Aero Group A2-2 Odometry Code

Code written by Duncan Hamill, Tom Griffiths, Ali Hajizadah, Robin Hannaford, and Felix Harris.

Odometry.ino - Arduino logic

```
Odometry Task - Aero 2 Group 2
 3
 4
                 Code written by Duncan R Hamill - 28262174
 5
                 Tested & verified by Tom Griffiths - 28290771, Ali Hajizadah - 29053056, Robin Hannaford -
 6
 7
                All distances in mm, all angles in degrees
8
     */
9
10
    #include "defines.h"
    #include "encoderInteraction.cpp"
11
    #include "Course.cpp"
12
13
14
    #include <Wire.h>
15
    #include <Servo.h>
16
    void setup() {
17
18
        // start serial for monitoring
19
        Serial.begin(9600);
20
21
        // setup the I2C and wait 100ms
22
        Wire.begin();
23
        delay(100);
24
25
        // zero the encoders
26
        resetEncoders();
27
28
        // setup servo pointers
29
        Servo* servo_ptr;
        servo_ptr->attach(SERVOPIN);
30
31
        int ServoPosition = SERVOINIT;
32
        servo_ptr->write(ServoPosition);
33
34
        // initialise the course
35
        Course course = Course(servo_ptr, &ServoPosition);
36
37
        // wait a bit before we start
38
        delay(1000);
39
40
        // TESTING - loop through legs and run their action \,
41
        //course.run();
42
        Circle c = Circle(150, 180, FORWARD, 0, false);
43
44
        c.run();
45
46
        // turn on for event
47
        //finished();
48
49
    }
50
51
    // just a bit of fun
52
    void finished() {
53
        tone(9,660,100);
54
        delay(150);
```

```
55
        tone(9,660,100);
56
        delay(300);
57
        tone(9,660,100);
58
        delay(300);
59
        tone(9,510,100);
60
        delay(100);
61
        tone(9,660,100);
62
        delay(300);
63
        tone(9,770,100);
64
        delay(550);
65
        tone(9,380,100);
66
        delay(575);
67
    }
68
69
    void loop() {
70
71
    }
    Course.cpp - Logic for completing the course
 1
 2
       Course list - includes Legs variable that stores how to run the course
 3
 4
 5
    #include "defines.h"
 6
    #include "Leg.cpp"
 7
    #include "Line.cpp"
    #include "Circle.cpp"
8
9
10
    // include guard
11
    #ifndef COURSE_CPP
    #define COURSE_CPP
12
13
14
    class Course {
15
      public:
16
        Leg** legs;
17
18
        Course(Servo* servo_ptr, int* servoPosition_ptr) {
19
            legs = new Leg*[13];
20
21
22
            * ---- THE LEG CODE ----
23
24
            * Each leg represents a part of the course, with the following parameters
25
                   Line - Distance, Direction, Angle to turn at end, drop M&M
26
                 Circle - Radius, angle to move through, direction, angle to turn at end, drop M&M
27
            */
28
            legs[0] = new Line(428, FORWARD, 0, true);
29
            legs[1] = new Line(504, FORWARD, -37, false);
30
            legs[2] = new Circle(180, 270, BACKWARD, 90, true);
31
            legs[3] = new Line(180, FORWARD, -40, false);
32
            legs[4] = new Line(622, BACKWARD,50, true);
33
            legs[5] = new Line(400, BACKWARD, -90, false);
34
            legs[6] = new Line(400, FORWARD, 90, true);
35
            legs[7] = new Line(400, FORWARD, 90, false);
36
            legs[8] = new Line(660, FORWARD, 90, true);
37
            legs[9] = new Circle(260, 90, FORWARD, -90, false);
38
            legs[10] = new Line(500, FORWARD, 90, false);
39
            legs[11] = new Line(260, FORWARD, 90, false);
40
            legs[12] = new Line(340, FORWARD, 143, false);
41
```

Page **2** of **12**

```
42
            for (int i = 0; i < 13; i++) {</pre>
                 legs[i]->servo = servo_ptr;
43
44
                 legs[i]->servoPosition = servoPosition_ptr;
45
            }
        }
46
47
48
        void run() {
49
            for (int i = 0; i < 13; i++) {</pre>
                 Serial.print("Running leg ");
50
51
                 Serial.println(i);
52
53
                 legs[i]->run();
            }
54
55
        }
56
    };
57
58
    #endif
    Leg.cpp - defines code for completing a section of the course
1
 2
        Leg class - controls the robot for one 'leg' (segment) of the course
 3
                     includes logic for the action, rotate, and stop functions
 4
     */
 5
 6
    #include "defines.h"
 7
    #include "encoderInteraction.cpp"
8
9
    #include <Arduino.h>
   #include <Servo.h>
10
11
    #include <Wire.h>
12
13
   // include guard
14
   #ifndef LEG_CPP
15
   #define LEG_CPP
16
17
    class Leg
18
19
      public:
20
        // pointer to serveo position variable
21
        int* servoPosition;
22
        Servo* servo;
23
24
        // Should we drop an M&M?, leg finished successfully?
25
        bool drop, direction;
26
27
        // Virtual function that will be called to run this leg of the course.
28
        virtual void run();
29
30
        // perform actions at waypoint, including dropping M&M if needed
31
        void action() {
32
            // turn on LED and buzzer
33
            digitalWrite(LEDPIN, HIGH);
34
            tone(PIEZOPIN, PIEZOFREQ);
35
36
            // if need to drop M&M, drop one, if not delay so we can see and hear buzzer
37
            if (this->drop) { this->dispense(); }
38
            else { delay(NOTIFYPAUSE); }
39
40
            // turn off led & buzzer
41
            digitalWrite(LEDPIN, LOW);
```

Page **3** of **12**

```
42
             noTone(PIEZOPIN);
43
         }
44
45
         // dispense an M&M
46
         void dispense() {
47
             // increase servo position
48
             *servoPosition += SERVOSTEP;
49
50
             // make sure we don't accidentally run through all positions
             if (*servoPosition >= 179) {
51
52
                 *servoPosition = 179;
53
54
55
             // write the servo position and wait to ensure clean drop
56
             servo->write(*servoPosition);
57
             delay(SERVOPAUSE);
58
         }
59
60
         // rotate by the given angle (+ve clockwise), returning the actual angle rotated
         float rotate(int t, bool correction) {
61
62
             Serial.print("Rotate ");
63
64
             Serial.print(t);
65
66
             // find distance needed to rotate
67
             int dist = (int)(2 * PI * WHEELDIST * ((float)abs(t) / (float)360));
68
69
             Serial.print(dist);
 70
             Serial.print(" ");
 71
 72
             // speeds of each wheel
73
             int leftWheel, rightWheel, rotateSpeed;
74
75
             // if we're in correction mode rotate slower
76
             if (correction) {
 77
                 rotateSpeed = DUALSPEED * 0.1;
78
             } else {
79
                 rotateSpeed = DUALSPEED * 0.5;
80
             }
81
82
             Serial.print((int)rotateSpeed);
83
84
             // set speeds of each wheel depending on direction (+ve -> left goes forwards)
85
             if (t > 0) {
                 leftWheel = (128 + rotateSpeed);
86
87
                 rightWheel = (128 - rotateSpeed);
88
             } else {
89
                 leftWheel = 128 - rotateSpeed;
90
                 rightWheel = 128 + rotateSpeed;
91
             }
92
93
             Serial.print(" ");
94
             Serial.print((int)leftWheel);
95
             Serial.print(" ");
             Serial.println((int)rightWheel);
96
97
98
             // while we've not rotated less that the required distance
99
             while (averageDistance() <= dist) {</pre>
100
                 // set wheels to spin at different speeds
101
                 Wire.beginTransmission(MD25ADDR);
102
                 Wire.write(MODE);
```

```
103
                  Wire.write(MODESEPERATE);
104
                  Wire.endTransmission();
105
106
                  // set the acceleration mode to fast
107
                  Wire.beginTransmission(MD25ADDR);
108
                  Wire.write(ACCEL);
109
                  Wire.write(ACCELDEFAULT);
110
                  Wire.endTransmission();
111
112
                  // Set left wheel speed
113
                  Wire.beginTransmission(MD25ADDR);
114
                  Wire.write(SPEEDLEFT);
115
                  Wire.write((char)leftWheel);
116
                  Wire.endTransmission();
117
118
                  // set right wheel speed
119
                  Wire.beginTransmission(MD25ADDR);
120
                  Wire.write(SPEEDRIGHT);
121
                  Wire.write((char)rightWheel);
122
                  Wire.endTransmission();
123
             }
124
125
             long avg = (long)averageDistance();
126
127
             Serial.print(avg);
             Serial.print(" ");
128
129
130
             float ang = (int)((float)(360 * avg)/((float)(2 * PI * WHEELDIST)));
131
132
             Serial.print(ang);
133
             Serial.print(" ");
134
135
             resetEncoders();
136
             this->stop();
137
             delay(50);
138
             return ang;
139
140
141
         // stop the vehicle
142
         void stop() {
143
             // allow both registers to be set to stop
144
             Wire.beginTransmission(MD25ADDR);
145
             Wire.write(MODE);
             Wire.write(MODESEPERATE);
146
147
             Wire.endTransmission();
148
149
             // high acceleration mode
150
             Wire.beginTransmission(MD25ADDR);
151
             Wire.write(ACCEL);
152
             Wire.write(10);
153
             Wire.endTransmission();
154
155
             // set left to stop
156
             Wire.beginTransmission(MD25ADDR);
             Wire.write(SPEEDLEFT);
157
158
             Wire.write(128);
159
             Wire.endTransmission();
160
161
             // set right to stop
162
             Wire.beginTransmission(MD25ADDR);
163
             Wire.write(SPEEDRIGHT);
```

Page **5** of **12**

```
164
             Wire.write(128);
165
             Wire.endTransmission();
166
             delay(50);
167
         }
168
    };
169
170
    #endif
     Line.cpp - logic for driving a straight line
 1
      st Line class - drives a straight leg of the course
 3
      */
 4
 5
     #include "defines.h"
     #include "encoderInteraction.cpp"
 6
 7
    #include "Leg.cpp"
 8
 9
    #include <Arduino.h>
    #include <Wire.h>
10
11
12
    // include guard
    #ifndef LINE_CPP
13
14
    #define LINE_CPP
15
16
    class Line: public Leg {
17
         // count how many times we loop over the drive sections, so we don't get stuck.
18
         int loopCount;
19
20
         // ramp function to increase speed over course of a line
21
         int ramp(int max, int dist, int x) {
22
             int offset = (dist / 2) - x;
23
             return (max - abs(offset));
24
         }
25
26
      public:
27
         // distance to travel, and how far to rotate to be pointing in correct direction at end of the leg
28
         int dist, endRot;
29
30
         // constructor
31
         Line(int d, int dir, int r, bool m) {
32
             this->dist = d;
33
             this->direction = dir;
34
             this->endRot = r;
35
             this->drop = m;
36
             this->loopCount = 0;
37
         }
38
39
         // implement the run function
40
         void run() {
41
42
             // run drive, get how far we actually drove
             int driven = this->drive(this->direction * this->dist, false);
43
44
45
             // calculate distance left to drive
46
             int shortfall = this->dist - driven;
47
48
             // aim to get within 2mm of the target waypoint, without going over MAXLOOPCOUNT
49
             while (abs(shortfall) > LINEARTOL && loopCount < MAXLOOPCOUNT) {</pre>
50
                 // if we aren't on target, drive the shortfall again, looping over to check we reached it
51
                 driven = this->drive(shortfall, true);
```

6 Page **6** of **12**

```
52
                  this->stop();
53
                  shortfall = abs(shortfall) - driven;
54
                  this->loopCount++;
55
56
57
             this->loopCount = 0;
58
59
             // now repeat this for rotation
60
             float rotated = this->rotate(this->endRot, false);
61
62
             float rotShortfall = this->endRot - rotated;
63
64
             while (abs(rotShortfall) > ANGULARTOL && loopCount < MAXLOOPCOUNT) {</pre>
65
                  Serial.println("Correcting rotation");
66
                  rotated = this->rotate(rotShortfall, true);
67
                  this->stop();
68
                  rotShortfall = abs(rotShortfall) - rotated;
69
                  this->loopCount++;
70
             }
71
             this->loopCount = 0;
 72
73
             // blink light, sound buzzer, and drop M&M if needed
74
             this->action();
75
         }
 76
77
         // move the wheels the desired distance, and return the actual distance driven
 78
         int drive(int d, bool correction) {
             Serial.print("Line ");
 79
80
             if (d == 0) {
81
82
                 return 0;
83
             }
84
85
             while(int avgD = averageDistance() <= abs(d)) {</pre>
86
                  int spd;
87
88
                  // if in a correction, go slowly for more accuracy, else increase speed over course of a li
89
                  if (correction) {
90
                      spd = DUALSPEED * 0.2;
91
92
                      spd = ramp(DUALSPEED, abs(d), avgD);
93
94
95
                  // Set both wheels to spin at the same rate
96
                  Wire.beginTransmission(MD25ADDR);
97
                  Wire.write(MODE);
98
                  Wire.write(MODEUNSIGNEDDUAL);
99
                  Wire.endTransmission();
100
101
                  // set the acceleration mode to fast
102
                  Wire.beginTransmission(MD25ADDR);
103
                  Wire.write(ACCEL);
104
                 Wire.write(ACCELDEFAULT);
105
                 Wire.endTransmission();
106
                  // set the speed
107
108
                  Wire.beginTransmission(MD25ADDR);
109
                  Wire.write(SPEEDLEFT);
110
111
                  // if we're given a negative distance, drive backwards
112
                  if (d < 0) {</pre>
```

```
113
                     Wire.write((char)(128 - spd));
114
                 } else {
115
                     Wire.write((char)(128 + spd));
116
                 }
117
                 Wire.endTransmission();
118
             }
119
120
             // return the read distance
121
             int avg = averageDistance();
122
             Serial.println(avg);
123
             resetEncoders();
124
             this->stop();
125
             delay(50);
126
             return avg;
127
         }
128
    };
129
130
    #endif
     Circle.cpp - logic for driving an arc of the course
 1
 2
        Circle class - contains logic for driving a circular section of the course
 3
 4
 5
    #include "defines.h"
 6
    #include "encoderInteraction.cpp"
 7
    #include "Leg.cpp"
 8
 9
     #include <Arduino.h>
10
    #include <Wire.h>
11
12
    // include guard
13
    #ifndef CIRCLE_CPP
14
    #define CIRCLE_CPP
15
16
    class Circle: public Leg {
17
         // loop counter to ensure we don't get stuck in a loop, distance the outer wheel has to rotate
18
         int loopCount, direction, outerDist, innerDist;
19
      public:
20
         // radius of the circle, angular distance to travel, final rotation for next leg
```

2223

24

25

26

27

28

29

30

3132

33

3435

36

37

38

3940

}

void run() {

int radius, theta, endRot;

this->radius = r;

this->endRot = eR;

this->loopCount = 0;

 $\//\$ implement the run function

this->drop = m;

this->direction = dir;

this->theta = t;

Circle(int r, int t, int dir, int eR, bool m) {

// constructor

Page **8** of **12**

// compute the outerDist as 2*pi*(radius of circle + distance to outer wheel from center of rob

this->outerDist = (int)(2 * PI * (this->radius + WHEELDIST) * ((float)abs(this->theta) / 360));

this->innerDist = (int)(2 * PI * (this->radius - WHEELDIST) * ((float)abs(this->theta) / 360));

// similar procedure for innerDist, but subtract the wheel distance instead

```
Serial.print("Circle ");
41
42
43
             Serial.print(this->radius);
44
             Serial.print(" ");
45
             Serial.print(this->outerDist);
             Serial.print(" ");
46
             Serial.print(this->innerDist);
47
             Serial.print(" ");
48
49
50
             // drive round in a circle
51
             int driven = this->drive(this->radius, false);
52
53
             // get angular shortfall
             int angShortfall = this->theta - driven;
54
55
56
             while (abs(angShortfall) <= ANGULARTOL && this->loopCount < MAXLOOPCOUNT) {</pre>
57
                 driven = this->drive(angShortfall, true);
58
                 angShortfall = abs(angShortfall) - driven;
59
                 this->loopCount++;
             }
60
61
             this->loopCount = 0;
62
63
             // rotate to start of next leg
64
             this->rotate(endRot, false);
65
66
             // perform any actions needed
67
             this->action();
68
         }
69
 70
         int drive(int t, bool correction) {
71
             char innerWheel, outerWheel, innerSpeed, outerSpeed;
72
73
             // reset inner distance to theta
74
             this->innerDist = (int)(2 * PI * (this->radius - WHEELDIST) * ((float)abs(t) / 360));
75
 76
             // if we're going forward, the left wheel is on the inside, else its the outside wheel
 77
             if (this->direction == FORWARD) {
                 innerWheel = ENCODELEFT;
78
79
                 outerWheel = ENCODERIGHT;
80
                 innerSpeed = SPEEDLEFT;
81
                 outerSpeed = SPEEDRIGHT;
82
             } else {
83
                 innerWheel = ENCODERIGHT;
84
                 outerWheel = ENCODELEFT;
85
                 innerSpeed = SPEEDRIGHT;
86
                 outerSpeed = SPEEDLEFT;
             }
87
88
89
             // angular velocity from dual speed, with direction
90
             float omega = this->direction * ((float)DUALSPEED * 0.5 / (float)this->radius);
91
92
             // if correction, reduce the speed for greater accuracy
93
             if (correction) {
94
                 omega *= 0.2;
95
             }
96
97
             Serial.print("Omega: ");
98
             Serial.println(omega);
99
100
             // loop through driving until one of the distances is over it's limit
101
             while (individualDistance(innerWheel) <= this->innerDist) {
```

Page **9** of **12**

```
102
                 // Set wheels to spin at different rates
103
                 Wire.beginTransmission(MD25ADDR);
104
                 Wire.write(MODE);
105
                 Wire.write(MODESEPERATE);
106
                 Wire.endTransmission();
107
108
                 // set the acceleration mode to fast
109
                 Wire.beginTransmission(MD25ADDR);
110
                 Wire.write(ACCEL);
                 Wire.write(ACCELDEFAULT);
111
112
                 Wire.endTransmission();
113
114
                 // Set outer wheel speed
115
                 Wire.beginTransmission(MD25ADDR);
116
                 Wire.write(outerSpeed);
117
                 Wire.write((char)(128 + (this->radius + WHEELDIST) * omega));
118
                 Wire.endTransmission();
119
120
                 // set inner wheel speed
121
                 Wire.beginTransmission(MD25ADDR);
122
                 Wire.write(innerSpeed);
123
                 Wire.write((char)(128 + (this->radius - WHEELDIST) * omega));
124
                 Wire.endTransmission();
             }
125
126
127
             int innerDriven = individualDistance(innerWheel);
128
             // get the angle driven through
129
             int ang = 360 * innerDriven / (2 * PI * (this->radius - WHEELDIST));
130
             resetEncoders();
131
             this->stop();
132
             return ang;
133
         }
134
135
     };
136
137
     #endif
     encoderInteraction.cpp - functions for interacting with the MD25 encoders
 1
 2
         Encoder interaction file, contains functions to read and clear MD25 encoders
 3
      *
                 Uses inline functions to prevent multiple definitions
 4
      */
 5
     #include "defines.h"
 6
 8
     #include <Arduino.h>
 9
     #include <Wire.h>
 10
 11
    // include guard
    #ifndef ENCODERINTERACTION_CPP
12
13
    #define ENCODERINTERACTION_CPP
14
15
     // find distance a specific wheel has moved
     inline int individualDistance(char side) {
16
17
         // set MD25 to send the encoder for the given side
18
         Wire.beginTransmission(MD25ADDR);
19
         Wire.write(side);
20
         Wire.endTransmission();
21
```

// request 4 bytes from the MD25

```
23
        Wire.requestFrom(MD25ADDR, 4);
24
25
        // wait for first 4 bytes back
26
        while (Wire.available() < 4);</pre>
27
28
        // get all bytes of the click var
29
        long clicks = Wire.read();
        clicks <<= 8;
30
31
        clicks += Wire.read();
32
        clicks <<= 8;
33
        clicks += Wire.read();
34
        clicks <<= 8;
35
        clicks += Wire.read();
36
37
        delay(5);
38
39
        // convert clicks to mm
40
        int dist = clicks * CLICKSTOMM;
41
42
        // return absolute distance moved
43
        return abs(dist);
44
    }
45
46
   // reset distance encoders between legs
47
    inline void resetEncoders() {
48
        Wire.beginTransmission(MD25ADDR);
49
        Wire.write(CMD);
50
        Wire.write(CLEARENCODERREGISTERS);
51
        Wire.endTransmission();
52
        delay(50);
53
    }
54
55
   // find the average distance travelled
56
    inline int averageDistance() {
57
        // get individual wheel distances
58
        int distLeft = individualDistance(ENCODELEFT);
59
        int distRight = individualDistance(ENCODERIGHT);
60
61
        // find the absolute distance
62
        distLeft = abs(distLeft);
63
        distRight = abs(distRight);
64
65
        // return the average
66
        return (int)((distLeft + distRight)/ 2);
67
    }
68
69
    #endif
    defines.h - header including all definitions
 1
 2
       Defines for odometry task
 3
     */
 4
 5
    // include guard
    #ifndef DEFINES_H
 6
 7
    #define DEFINES_H
9
   // constant definitions
10 #define MAXLOOPCOUNT 5
                                     // maximum times to loop while correcting steer/drive
11 #define WHEELDIST 125
                                     // distance between centre of robot and centre of wheels
```

```
#define PIEZOFREQ 1000
                                  // frequency to sound the buzzer at
12
13 #define NOTIFYPAUSE 200
                                  // time to sound buzzer and flash light if not dropping M&M
                                  // initial angle for servo to sit at (the empty hole)
14 #define SERVOINIT 0
15 #define SERVOSTEP 34
                                  // angle to rotate servo by in order to move to next hole
16 #define SERVOPAUSE 400
                                  // time to wait to ensure M&M drops cleanly
   #define DUALSPEED 50
                                  // speed of the motors in dual mode
   #define FORWARD 1
                                  // multiplier to move forward
18
   #define BACKWARD -1
                                  // backwards multiplier
19
20
   #define LINEARTOL 2
                                  // linear tolerance for accuracy in straight line
   #define ANGULARTOL 0.5
   #define CLICKSTOMM 0.890
22
                                  // conversion factor from clicks to mm
23
24
   // MD25 I2C codes
25
   #define MD25ADDR 0x58
                                   // I2C MD25 address
26 #define SPEEDLEFT 0x00
                                  // MD25 register for speed #1 (left)
                                  // " " "
27 #define SPEEDRIGHT 0x01
                                                             #2 (right)
28 #define ENCODELEFT 0x02
                                  // encoder address left
                                         " right
29 #define ENCODERIGHT 0x06
                                  // "
30 #define ACCEL 0x0E
                                  // Acceleration encoder
31 #define MODE 0x0F
                                  // mode register
32 #define CMD 0x10
                                   // command register
33
34
   // MD25 command codes
   #define CLEARENCODERREGISTERS 0x20 // code to clear encoder values
35
   // MD25 acceleration modes
37
38
   #define ACCELDEFAULT 2
                                   // acceleration mode
39
40
   // MD25 modes
   #define MODEUNSIGNEDDUAL 2
41
42
   #define MODEDUAL 3
                                   // dual motor mode, all off speed 1
   #define MODESEPERATE 0
43
                                  // seperate motor speeds
44
45
   // pin definitions
46 #define LEDPIN 8
                                  // led pin
47
   #define PIEZOPIN 9
                                  // buzzer spin
48
   #define SERVOPIN 10
                                  // servo pin
49
50 #endif
```

Page **12** of **12**