  |
|  | /\*\* |
|  | \* setupIRCalibrate function |
|  | \* Used to calibrate the initial left and right distance of the mBot |
|  | \* Takes 10 readings from left and right before calculating average |
|  | \* Average value will be setpointLeft and setpointRight |
|  | \* Upon calibration, PID function is set to AUTOMATIC |
|  | \* |
|  | \* @param[in] inputLeft is the readings from the left IR sensor |
|  | \* @param[in] inputRight is the readings from the right IR sensor |
|  | \* @param[out] setpointLeft is the calibrated base point of the left IR sensor (to use with PID) |
|  | \* @param[out] setpointRight is the calibrated base point of the right IR sensor (to use with PID) |
|  | \*/ |
|  | void setupIRCalibrate() { |
|  | for (int i = 0; i < 10; i++) { |
|  | inputRight = analogRead(RIGHT\_IR); |
|  | inputLeft = analogRead(LEFT\_IR); |
|  | setpointLeft += inputLeft; |
|  | setpointRight += inputRight; |
|  | delay(100); |
|  | } |
|  | setpointLeft /= 10; |
|  | setpointRight /= 10; |
|  | Serial.println(setpointLeft); |
|  | Serial.println(setpointRight); |
|  | // turn PID on |
|  | leftPID.SetMode(AUTOMATIC); |
|  | rightPID.SetMode(AUTOMATIC); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* extremeIR function |
|  | \* Should readings from the left or right sensor be too low, mBot will |
|  | \* manually move towards the right or left to avoid the wall |
|  | \* |
|  | \* @param[in] inputLeft is the readings from the left IR sensor |
|  | \* @param[in] inputRight is the readings from the right IR sensor |
|  | \*/ |
|  | void extremeIR() { |
|  | if (inputLeft < 300) { |
|  | move(1, 255, 165); |
|  | delay(100); |
|  | } |
|  | else if (inputRight < 300) { |
|  | move(1, 165, 255); |
|  | delay(100); |
|  | } |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*\* |
|  | \* Motor functions |
|  | \* Responsible for changing the MeDCMotor values for movement |
|  | \* Functions defined under here are used for any form of movement that is determined by the color challenge or the sound challenge |
|  | \*/ |
|  |  |
|  | //time for turn left or right |
|  | #define TIME\_TURN\_MAX 240.0 |
|  | #define SPEED\_MAX 250.0 |
|  |  |
|  | int speedLeft = SPEED\_MAX; |
|  | int speedRight = SPEED\_MAX; |
|  | MeDCMotor motor1(M1); |
|  | MeDCMotor motor2(M2); |
|  |  |
|  | /\*\* |
|  | \* move function |
|  | \* We designed this function as we realised that in order to move forward or backward |
|  | \* the values for the left and right motor have to be inversed. Eg. Left is positive |
|  | \* and right is negative or vice versa. |
|  | \* As such, utilizing an if else statement at the end, we are able to avoid confusing ourselves |
|  | \* when we wish our mBot to move in a certain direction |
|  | \* We crunched some numbers and decided to change the delay of each turn according to |
|  | \* a constant and the maximum speed of our mBot. |
|  | \* |
|  | \* @param[in] direction is an indication of the direction in which we desire the mBot to movement |
|  | \* @param[in] speedLeft is the speed at which the left motor turns. Usually determined via PID |
|  | \* @param[in] speedRight is the speed at which the right motor turns. Usually determined via PID |
|  | \*/ |
|  | void move(int direction, int speedLeft, int speedRight) { |
|  | int leftSpeed = 0; |
|  | int rightSpeed = 0; |
|  | // 1 is move forward |
|  | // 2 is move backward |
|  | // 3 is turn left |
|  | // 4 is turn right |
|  | if (direction == 1) { |
|  | leftSpeed = speedLeft; |
|  | rightSpeed = speedRight; |
|  | } |
|  | else if (direction == 2) { |
|  | leftSpeed = -speedLeft; |
|  | rightSpeed = -speedRight; |
|  | } |
|  | else if (direction == 3) { |
|  | leftSpeed = -speedLeft; |
|  | rightSpeed = speedRight; |
|  | } |
|  | else if (direction == 4) { |
|  | leftSpeed = speedLeft; |
|  | rightSpeed = -speedRight; |
|  | } |
|  | motor1.run(M1 == M1 ? -(leftSpeed) : (leftSpeed)); |
|  | motor2.run(M2 == M1 ? -(rightSpeed) : (rightSpeed)); |
|  | } |
|  |  |
|  | void turnLeft(int speedLeft, int speedRight) { |
|  | move(3, speedLeft, speedRight); |
|  | delay(TIME\_TURN\_MAX \* SPEED\_MAX / (speedLeft / 2 + speedRight / 2)); |
|  | move(1, speedLeft, speedRight); |
|  | delay(200); |
|  | stop(); |
|  | } |
|  |  |
|  | void turnRight(int speedLeft, int speedRight) { |
|  | move(4, speedLeft, speedRight); |
|  | delay(TIME\_TURN\_MAX \* SPEED\_MAX / (speedLeft / 2 + speedRight / 2)); |
|  | move(1,speedLeft, speedRight); |
|  | delay(100); |
|  | stop(); |
|  | } |
|  |  |
|  | void turn180(int speedLeft, int speedRight) { |
|  | if (analogRead(LEFT\_IR) > analogRead(RIGHT\_IR)) { |
|  | move(3, speedLeft, speedRight); |
|  | delay(2 \* TIME\_TURN\_MAX \* SPEED\_MAX / (speedLeft / 2 + speedRight / 2)); |
|  | stop(); |
|  | } |
|  | else { |
|  | move(4, speedLeft, speedRight); |
|  | delay( 2 \* TIME\_TURN\_MAX \* SPEED\_MAX / (speedLeft / 2 + speedRight / 2)); |
|  | stop(); |
|  | } |
|  | } |
|  |  |
|  | void turnULeft(int speedLeft, int speedRight) { |
|  | turnLeft(speedLeft, speedRight); |
|  | delay(50); |
|  | while (ultrasonicSensor.distanceCm() > ULTRADISTANCE) { |
|  | move(1, speedLeft, speedRight); |
|  | } |
|  | delay(50); |
|  | move(3, speedLeft, speedRight); |
|  | delay(50 + TIME\_TURN\_MAX \* SPEED\_MAX / (speedLeft / 2 + speedRight / 2)); |
|  | stop(); |
|  | } |
|  |  |
|  | void turnURight(int speedLeft, int speedRight) { |
|  | turnRight(speedLeft, speedRight); |
|  | delay(50); |
|  | while (ultrasonicSensor.distanceCm() > ULTRADISTANCE) { |
|  | move(1, speedLeft, speedRight); |
|  | } |
|  | delay(50); |
|  | move(4, speedLeft, speedRight); |
|  | delay(50 + TIME\_TURN\_MAX \* SPEED\_MAX / (speedLeft / 2 + speedRight / 2)); |
|  | stop(); |
|  | } |
|  |  |
|  | void stop() { |
|  | motor1.run(0); |
|  | motor2.run(0); |
|  | } |
|  |  |
|  |  |
|  |  |
|  | /\*\* |
|  | \* Black Line Sensing |
|  | \* Sensor is attached to PORT\_2 of the mBot expansion |
|  | \* Senses if there is a black line |
|  | \* Black line is an indication of a challenge, sound or colour |
|  | \* We set it such that only when both sensor are inside a black line then |
|  | \* the mBot will stop. |
|  | \* |
|  | \* @param[out] returns 1 if black line is sensed. Else it will return 0 |
|  | \*/ |
|  | MeLineFollower linefollower\_2(PORT\_2); |
|  |  |
|  | int isBlackLine() { |
|  | if ((linefollower\_2.readSensors() == S1\_IN\_S2\_IN) ) { |
|  | stop(); |
|  | return 1; |
|  | } |
|  | return 0; |
|  | } |
|  |  |
|  |  |
|  |  |
|  | /\*\* |
|  | \* For Color Sensing Challenge |
|  | \* whiteArray holds measured values for white paper |
|  | \* blackArray holds measured values for black paper |
|  | \* greyArray holds the difference between white and black |
|  | \* colorArray will be used to hold measurements of measured |
|  | \* color paper |
|  | \* |
|  | \* |
|  | \* Green: 50, 133, 50 (Right turn) |
|  | \* Red: 208, 38, 30 (Left turn) |
|  | \* Blue: 45, 130, 160 (Two successive right turns in two grids) |
|  | \* Black: -5 -5 -5 (Check for Ultrasonic/ Victory) |
|  | \* White: 275, 275, 280 (Uturn in one grid) |
|  | \* Orange: 222, 63, 38 (Two successive left turns in two grids) |
|  | \*/ |
|  |  |
|  | // Define time delay before the LED is ON |
|  | #define LED\_RGBWait 20 |
|  | // Define time delay before the next RGB colour turns ON to allow LDR to stabilize |
|  | #define RGBWait 20 |
|  | // Define time delay before taking another LDR reading |
|  | #define LDRWait 10 |
|  | #define TIMES 20 |
|  |  |
|  | MeLightSensor lightsensorTOP(PORT\_6); |
|  | MeRGBLed rgbled(PORT\_7, 2); |
|  |  |
|  | float colourArray[] = {0, 0, 0}; |
|  | float whiteArray[] = {479.00, 334.00, 432.00}; |
|  | float blackArray[] = {232.00, 160.00, 207.00}; |
|  | float greyDiff[] = {246.00, 174.00, 225.00}; |
|  | char colourStr[3][5] = {"R= ", "G= ", "B= "}; |
|  |  |
|  | void setup\_Color\_Challenge() { |
|  | turnOffLed(0); |
|  | setBalance(); |
|  | } |
|  |  |
|  | void setBalance() { |
|  | // Calibrates for whiteArray |
|  | Serial.println("Put White Sample For Calibration ..."); |
|  | buzzer.tone(8, NOTE\_G4, 250); |
|  | delay(5000); // Delay 5 seconds to get white sample ready |
|  | turnOffLed(0); |
|  | for (int i = 0; i <= 2; i++) { |
|  | // Turn on LED Red, Green, Blue at a time |
|  | turnOnOffRGBLed(i, 0); |
|  | whiteArray[i] = getAvgReading(TIMES); //Calibrates whiteArray based on average reading |
|  | turnOffLed(0); |
|  | delay(LED\_RGBWait); |
|  | } |
|  | Serial.println("Put Black Sample For Calibration ..."); |
|  | buzzer.tone(8, NOTE\_C5, 250); |
|  | delay(5000); // Delay 5 seconds to get black sample ready |
|  | for (int i = 0; i <= 2; i++) { |
|  | // Turn on LED Red, Green, Blue at a time |
|  | turnOnOffRGBLed(i, 0); |
|  | blackArray[i] = getAvgReading(TIMES); // Calibrates blackArray based on average reading |
|  | turnOffLed(0); |
|  | delay(LED\_RGBWait); |
|  | // greDiff is the difference between the maximum possible and the minimum possible values |
|  | greyDiff[i] = whiteArray[i] - blackArray[i]; |
|  | } |
|  | Serial.println("Colour Sensor Is Ready."); |
|  | buzzer.tone(8, NOTE\_E5, 250); |
|  | } |
|  |  |
|  | void turnOnOffRGBLed(int i, int light) { |
|  | if (i == 0) { |
|  | turnOnRedLed(light); |
|  | } |
|  | else if (i == 1) { |
|  | turnOnGreenLed(light); |
|  | } |
|  | else { |
|  | turnOnBlueLed(light); |
|  | } |
|  | } |
|  |  |
|  | void turnOnRedLed(int light) { |
|  | rgbled.setColor(light, 255, 0, 0); |
|  | rgbled.show(); |
|  | delay(LED\_RGBWait); |
|  | } |
|  |  |
|  | void turnOnGreenLed(int light) { |
|  | rgbled.setColor(light, 0, 255, 0); |
|  | rgbled.show(); |
|  | delay(LED\_RGBWait); |
|  | } |
|  |  |
|  | void turnOnBlueLed(int light) { |
|  | rgbled.setColor(light, 0, 0, 255); |
|  | rgbled.show(); |
|  | delay(LED\_RGBWait); |
|  | } |
|  |  |
|  | void turnOnWhite(int light) { |
|  | rgbled.setColor(light, 255, 255, 255); |
|  | rgbled.show(); |
|  | delay(LED\_RGBWait); |
|  | } |
|  |  |
|  | void turnOffLed(int light) { |
|  | rgbled.setColor(light, 0, 0, 0); |
|  | rgbled.show(); |
|  | delay(LED\_RGBWait); |
|  | } |
|  |  |
|  | void loopColorChallenge() { |
|  | for (int c = 0; c <= 2; c++) { |
|  | Serial.print(colourStr[c]); |
|  | turnOnOffRGBLed(c, 0); |
|  | colourArray[c] = getAvgReading(TIMES); |
|  | // the average reading returned minus the lowest value divided by the maximum possible range, |
|  | // multiplied by 255 will give a value between 0-255, representing the value for the current reflectivity |
|  | colourArray[c] = (colourArray[c] - blackArray[c]) / (greyDiff[c]) \* 255; |
|  | turnOffLed(0); |
|  | delay(10); |
|  | Serial.println(int(colourArray[c])); |
|  | } |
|  | colorChecker(); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Green: 50, 133, 50 (Right turn) |
|  | \* Red: 208, 38, 30 (Left turn) |
|  | \* Blue: 45, 130, 160 (Two successive right turns in two grids) |
|  | \* Black: -5 -5 -5 (Check for Ultrasonic/ Victory) |
|  | \* White: 275, 275, 280 (Uturn in one grid) |
|  | \* Orange: 222, 63, 38 (Two successive left turns in two grids) |
|  | \*/ |
|  |  |
|  | void colorChecker() { |
|  | // Red |
|  | if ((colourArray[0] > 180) && (colourArray[1] < 45) && (colourArray[2] < 60)) { |
|  | turnLeft(speedLeft, speedRight); |
|  | return; |
|  | } |
|  | // Green |
|  | if ((colourArray[0] < 70) && (colourArray[1] > 100) && (colourArray[2] < 70)) { |
|  | turnRight(speedLeft, speedRight); |
|  | return; |
|  | } |
|  | // Blue |
|  | if ((colourArray[0] < 70) && (colourArray[1] > 100) && (colourArray[2] > 130)) { |
|  | turnURight(speedLeft, speedRight); |
|  | return; |
|  | } |
|  | // White |
|  | if ((colourArray[0] > 200) && (colourArray[1] > 200) && (colourArray[2] > 200)) { |
|  | turn180(speedLeft, speedRight); |
|  | return; |
|  | } |
|  | // Optional Orange |
|  | if ((colourArray[0] > 200) && (colourArray[1] > 45) && (colourArray[2] < 60)) { |
|  | turnULeft(speedLeft, speedRight); |
|  | return; |
|  | } |
|  | // Black |
|  | if ((colourArray[0] < 40) && (colourArray[1] < 40) && (colourArray[2] < 40)) { |
|  | soundChallenge(); |
|  | return; |
|  | } |
|  | } |
|  |  |
|  | // This fucntion is used to get the average reading |
|  | int getAvgReading(int times) { |
|  | int reading; |
|  | int total = 0; |
|  | for (int i = 0; i < times; i++) { |
|  | reading = lightsensorTOP.read(); |
|  | total = reading + total; |
|  | delay(LED\_RGBWait); |
|  | } |
|  | return total / times; |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*\* |
|  | \* For Sound Challenge |
|  | \* Required to read in sound frequencies of 3000Hz and 300Hz from the mic |
|  | \* S2 is 3000Hz |
|  | \* S1 is 300Hz |
|  | \* Read Voltage values from both analogPins and compare their Voltage |
|  | \* If 300Hz louder, Left turn |
|  | \* If 3000Hz louder, Right turn |
|  | \* If both the same amplitude, U-turn same grid |
|  | \*/ |
|  |  |
|  | #define Reader3000Hz A1 |
|  | #define Reader300Hz A0 |
|  |  |
|  | int volt3000, volt300; |
|  |  |
|  | void soundChallenge() { |
|  | float reading\_low = 0; |
|  | float reading\_high = 0; |
|  | for (long i = 0; i < 20; i ++) { |
|  | reading\_low += analogRead(Reader300Hz); |
|  | reading\_high += analogRead(Reader3000Hz); |
|  | delay(50); |
|  | } |
|  | float avghigh = reading\_high/20; //Gets average reading for 3000Hz |
|  | float avglow = reading\_low/20; //Gets average reading for 300Hz |
|  | double ratio = avghigh/(avglow); |
|  | // Numbers are determined through trial and error |
|  | if (ratio > 1) { |
|  | if (ratio <= 5){ |
|  | turnLeft(speedLeft, speedRight); |
|  | } else if ((ratio > 5) && (ratio < 9)) { |
|  | turn180(speedLeft, speedRight); |
|  | } else if (ratio >= 9) { |
|  | turnRight(speedLeft, speedRight); |
|  | } |
|  | } else { |
|  | play(); |
|  | } |
|  | delay(100); |
|  | } |
|  |  |
|  |  |
|  | /\*\* |
|  | \* Main Methods |
|  | \*/ |
|  | void setup() { |
|  | Serial.begin(9600); |
|  | setupIRCalibrate(); |
|  | Serial.println("IR CALIBRATION DONE"); |
|  | //setup\_Color\_Challenge(); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Checks for black line |
|  | \* If no black line, continue moving forward. and correct itself using PID functions |
|  | \* If black line, checks for colour first |
|  | \* If colour is black, checks for sound. |
|  | \* If no sound, play victory music. Else, follow accordingly to certian sound or colour |
|  | \*/ |
|  | void loop() { |
|  | if (isBlackLine() == 1) { |
|  | Serial.println("BLACK LINE!!!"); |
|  | loopColorChallenge(); |
|  | delay(100); |
|  | } |
|  | inputLeft = analogRead(LEFT\_IR); |
|  | inputRight = analogRead(RIGHT\_IR); |
|  | if ((inputLeft < 280) || (inputRight < 280)) { |
|  | extremeIR(); |
|  | } else { |
|  | leftPID.Compute(); |
|  | rightPID.Compute(); |
|  | speedLeft = -(outputRight \* 2.2) + 250; //Base speed of 250, decreased by PID to go in straight line |
|  | speedRight = -(outputLeft \* 2.2) + 250; //Base speed of 250, decreased by PID to go in straight line |
|  | move(1, speedLeft, speedRight); |
|  | } |
|  | } |