Mines Robotics Labs

Generated by Doxygen 1.8.17

Chapter 1

Lab 4 Code Documentation

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

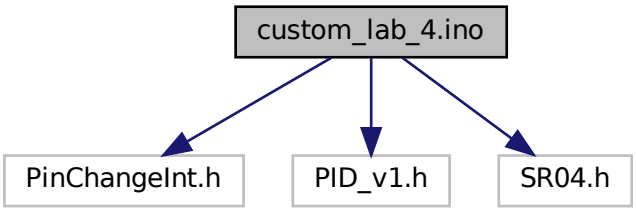
File Documentation

3.1 custom_lab_4.ino File Reference

A basic feedback controlled system for an Arduino based robot with ultrasonic and IR distance sensors.

```
#include <PinChangeInt.h>
#include <PID_v1.h>
#include <SR04.h>
```

Include dependency graph for custom_lab_4.ino:



Macros

- #define IN1 5

  Libraries for interrupts and PID.
- #define IN2 6
- #define IN3 7
- #define IN4 8
- #define A 0

  Motor control.
- #define B 1
- #define pwmA 3
- #define dirA 9
- #define pwmB 4

- #define dirB 13
- #define pushButton 2

  Start stop button.
- #define EncoderCountsPerRev 12.0

  Drive constants - dependent on robot configuration.
- #define DistancePerRev 51.0
- #define DegreesPerRev 27.0
- #define EncoderMotorLeft 7
- #define EncoderMotorRight 8
- #define DISTANCE_SEG 10

  CM.
- #define FORWARD 0

  Enum defines.
- #define RIGHT 1
- #define LEFT 2

Functions

- void resetPWM ()

  Helper function for setting the PWM back to default value.
- void adjustPWM ()

  Run the PID loop calculation and set out put to motors output in PWM.
- void indexLeftEncoderCount ()

  ISR for left encoder.
- void indexRightEncoderCount ()

  ISR for incrementing right encoder.
- void calculateDesiredCount (int distance)

  Calculate how many encoder counts we expect given the distance provided based on the bot intrinsics.
- void calculateDesiredCountTurn (int degrees)

  Calculate how many encoder counts we expect given the degrees provided.
- void turnRight (int degrees)

  Turn bot to given degrees.
- void turnLeft (int degrees)

  Turn bot right to given degrees.
- void driveForward (int distance)

  Function to drive bot forward until encoders are within range.
- void driveBackward (int distance)

  Drive the bot backwards.
- float readDistance (int sensor)

  Function for reading the distance sensors.
- void explore ()

  The exploritory function to allow the system to navigate unseen envioronment Using left hand rule.
- void optimize ()

  This is what youve all been waiting for one darn good looking solution to maze optimation. Iterates over the movesList looking for specific patterns it can reduce into simpler sequences Key assumption: Explored using Left hand rule.
- void configure ()

  Function for configuration of pin states and interrupts.
- void idle ()

  Default behavior when not driving, waits for the pushButton to be pressed so it can execute next command Blocking function.
- void setup ()

Entry point of program handles serial setup and PID config.

- void react_left ()

  This is the logic to execute if we hit a push button ideally this is never executed as we shoudl never actually hit the walls.
- void react_right ()
- void react_forward ()
- void drive ()

  Main drive execution of program, iterates through moves list executing next move with corresponding distance or degrees.
- void loop ()

  Loop execution of the program.

Variables

- double leftEncoderCount = 0

  Lab specific variables.
- double rightEncoderCount = 0
- int wallDist = 5
- int irSensor = A0

  IR sensors.
- int trig = 12

  Ultrasonic sensors.
- int echo = 11
- SR04 sideUS = SR04(trig, echo)
- int motorLeft_PWM = 180

  Default motor pwm values.
- int motorRight_PWM = 200
- int milliSecondsPer90Deg = 900

  Time it takes to move 90 degrees.
- double desiredCount

  How many encoder counts for given distance.
- int movesCount = 0
- int moveList [50]
- int optimizedMoves [50]
- double leftOutput

  PID values setpoints = desired counts, output = PWM, input = current counts.
- double rightOutput

3.1.1 Detailed Description

A basic feedback controlled system for an Arduino based robot with ultrasonic and IR distance sensors.

Author

Christian Prather

Version

0.1

Date

2020-10-23

3.1.2 Macro Definition Documentation

3.1.2.1 A

```
#define A 0
```

Motor control.

Definition at line 29 of file custom_lab_4.ino.

3.1.2.2 B

```
#define B 1
```

Definition at line 30 of file custom_lab_4.ino.

3.1.2.3 DegreesPerRev

```
#define DegreesPerRev 27.0
```

Definition at line 42 of file custom_lab_4.ino.

3.1.2.4 dirA

```
#define dirA 9
```

Definition at line 32 of file custom_lab_4.ino.

3.1.2.5 dirB

```
#define dirB 13
```

Definition at line 34 of file custom_lab_4.ino.

3.1.2.6 DISTANCE_SEG

```
#define DISTANCE_SEG 10
```

CM.

Definition at line 51 of file custom_lab_4.ino.

3.1.2.7 DistancePerRev

```
#define DistancePerRev 51.0
```

Definition at line 41 of file custom_lab_4.ino.

3.1.2.8 EncoderCountsPerRev

```
#define EncoderCountsPerRev 12.0
```

Drive constants - dependent on robot configuration.

Definition at line 40 of file custom_lab_4.ino.

3.1.2.9 EncoderMotorLeft

```
#define EncoderMotorLeft 7
```

Definition at line 44 of file custom_lab_4.ino.

3.1.2.10 EncoderMotorRight

```
#define EncoderMotorRight 8
```

Definition at line 45 of file custom_lab_4.ino.

3.1.2.11 FORWARD

```
#define FORWARD 0
```

Enum defines.

Definition at line 54 of file custom_lab_4.ino.

3.1.2.12 IN1

```
#define IN1 5
```

Libraries for interrupts and PID.

Global Defines Motor driver connections

Definition at line 23 of file custom_lab_4.ino.

3.1.2.13 IN2

```
#define IN2 6
```

Definition at line 24 of file custom_lab_4.ino.

3.1.2.14 IN3

```
#define IN3 7
```

Definition at line 25 of file custom_lab_4.ino.

3.1.2.15 IN4

```
#define IN4 8
```

Definition at line 26 of file custom_lab_4.ino.

3.1.2.16 LEFT

```
#define LEFT 2
```

Definition at line 56 of file custom_lab_4.ino.

3.1.2.17 pushButton

```
#define pushButton 2
```

Start stop button.

Definition at line 37 of file custom_lab_4.ino.

3.1.2.18 pwmA

```
#define pwmA 3
```

Definition at line 31 of file custom_lab_4.ino.

3.1.2.19 pwmB

```
#define pwmB 4
```

Definition at line 33 of file custom_lab_4.ino.

3.1.2.20 RIGHT

```
#define RIGHT 1
```

Definition at line 55 of file custom_lab_4.ino.

3.1.3 Function Documentation

3.1.3.1 adjustPWM()

```
void adjustPWM ( )
```

Run the PID loop calculation and set out put to motors output in PWM.

Definition at line 103 of file custom_lab_4.ino.

```
104 {
105     // Compute the pid values
106     leftPID.Compute();
107     rightPID.Compute();
108 
109     // Set the pid values within range
110     motorLeft_PWM = constrain(leftOutput, 150, 250);
111     motorRight_PWM = constrain(rightOutput, 150, 235);
112     Serial.print("Left PWM: ");
113     Serial.print(motorLeft_PWM);
114     Serial.print(" ");
115     Serial.println(leftEncoderCount);
116     Serial.print("Right PWM: ");
117     Serial.print(motorRight_PWM);
118     Serial.print(" ");
119     Serial.println(rightEncoderCount);
120 }
```

3.1.3.2 calculateDesiredCount()

```
void calculateDesiredCount (
            int distance )
```

Calculate how many encoder counts we expect given the distance provided based on the bot intrinsics.

Parameters

distance	

Definition at line 146 of file custom_lab_4.ino.

```
147 {
148     double revolutionsRequired = distance / DistancePerRev;
149 
150     desiredCount = revolutionsRequired * EncoderCountsPerRev;
151     // Reset encoder counts
152     leftEncoderCount = 0;
153     rightEncoderCount = 0;
154     Serial.print("Desired Count: ");
155     Serial.println(desiredCount);
156 }
```

3.1.3.3 calculateDesiredCountTurn()

```
void calculateDesiredCountTurn (
              int degrees )
```

Calculate how many encoder counts we expect given the degrees provided.

Parameters

degrees	

Definition at line 163 of file custom_lab_4.ino.

```
164 {
165     double revolutionsRequired = degrees / DegreesPerRev;
166     desiredCount = revolutionsRequired * EncoderCountsPerRev;
167     leftEncoderCount = 0;
168     rightEncoderCount = 0;
169     Serial.print("Desired Count: ");
170     Serial.println(desiredCount);
171 }
```

3.1.3.4 configure()

```
void configure ( )
```

Function for configuration of pin states and interrupts.

Definition at line 527 of file custom_lab_4.ino.

```
528 {
529     // set up the motor drive ports
530     pinMode(pwmA, OUTPUT);
531     pinMode(dirA, OUTPUT);
532     pinMode(pwmB, OUTPUT);
533     pinMode(dirB, OUTPUT);
534 
535     pinMode(pushButton, INPUT_PULLUP);
536 
537     pinMode(EncoderMotorLeft, INPUT_PULLUP); //set the pin to input
538     PCintPort::attachInterrupt(EncoderMotorLeft, indexLeftEncoderCount, CHANGE);
539 
540     pinMode(EncoderMotorRight, INPUT_PULLUP); //set the pin to input
541     PCintPort::attachInterrupt(EncoderMotorRight, indexRightEncoderCount, CHANGE);
542 }
```

3.1.3.5 drive()

```
void drive ( )
```

Main drive execution of program, iterates through moves list executing next move with corresponding distance or degrees.

Definition at line 599 of file custom_lab_4.ino.

```
600 {
601     // Iterate over the list jumping by two each time
602     for (int i = 0; i < sizeof(optimizedMoves); i += 2)
603     {
604         idle();
605         switch (moveList[i])
606         {
607         case LEFT:
608             turnLeft(moveList[i + 1]);
609             break;
610         case RIGHT:
611             turnRight(moveList[i + 1]);
612             break;
613         case FORWARD:
614             driveForward(moveList[i + 1]);
615             break;
616         default:
617             break;
618         }
619     }
620
621 }
```

3.1.3.6 driveBackward()

```
void driveBackward (
              int distance )
```

Drive the bot backwards.

Parameters

distance	

Definition at line 290 of file custom_lab_4.ino.

```
291 {
292     resetPWM();
293     calculateDesiredCount(distance);
294
295     // Loop unitl the encoders read correct
296
297     while ((desiredCount - leftEncoderCount) > 3 || (desiredCount - rightEncoderCount) > 3)
298     {
299         adjustPWM();
300         //To drive backward, motors go in the same direction
301
302         if ((desiredCount - leftEncoderCount) > 3)
303         {
304             run_motor(A, motorLeft_PWM); //change PWM to your calibrations
305         }
306         if ((desiredCount - rightEncoderCount) > 3)
307         {
308             run_motor(B, motorRight_PWM); //change PWM to your calibrations
309         }
310     }
311
312     // motors stop
313     run_motor(A, 0);
314     run_motor(B, 0);
315     Serial.println("Done driving backwards");
```

```
316     Serial.print("L: ");
317     Serial.println(leftEncoderCount);
318     Serial.print("R: ");
319     Serial.println(rightEncoderCount);
320 }
```

3.1.3.7 driveForward()

```
void driveForward (
            int distance )
```

Function to drive bot forward until encoders are within range.

Parameters

distance	

Definition at line 252 of file custom_lab_4.ino.

```
253 {
254     Serial.println("Driving Forward...");
255     resetPWM();
256     calculateDesiredCount(distance);
257 
258     // Loop unitl the encoders read correct
259 
260     while ((desiredCount - leftEncoderCount) > 3 || (desiredCount - rightEncoderCount) > 3)
261     {
262         adjustPWM();
263         //To drive forward, motors go in the same direction
264 
265         if ((desiredCount - leftEncoderCount) > 3)
266         {
267             run_motor(A, -motorLeft_PWM); //change PWM to your calibrations
268         }
269         if ((desiredCount - rightEncoderCount) > 3)
270         {
271             run_motor(B, -motorRight_PWM); //change PWM to your calibrations
272         }
273     }
274 
275     // motors stop
276     run_motor(A, 0);
277     run_motor(B, 0);
278     Serial.println("Done driving forward");
279     Serial.print("L: ");
280     Serial.println(leftEncoderCount);
281     Serial.print("R: ");
282     Serial.println(rightEncoderCount);
283 }
```

3.1.3.8 explore()

```
void explore ( )
```

The exploritory function to allow the system to navigate unseen envioronment Using left hand rule.

There is no wall to left of bot

Not recording degrees as the assumption is every turn on 90 degrees

Can drive forward

Trapped turn Right

Definition at line 351 of file custom_lab_4.ino.

```
352 {
353     while (digitalRead(pushButton) == 1)
354     {
355         float front = readDistance(0);
356         float side = readDistance(1);
357 
359         if (side > wallDist)
360         {
361             turnLeft(90);
363             moveList[movesCount] = "LEFT";
364             movesCount++;
365         }
367         else if (front > wallDist)
368         {
369             driveForward(DISTANCE_SEG);
370             moveList[movesCount] = "FORWARD";
371             movesCount++;
372         }
374         else
375         {
376             turnRight(90);
377             moveList[movesCount] = "RIGHT";
378             movesCount++;
379         }
380     }
381 }
```

3.1.3.9 idle()

```
void idle ( )
```

Default behavior when not driving, waits for the pushButton to be pressed so it can execute next command Blocking function.

Definition at line 549 of file custom_lab_4.ino.

```
550 {
551     Serial.println("Idle..");
552     while (digitalRead(pushButton) == 1)
553         ; // wait for button push
554     while (digitalRead(pushButton) == 0)
555         ;        // wait for button release
556     delay(2000); // Give time to move hand
557 }
```

3.1.3.10 indexLeftEncoderCount()

```
void indexLeftEncoderCount ( )
```

ISR for left encoder.

Definition at line 125 of file custom_lab_4.ino.

```
126 {
127     leftEncoderCount++;
128     //Serial.println("Left Encoder ++");
129 }
```

3.1.3.11 indexRightEncoderCount()

```
void indexRightEncoderCount ( )
```

ISR for incrementing right encoder.

Definition at line 134 of file custom_lab_4.ino.

```
135 {
136     rightEncoderCount++;
137     //Serial.println("Right Encoder ++");
138 }
```

3.1.3.12 loop()

```
void loop ( )
```

Loop execution of the program.

Definition at line 626 of file custom_lab_4.ino.

```
627 {
628     explore();
629     optimize();
630     drive();
631 }
```

3.1.3.13 optimize()

```
void optimize ( )
```

This is what youve all been waiting for one darn good looking solution to maze optimation. Iterates over the moves↩ List looking for specific patterns it can reduce into simpler sequences Key assumption: Explored using Left hand rule.

Key patterns 0 = F, 1 = R, 2 = L, 3 = DELETE

This is going to be checking in a priority tree fashion given highest priority given highest priority patterns are 6 long then 5 long then 4 I can batch this

Get next move in explored list

Get next 6 moves if enough in list

Definition at line 389 of file custom_lab_4.ino.

```
390 {
392     int keyPatterns_6[2][6] = {{0, 0, 1, 1, 0, 0}, {2, 0, 1, 1, 0, 2}};
393     int keyPatterns_5[2][5] = {{2, 0, 1, 1, 0}, {0, 1, 1, 0, 2}};
394     int keyPatterns_4[1][4] = {{0, 1, 1, 0}};
395
396     int optimizedPattern_6[1][8] = {{FORWARD, 2 * DISTANCE_SEG, RIGHT, 90, RIGHT, 90, FORWARD,
      DISTANCE_SEG}};
397     int optimizedPattern_5[2][2] = {{RIGHT, 90}, {RIGHT, 90}};
398     int optimizedPatter_4[1][4] = {{LEFT, 90, LEFT, 90}};
402     for (int i = 0; i < movesCount; i++)
403     {
405         // int move = moveList[i];
407
408         // Check 6 out first
409         int future[6];
410         for (int j = 0; j < 6; j++)
```

```
411         {
412             if ((j + i) < movesCount)
413             {
414                 future[j] = moveList[j + i];
415             }
416         }
417         int tracker = 0;
418         for (auto potential : keyPatterns_6)
419         {
420             bool match = true;
421             for (int m = 0; m < 6; m++)
422             {
423                 if (future[m] != potential[m])
424                 {
425                     match = false;
426                 }
427             }
428             if (match)
429             {
430                 int keyPatternLength = (sizeof(potential) / sizeof(potential[0]));
431                 // Insert optimized move
432                 for (int x = 0; x < (sizeof(optimizedPattern_6[tracker]) /
    sizeof(optimizedPattern_6[tracker][0])); x++)
433                 {
434                     if (optimizedPattern_6[tracker][x] != 3)
435                     {
436                         optimizedMoves[x] = optimizedPattern_6[tracker][x];
437                     }
438                 }
439                 i = i + 6;
440                 break;
441             }
442             tracker = tracker + 1;
443         }
444 
445         // Check 5 out first
446         int future_5[5];
447         for (int j = 0; j < 5; j++)
448         {
449             if ((j + i) < movesCount)
450             {
451                 future_5[j] = moveList[j + i];
452             }
453         }
454         tracker = 0;
455         for (auto potential : keyPatterns_6)
456         {
457             bool match = true;
458             for (int m = 0; m < 5; m++)
459             {
460                 if (future_5[m] != potential[m])
461                 {
462                     match = false;
463                 }
464             }
465             if (match)
466             {
467                 int keyPatternLength = (sizeof(potential) / sizeof(potential[0]));
468                 // Insert optimized move
469                 for (int x = 0; x < (sizeof(optimizedPattern_6[tracker]) /
    sizeof(optimizedPattern_6[tracker][0])); x++)
470                 {
471                     if (optimizedPattern_6[tracker][x] != 3)
472                     {
473                         optimizedMoves[x] = optimizedPattern_6[tracker][x];
474                     }
475                 }
476                 i = i + 5;
477                 break;
478             }
479             tracker = tracker + 1;
480         }
481 
482         // Check 4 out first
483         int future_4[4];
484         for (int j = 0; j < 4; j++)
485         {
486             if ((j + i) < movesCount)
487             {
488                 future_4[j] = moveList[j + i];
489             }
490         }
491         tracker = 0;
492         for (auto potential : keyPatterns_6)
493         {
494             bool match = true;
495             for (int m = 0; m < 4; m++)
496 
497 
```

```
498             {
499                 if (future_4[m] != potential[m])
500                 {
501                     match = false;
502                 }
503             }
504             if (match)
505             {
506                 int keyPatternLength = (sizeof(potential) / sizeof(potential[0]));
507                 // Insert optimized move
508                 for (int x = 0; x < (sizeof(optimizedPattern_6[tracker]) /
      sizeof(optimizedPattern_6[tracker][0])); x++)
509                 {
510                     if (optimizedPattern_6[tracker][x] != 3)
511                     {
512                         optimizedMoves[x] = optimizedPattern_6[tracker][x];
513                     }
514                 }
515                 i = i + 4;
516                 break;
517             }
518             tracker = tracker + 1;
519         }
520     }
521 }
```

3.1.3.14 react_forward()

```
void react_forward ( )
```

Definition at line 589 of file custom_lab_4.ino.

```
590 {
591     // TODO: Check which button was hit
592     driveBackward(50);
593 }
```

3.1.3.15 react_left()

```
void react_left ( )
```

This is the logic to execute if we hit a push button ideally this is never executed as we shoudl never actually hit the walls.

Definition at line 575 of file custom_lab_4.ino.

```
576 {
577     // TODO: Check which button was hit
578 
579     driveBackward(20);
580     turnRight(30);
581 }
```

3.1.3.16 react_right()

```
void react_right ( )
```

Definition at line 582 of file custom_lab_4.ino.

```
583 {
584     // TODO: Check which button was hit
585 
586     driveBackward(20);
587     turnLeft(30);
588 }
```

3.1.3.17 readDistance()

```
float readDistance (
            int sensor )
```

Function for reading the distance sensors.

Parameters

sensor	0 = IR, 1 = Ultrasonic

Returns

float distance (cm)

Definition at line 328 of file custom_lab_4.ino.

```
329 {
330     float distance = 0.0;
331     switch (sensor)
332     {
333     case 0:
334         int reading = analogRead(irSensor);
335         distance = ((0.00031) * reading) + 0.002;
336         break;
337     case 1:
338         distance = sideUS.Distance();
339         break;
340
341     default:
342         break;
343     }
344     return distance;
345 }
```

3.1.3.18 resetPWM()

```
void resetPWM ( )
```

Helper function for setting the PWM back to default value.

Definition at line 93 of file custom_lab_4.ino.

```
94 {
95     motorLeft_PWM = 180;
96     motorRight_PWM = 200;
97 }
```

3.1.3.19 setup()

```
void setup ( )
```

Entry point of program handles serial setup and PID config.

Definition at line 562 of file custom_lab_4.ino.

```
563 {
564     Serial.begin(9600);
565     Serial.println("Setting up.....");
566     configure();
567     leftPID.SetMode(AUTOMATIC);
568     rightPID.SetMode(AUTOMATIC);
569 }
```

3.1.3.20 turnLeft()

```
void turnLeft (
            int degrees )
```

Turn bot right to given degrees.

Parameters

degrees	

Definition at line 215 of file custom_lab_4.ino.

```
216 {
217     resetPWM();
218     calculateDesiredCountTurn(degrees);
219 
220     // Loop unitl the encoders read correct
221 
222     while ((desiredCount - leftEncoderCount) > 3)
223     {
224         adjustPWM();
225         //To drive forward, motors go in the same direction
226 
227         if ((desiredCount - leftEncoderCount) > 3)
228         {
229             run_motor(A, motorLeft_PWM); //change PWM to your calibrations
230         }
231         if ((desiredCount - rightEncoderCount) > 3)
232         {
233             run_motor(B, -motorRight_PWM); //change PWM to your calibrations
234         }
235     }
236 
237     // motors stop
238     run_motor(A, 0);
239     run_motor(B, 0);
240     Serial.println("Done driving Left");
241     Serial.print("L: ");
242     Serial.println(leftEncoderCount);
243     Serial.print("R: ");
244     Serial.println(rightEncoderCount);
245 }
```

3.1.3.21 turnRight()

```
void turnRight (
            int degrees )
```

Turn bot to given degrees.

Parameters

degrees	

Definition at line 178 of file custom_lab_4.ino.

```
179 {
180     resetPWM(); // Reset pwm
181     calculateDesiredCountTurn(degrees);
182     // While the encoders are not correct adjust PWM with PID loop
183     // Loop unitl the encoders read correct
184 
185     while ((desiredCount - rightEncoderCount) > 3)
186     {
187         adjustPWM();
```

```
188         //To drive forward, motors go in the same direction
189
190         if ((desiredCount - leftEncoderCount) > 3)
191         {
192             run_motor(A, -motorLeft_PWM); //change PWM to your calibrations
193         }
194         if ((desiredCount - rightEncoderCount) > 3)
195         {
196             run_motor(B, motorRight_PWM); //change PWM to your calibrations
197         }
198     }
199
200     // motors stop
201     run_motor(A, 0);
202     run_motor(B, 0);
203     Serial.println("Done driving Right");
204     Serial.print("L: ");
205     Serial.println(leftEncoderCount);
206     Serial.print("R: ");
207     Serial.println(rightEncoderCount);
208 }
```

3.1.4 Variable Documentation

3.1.4.1 desiredCount

```
double desiredCount
```

How many encoder counts for given distance.

Definition at line 74 of file custom_lab_4.ino.

3.1.4.2 echo

```
int echo = 11
```

Definition at line 63 of file custom_lab_4.ino.

3.1.4.3 irSensor

```
int irSensor = A0
```

IR sensors.

Definition at line 59 of file custom_lab_4.ino.

3.1.4.4 leftEncoderCount

```
PID leftPID & leftEncoderCount = 0
```

Lab specific variables.

Definition at line 48 of file custom_lab_4.ino.

3.1.4.5 leftOutput

```
double leftOutput
```

PID values setpoints = desired counts, output = PWM, input = current counts.

Definition at line 86 of file custom_lab_4.ino.

3.1.4.6 milliSecondsPer90Deg

```
int milliSecondsPer90Deg = 900
```

Time it takes to move 90 degrees.

Definition at line 71 of file custom_lab_4.ino.

3.1.4.7 motorLeft_PWM

```
int motorLeft_PWM = 180
```

Default motor pwm values.

Definition at line 67 of file custom_lab_4.ino.

3.1.4.8 motorRight_PWM

```
int motorRight_PWM = 200
```

Definition at line 68 of file custom_lab_4.ino.

3.1.4.9 moveList

```
int moveList[50]
```

Definition at line 78 of file custom_lab_4.ino.

3.1.4.10 movesCount

```
int movesCount = 0
```

Definition at line 76 of file custom_lab_4.ino.

3.1.4.11 optimizedMoves

```
int optimizedMoves[50]
```

Definition at line 80 of file custom_lab_4.ino.

3.1.4.12 rightEncoderCount

```
PID rightPID & rightEncoderCount = 0
```

Definition at line 49 of file custom_lab_4.ino.

3.1.4.13 rightOutput

```
double rightOutput
```

Definition at line 86 of file custom_lab_4.ino.

3.1.4.14 sideUS

```
SR04 sideUS = SR04(trig, echo)
```

Definition at line 64 of file custom_lab_4.ino.

3.1.4.15 trig

```
int trig = 12
```

Ultrasonic sensors.

Definition at line 62 of file custom_lab_4.ino.

3.1.4.16 wallDist

```
int wallDist = 5
```

Definition at line 50 of file custom_lab_4.ino.

3.2 motors.ino File Reference

Functions

- void motor_setup ()
- void run_motor (int motor, int pwm)

3.2.1 Function Documentation

3.2.1.1 motor_setup()

```
void motor_setup ( )
```

Definition at line 1 of file motors.ino.

```
2 {
3   // if using dual motor driver
4   // define driver pins as outputs
5   pinMode(IN1, OUTPUT);
6   pinMode(IN2, OUTPUT);
7   pinMode(IN3, OUTPUT);
8   pinMode(IN4, OUTPUT);
9   // initialize all pins to zero
10   digitalWrite(IN1, 0);
11   digitalWrite(IN2, 0);
12   digitalWrite(IN3, 0);
13   digitalWrite(IN4, 0);
14   return;
15 } // end function
```

3.2.1.2 run_motor()

```
void run_motor (
              int motor,
              int pwm )
```

Definition at line 19 of file motors.ino.

```
20 {
21   int dir = (pwm / abs(pwm)) > 0; // returns if direction is forward (1) or reverse (0)
22   pwm = abs(pwm);                 // only positive values can be sent to the motor
23
24   switch (motor)
25   {       // find which motor to control
26   case A: // if A, write A pins
27     if (dir)
28     {                          // If dir is forward
29       analogWrite(IN1, pwm);   // IN1 is the forward pwm pin
30       digitalWrite(IN2, LOW); // IN2 is low
31     }
32     else
33     {
34       digitalWrite(IN1, LOW); // IN1 is low
35       analogWrite(IN2, pwm);   // IN2 is the reverse pwm pin
36     }                          // end if
37     break;                     // end case A
38   case B:                      // if B, write B pins
39     if (dir)
40     {                          // if dir is forward
41       analogWrite(IN3, pwm);   // IN3 is the forward pwm pin
42       digitalWrite(IN4, LOW); // IN4 is low
43     }
44     else
45     {
46       digitalWrite(IN3, LOW); //IN3 is low
47       analogWrite(IN4, pwm);   // IN4 is the reverse pwm pin
48     }                          // end if
49     break;                     // end case B
50   }                            // end switch case
51   return;
52 } // end function
```

Index