Aero Group A1-2 Odometry Code

Odometry.ino - Arduino logic

```
/*
 *      Odometry Task - Aero 2 Group 2
 *
 *          16/11/2017
 *
 *          Code written by Duncan R Hamill - 28262174
 *          Tested & verified by Tom Griffiths - 28290771, Ali Hajizadah - 29053056, Robin Hannaford - 28331893, Felix Harris - 28611969
 *
 *          All distances in mm, all angles in degrees
 */

#include "defines.h"
#include "encoderInteraction.cpp"
#include "Course.cpp"

#include <Wire.h>

void setup() {
    // start serial for monitoring
    Serial.begin(9600);

    // setup the I2C and wait 100ms
    Wire.begin();
    delay(100);

    // zero the encoders
    resetEncoders();

    // initialise the course
    Course course = Course();

    // wait a bit before we start
    delay(1000);

    // go around the course
    course.run();

    finished();

}

// just a bit of fun
void finished() {
    tone(9,660,100);
    delay(150);
    tone(9,660,100);
    delay(300);
    tone(9,660,100);
    delay(300);
    tone(9,510,100);
    delay(100);
    tone(9,660,100);
    delay(300);
    tone(9,770,100);
    delay(550);
    tone(9,380,100);
    delay(575);
}

void loop() {

}
```

Course.cpp - Logic for completing the course

```
/*
 *  Course list - includes Legs variable that stores how to run the course
 *  Also includes servo level logic
 */

#include "defines.h"
#include "Leg.cpp"
#include "Line.cpp"
#include "Circle.cpp"

#include <Servo.h>
```

```

// include guard
#ifndef COURSE_CPP
#define COURSE_CPP

class Course {
  public:
    // pointer to the leg pointer array
    Leg** legs;

    // global servo and servo position
    Servo servo;
    int servoPosition;

    // constructor
    Course() {
        Serial.println("Constructing course");

        // set the servo to it's initial position
        ServoSetup();

        // initialise the legs pointer
        legs = new Leg*[13];

        /*
        * ---- THE LEG CODE ----
        *
        * Each leg represents a part of the course, with the following parameters
        *      Line - Distance, Direction, Angle to turn at end, drop M&M
        *    Circle - Radius, angle to move through, direction, angle to turn at end, drop M&M
        *
        *    Many of these distances and angles found empirically
        */
        legs[0] = new Line(425, BACKWARD, 0, false);
        legs[1] = new Line(455, BACKWARD,-37, false);
        legs[2] = new Circle(172, 250, FORWARD, 87, true);
        legs[3] = new Line(170, BACKWARD, -37, false);
        legs[4] = new Line(595, FORWARD, 39, true);
        legs[5] = new Line(400, FORWARD, 87, false);
        legs[6] = new Line(405, FORWARD, 89, true);
        legs[7] = new Line(400, FORWARD, 88, false);
        legs[8] = new Line(665, FORWARD, -90, true);
        legs[9] = new Circle(260, 75, FORWARD, -85, false);
        legs[10] = new Line(530, BACKWARD, 87, true);
        legs[11] = new Line(256, BACKWARD, 88, false);
        legs[12] = new Line(335, FORWARD, 143, false);

    }

    // attach the servo pin to the servo object and set it to it's initial position
    void ServoSetup() {
        servo.attach(SERVOPIN);
        servoPosition = SERVOINIT;
        servo.write(servoPosition);
    }

    // run the whole course by looping through each leg, calling it's run function, and then calling the action
    void run() {
        for (int i = 0; i < 13; i++) {
            Serial.print("Running leg ");
            Serial.println(i);

            legs[i]->run();

            // pass the leg's drop variable into the action function
            action(legs[i]->drop);
        }
    }

    // perform actions at waypoint, including dropping M&M if needed
    void action(bool drop) {
        // turn on LED and buzzer
        digitalWrite(LEDPIN, HIGH);
        tone(PIEZOPIN, PIEZOFREQ);

        // if need to drop M&M, drop one, if not delay so we can see and hear buzzer
        if (drop) {
            dispense();
        } else {
            delay(NOTIFYPAUSE);
        }

        // turn off led & buzzer
        digitalWrite(LEDPIN, LOW);
        noTone(PIEZOPIN);
    }

    // dispense an M&M
    void dispense() {
        // increase servo position
        servoPosition += SERVOSTEP;

        // make sure we don't accidentally run through all positions
        if (servoPosition >= 179) {
            servoPosition = 179;
        }

        // write the servo position and wait to ensure clean drop
        servo.write(servoPosition);
        delay(SERVOPAUSE);
    }
};

#endif
```

Leg.cpp - defines code for completing a section of the course

```
/*
 *  Leg class - controls the robot for one 'leg' (segment) of the course
 *              includes logic for the action, rotate, and stop functions
 */

#include "defines.h"
#include "encoderInteraction.cpp"

#include <Arduino.h>
#include <Servo.h>
#include <Wire.h>

// include guard
#ifndef LEG_CPP
#define LEG_CPP

class Leg
{
  public:

    // Should we drop an M&M?, leg finished successfully?
    bool drop, dir;

    // Virtual function that will be called to run this leg of the course.
    virtual void run();

    // rotate by the given angle (+ve clockwise), returning the actual angle rotated
    float rotate(float t, bool correction) {
        resetEncoders();

        if (t == 0) {
            return 0;
        }

        // find distance needed to rotate as an arc length of the required angle
        float dist = (2 * PI * WHEELDIST * ((float)fabs(t) / (float)360));

        // speeds of each wheel
        int leftWheel, rightWheel, rotateSpeed;

        // if we're in correction mode rotate slower
        if (correction) {
            rotateSpeed = DUALSPEED * 0.1;
        } else {
            rotateSpeed = DUALSPEED * 0.4;
        }

        // set speeds of each wheel depending on direction (+ve -> left goes forwards)
        if (t > 0) {
            leftWheel = (128 + rotateSpeed);
            rightWheel = (128 - rotateSpeed);
        } else {
            leftWheel = 128 - rotateSpeed;
            rightWheel = 128 + rotateSpeed;
        }

        // while we've not rotated less that the required distance
        while (averageDistance() <= dist) {

            // set wheels to spin at different speeds
            Wire.beginTransmission(MD25ADDR);
            Wire.write(MODE);
            Wire.write(MODESEPERATE);
            Wire.endTransmission();

            // Set left wheel speed
            Wire.beginTransmission(MD25ADDR);
            Wire.write(SPEEDLEFT);
            Wire.write((unsigned char)leftWheel);
            Wire.endTransmission();

            // set right wheel speed
            Wire.beginTransmission(MD25ADDR);
            Wire.write(SPEEDRIGHT);
            Wire.write((unsigned char)rightWheel);
            Wire.endTransmission();
        }

        // get the average distance we travelled
        long avg = (long)averageDistance();

        // convert that to an angle
        float ang =((float)(360 * avg)/((float)(2 * PI * WHEELDIST)));

        // negate it if we went backwards
        if (t < 0) {
            ang *= -1;
        }

        // reset encoders, stop, and delay slightly
        resetEncoders();
        this->stop();
        delay(50);
        return ang;
    }

    // stop the vehicle
    void stop() {
        // allow both registers to be set to stop
        Wire.beginTransmission(MD25ADDR);
        Wire.write(MODE);
        Wire.write(MODESEPERATE);
        Wire.endTransmission();

        // high acceleration mode
        Wire.beginTransmission(MD25ADDR);
        Wire.write(ACCEL);
        Wire.write(10);
        Wire.endTransmission();

        // set left to stop
        Wire.beginTransmission(MD25ADDR);
        Wire.write(SPEEDLEFT);
        Wire.write(128);
        Wire.endTransmission();

        // set right to stop
        Wire.beginTransmission(MD25ADDR);
        Wire.write(SPEEDRIGHT);
        Wire.write(128);
        Wire.endTransmission();
        delay(50);
    }
};

#endif
```

Line.cpp - logic for driving a straight line

```
/*
 *  Line class - drives a straight leg of the course
 */

#include "defines.h"
#include "encoderInteraction.cpp"
#include "Leg.cpp"

#include <Arduino.h>
#include <Wire.h>

// include guard
#ifndef LINE_CPP
#define LINE_CPP

class Line: public Leg {
  // count how many times we loop over the drive sections, so we don't get stuck.
  int loopCount, dir;

  // ramp function to increase speed over course of a line
  int ramp(int m, int dist, int x) {
    int offset = (2 * (float)m / dist)*(x - ((float)dist / 2));
    int spd = m - abs(offset);
    return spd;
  }

 public:
  // distance to travel, and how far to rotate to be pointing in correct direction at end of the leg
  int dist, endRot;

  // constructor
  Line(int d, int _dir, int r, bool m) {
    this->dist = d;
    this->dir = _dir;
    this->endRot = r;
    this->drop = m;
    this->loopCount = 0;
  }

  // implement the run function
  void run() {

    // empirical adjustment factor to account for backward overshoots
    if (this->dir == BACKWARD) {
      this->dist *= 0.9;
    }

    // run drive, get how far we actually drove
    int driven = this->drive(this->dir * this->dist, false);

    // calculate distance left to drive
    int shortfall = (this->dir * this->dist) - driven;

    // empirical correction for the shortfall distances
    shortfall *= 1.5;

    // aim to get within LINEARTOL of the target waypoint, without going over MAXLOOPCOUNT
    while (abs(shortfall) > LINEARTOL && loopCount < MAXLOOPCOUNT) {
      // if we aren't on target, drive the shortfall again, looping over to check we reached it
      driven = this->drive(shortfall, true);
      this->stop();
      shortfall = shortfall - driven;
      shortfall *= 1.5;
      this->loopCount++;

    }
    this->loopCount = 0;

    // now repeat this for rotation
    float rotated = this->rotate(this->endRot, false);

    float rotShortfall = this->endRot - rotated;

    while (fabs(rotShortfall) > ANGULARTOL && loopCount < MAXLOOPCOUNT) {
      rotated = this->rotate(rotShortfall, true);
      this->stop();
      rotShortfall = rotShortfall - rotated;
      this->loopCount++;
    }
    this->loopCount = 0;
  }

  // move the wheels the desired distance, and return the actual distance driven
  int drive(int d, bool correction) {

    // if given zero distance don't actually drive anything
    if (d == 0) {
      return 0;
    }

    // variable to hold average distance travelled
    int avgDist;

    // reset the encoders to get an accurate reading
    resetEncoders();
    do {
      avgDist = averageDistance();
```

```

            int spd;

            // if in a correction, go slowly for more accuracy
            if (correction) {
                spd = DUALSPEED * 0.1;
            } else {
                // set the speed to a ramp function, so we can correct for startup skew
                spd = 1 + ramp(DUALSPEED, abs(d), avgDist);
            }

            // Set both wheels to spin at the same rate
            Wire.beginTransmission(MD25ADDR);
            Wire.write(MODE);
            Wire.write(MODEUNSIGNEDDUAL);
            Wire.endTransmission();

            // set the acceleration mode to fast
            Wire.beginTransmission(MD25ADDR);
            Wire.write(ACCEL);
            Wire.write(ACCELDEFAULT);
            Wire.endTransmission();

            // set the speed
            Wire.beginTransmission(MD25ADDR);
            Wire.write(SPEEDLEFT);

            // if we're given a negative distance, drive backwards
            if (d < 0) {
                Wire.write((unsigned char)(128 - spd));
            } else {
                Wire.write((unsigned char)(128 + spd));
            }
            Wire.endTransmission();
        } while (avgDist <= abs(d));

        // return the read distance
        int avg = averageDistance();

        // negate the distance if we were going backwards
        if (d < 0) {
            avg *= -1;
        }

        resetEncoders();
        this->stop();
        delay(50);
        return avg;
    }
};

#endif
```

Circle.cpp - logic for driving an arc of the course

```
/*
 * Circle class - contains logic for driving a circular section of the course
 */

#include "defines.h"
#include "encoderInteraction.cpp"
#include "Leg.cpp"

#include <Arduino.h>
#include <Wire.h>

// include guard
#ifndef CIRCLE_CPP
#define CIRCLE_CPP

class Circle: public Leg {
    // loop counter to ensure we don't get stuck in a loop, distance the outer wheel has to rotate
    int loopCount, dir;
    float outerDist;

  public:
    // radius of the circle, angular distance to travel, final rotation for next leg
    int radius, theta, endRot;

    // constructor
    Circle(int r, int t, int _dir, int eR, bool m) {
        this->radius = r;
        this->theta = t;
        this->dir = _dir;
        this->endRot = eR;
        this->drop = m;
        this->loopCount = 0;

        // compute the outerDist as 2*pi*(radius of circle + distance to outer wheel from center of robot)*(theta/360), and parse to int
        this->outerDist = (float)(2 * PI * (this->radius + WHEELDIST) * ((float)abs(this->theta) / 360));
    }

    // implement the run function
    void run() {

        // set the robot to drive an arc in the specified direction, and at the given angle. Don't do corrective speeds
        float driven = this->drive(this->dir * this->theta, false);

        // get angular shortfall
        float angShortfall = (this->dir * this->theta) - driven;

        // call the drive function again with corrective speeds to solve any drive issues
        while (fabs(angShortfall) > ARCTOL && this->loopCount < MAXLOOPCOUNT) {
            driven = this->drive(angShortfall, true);
            this->stop();

            // subtract how far we moved from angShortfall so we get progressively closer to the target
            angShortfall = angShortfall - driven;
            this->loopCount++;
        }
        // reset the loop counter
        this->loopCount = 0;

        // rotate to start of next leg
        float rotated = this->rotate(this->endRot, false);

        // now correct rotation in a similar way to the arc drive
        float rotShortfall = this->endRot - rotated;

        while (fabs(rotShortfall) > ANGULARTOL && loopCount < MAXLOOPCOUNT) {
            Serial.println("Correcting rotation");
            rotated = this->rotate(rotShortfall, true);
            this->stop();
            rotShortfall = rotShortfall - rotated;
            this->loopCount++;
        }
        this->loopCount = 0;
    }

    // function to drive in an arc
    float drive(float t, bool correction) {
        // variables to store the encoders so we can drive clockwise and anti clockwise
        char innerWheel, outerWheel, innerSpeed, outerSpeed;

        // if we're given a zero angle don't do any driving
        if (t == 0) {
            return 0;
        }

        // set outerDistance to the arclength for the required theta
        this->outerDist = (float)(2 * PI * (this->radius + WHEELDIST) * ((float)fabs(t) / 360));

        // if we're going forward, the left wheel is on the inside, else its the outside wheel
        if (this->dir == FORWARD) {
            innerWheel = ENCODELEFT;
            outerWheel = ENCODERIGHT;
            innerSpeed = SPEEDLEFT;
            outerSpeed = SPEEDRIGHT;
        } else {
            innerWheel = ENCODERIGHT;
            outerWheel = ENCODELEFT;
            innerSpeed = SPEEDRIGHT;
            outerSpeed = SPEEDLEFT;
        }

        // angular velocity from dual speed
        float omega = ((float)DUALSPEED * 0.5 / (float)this->radius);

        // if have a negative angle, need to drive backward
        if (t < 0) {
            omega *= -1;
        }

        // if correction, reduce the speed for greater accuracy
        if (correction) {
            omega *= 0.3;
        }

        // set an unsigned char storing the velocity of each wheel
        unsigned char outerVel = 128 + (this->radius + WHEELDIST) * omega;
        unsigned char innerVel = 128 + (this->radius - WHEELDIST) * omega;

        // variable to store the distance moved by the outer wheel
        long outerDriven;

        // reset encoders so we have an accurate first reading.
        resetEncoders();

        // loop through driving until one of the outer distance is over it's limit
        do  {
            // Set wheels to spin at different rates
            Wire.beginTransmission(MD25ADDR);
            Wire.write(MODE);
            Wire.write(MODESEPERATE);
            Wire.endTransmission();

            // Set outer wheel speed
            Wire.beginTransmission(MD25ADDR);
            Wire.write(outerSpeed);
            Wire.write((unsigned char)outerVel);
            Wire.endTransmission();

            // set inner wheel speed
            Wire.beginTransmission(MD25ADDR);
            Wire.write(innerSpeed);
            Wire.write((unsigned char)innerVel);
            Wire.endTransmission();

            outerDriven = individualDistance(outerWheel);

        } while (outerDriven <= this->outerDist);

        // get the angle driven through
        float ang = (float)(360 * outerDriven)/ (float)(2 * PI * (this->radius + WHEELDIST));

        // if we were going to drive backwards negate the angle so correction doesn't go on for ever
        if (t < 0) {
            ang *= -1;
        }

        // reset the encoders, stop the robot, and return the angle traversed.
        resetEncoders();
        this->stop();
        return ang;
    }

};

#endif
```

encoderInteraction.cpp - functions for interacting with the MD25 encoders

```
/*
 *  Encoder interaction file, contains functions to read and clear MD25 encoders
 *          Uses inline functions to prevent multiple definitions
 */

#include "defines.h"

#include <Arduino.h>
#include <Wire.h>

// include guard
#ifndef ENCODERINTERACTION_CPP
#define ENCODERINTERACTION_CPP

// find distance a specific wheel has moved
inline int individualDistance(char side) {
    // set MD25 to send the encoder for the given side
    Wire.beginTransmission(MD25ADDR);
    Wire.write(side);
    Wire.endTransmission();

    // request 4 bytes from the MD25
    Wire.requestFrom(MD25ADDR, 4);

    // wait for first 4 bytes back
    while (Wire.available() < 4);

    // get all bytes of the click var
    long clicks = Wire.read();
    clicks <<= 8;
    clicks += Wire.read();
    clicks <<= 8;
    clicks += Wire.read();
    clicks <<= 8;
    clicks += Wire.read();

    delay(5);

    // convert clicks to mm
    float dist = clicks * CLICKSTOMM;

    // return absolute distance moved
    return fabs(dist);
}

// reset distance encoders between legs
inline void resetEncoders() {
    Wire.beginTransmission(MD25ADDR);
    Wire.write(CMD);
    Wire.write(CLEARENCODERREGISTERS);
    Wire.endTransmission();
}

// find the average distance travelled
inline float averageDistance() {
    // get individual wheel distances
    float distLeft = individualDistance(ENCODELEFT);
    float distRight = individualDistance(ENCODERIGHT);

    // find the absolute distance
    distLeft = fabs(distLeft);
    distRight = fabs(distRight);

    // return the average
    return ((distLeft + distRight)/ 2);
}

#endif
```

defines.h - header including all definitions

```
/*
 *  Defines for odometry task
 */

// include guard
#ifndef DEFINES_H
#define DEFINES_H

// constant definitions
#define MAXLOOPCOUNT 5          // maximum times to loop while correcting
 ↪  steer/drive
#define WHEELDIST 125           // distance between centre of robot and
 ↪  centre of wheels
#define PIEZOFREQ 1000          // frequency to sound the buzzer at
#define NOTIFYPAUSE 200         // time to sound buzzer and flash light if
 ↪  not dropping M&M
#define SERVOINIT 0             // initial angle for servo to sit at (the
 ↪  empty hole)
#define SERVOSTEP 34            // angle to rotate servo by in order to
 ↪  move to next hole
#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
#define BACKWARD -1             // backwards multiplier
#define LINEARTOL 1             // linear tolerance for accuracy in
 ↪  straight line
#define ANGULARTOL 0.75
#define ARCTOL 0.2
#define CLICKSTOMM 0.890        // conversion factor from clicks to mm

// MD25 I2C codes
#define MD25ADDR 0x58           // I2C MD25 address
```

```
#define SPEEDLEFT 0x00          // MD25 register for speed #1 (left)
#define SPEEDRIGHT 0x01         //   "      "       "      #2 (right)
#define ENCODELEFT 0x02         // encoder address left
#define ENCODERIGHT 0x06        // "        "      right
#define ACCEL 0x0E              // Acceleration encoder
#define MODE 0x0F               // mode register
#define CMD 0x10                // command register

// MD25 command codes
#define CLEARENCODERREGISTERS 0x20  // code to clear encoder values

// MD25 acceleration modes
#define ACCELDEFAULT 2          // acceleration mode

// MD25 modes
#define MODEUNSIGNEDDUAL 2
#define MODEDUAL 3              // dual motor mode, all off speed 1
#define MODESEPERATE 0          // seperate motor speeds

// pin definitions
#define LEDPIN 8                // led pin
#define PIEZOPIN 9              // buzzer pin
#define SERVOPIN 10             // servo pin

#endif
```