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
 *
 *          Code written by Duncan R Hamill - 28262174
 *          Tested & verified by Tom Griffiths - 28290771, Ali Hajizadah - 29053056, Robin Hannaford -
 *
 *          All distances in mm, all angles in degrees
 */

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

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

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

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

    // zero the encoders
    resetEncoders();

    // setup servo pointers
    Servo* servo_ptr;
    servo_ptr->attach(SERVOPIN);
    int ServoPosition = SERVOINIT;
    servo_ptr->write(ServoPosition);

    // initialise the course
    Course course = Course(servo_ptr, &ServoPosition);

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

    // TESTING - loop through legs and run their action
    //course.run();

    Circle c = Circle(150, 180, FORWARD, 0, false);
    c.run();

    // turn on for event
    //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
 */

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

// include guard
#ifndef COURSE_CPP
#define COURSE_CPP

class Course {
  public:
    Leg** legs;

    Course(Servo* servo_ptr, int* servoPosition_ptr) {
        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
        */
        legs[0] = new Line(428, FORWARD, 0, true);
        legs[1] = new Line(504, FORWARD,-37, false);
        legs[2] = new Circle(180, 270, BACKWARD, 90, true);
        legs[3] = new Line(180, FORWARD, -40, false);
        legs[4] = new Line(622, BACKWARD,50, true);
        legs[5] = new Line(400, BACKWARD,-90, false);
        legs[6] = new Line(400, FORWARD, 90, true);
        legs[7] = new Line(400, FORWARD, 90, false);
        legs[8] = new Line(660, FORWARD, 90, true);
        legs[9] = new Circle(260, 90, FORWARD, -90, false);
        legs[10] = new Line(500, FORWARD, 90, false);
        legs[11] = new Line(260, FORWARD, 90, false);
        legs[12] = new Line(340, FORWARD, 143, false);

        for (int i = 0; i < 13; i++) {
            legs[i]->servo = servo_ptr;
            legs[i]->servoPosition = servoPosition_ptr;
        }
    }

    void run() {
        for (int i = 0; i < 13; i++) {
            Serial.print("Running leg ");
            Serial.println(i);

            legs[i]->run();
        }
    }
};

#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:
    // pointer to serveo position variable
    int* servoPosition;
    Servo* servo;

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

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

    // perform actions at waypoint, including dropping M&M if needed
    void action() {
        // 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 (this->drop) { this->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);
    }

    // rotate by the given angle (+ve clockwise), returning the actual angle rotated
    float rotate(int t, bool correction) {
        Serial.print("Rotate ");

        Serial.print(t);

        // find distance needed to rotate
        int dist = (int)(2 * PI * WHEELDIST * ((float)abs(t) / (float)360));

        Serial.print(dist);
        Serial.print(" ");

        // 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 * 0.5;
        }

        Serial.print((int)rotateSpeed);

        // 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;
        }

        Serial.print(" ");
        Serial.print((int)leftWheel);
        Serial.print(" ");
        Serial.println((int)rightWheel);

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

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

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

        long avg = (long)averageDistance();

        Serial.print(avg);
        Serial.print(" ");

        float ang = (int)((float)(360 * avg)/((float)(2 * PI * WHEELDIST)));

        Serial.print(ang);
        Serial.print(" ");

        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;

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

  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->direction = dir;
        this->endRot = r;
        this->drop = m;
        this->loopCount = 0;
    }

    // implement the run function
    void run() {

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

        // calculate distance left to drive
        int shortfall = this->dist - driven;

        // aim to get within 2mm 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 = abs(shortfall) - driven;
            this->loopCount++;

        }
        this->loopCount = 0;

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

        float rotShortfall = this->endRot - rotated;

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

        // blink light, sound buzzer, and drop M&M if needed
        this->action();
    }

    // move the wheels the desired distance, and return the actual distance driven
    int drive(int d, bool correction) {
        Serial.print("Line ");

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

        while(int avgD = averageDistance() <= abs(d)) {
            int spd;

            // if in a correction, go slowly for more accuracy, else increase speed over course of a li
            if (correction) {
                spd = DUALSPEED * 0.2;
            } else {
                spd = ramp(DUALSPEED, abs(d), avgD);
            }

            // 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((char)(128 - spd));
            } else {
                Wire.write((char)(128 + spd));
            }
            Wire.endTransmission();
        }

        // return the read distance
        int avg = averageDistance();
        Serial.println(avg);
        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, direction, outerDist, innerDist;
  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->direction = 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 rob
        this->outerDist = (int)(2 * PI * (this->radius + WHEELDIST) * ((float)abs(this->theta) / 360));

        // similar procedure for innerDist, but subtract the wheel distance instead
        this->innerDist = (int)(2 * PI * (this->radius - WHEELDIST) * ((float)abs(this->theta) / 360));
    }

    // implement the run function
    void run() {
        Serial.print("Circle ");

        Serial.print(this->radius);
        Serial.print(" ");
        Serial.print(this->outerDist);
        Serial.print(" ");
        Serial.print(this->innerDist);
        Serial.print(" ");

        // drive round in a circle
        int driven = this->drive(this->radius, false);

        // get angular shortfall
        int angShortfall = this->theta - driven;

        while (abs(angShortfall) <= ANGULARTOL && this->loopCount < MAXLOOPCOUNT) {
            driven = this->drive(angShortfall, true);
            angShortfall = abs(angShortfall) - driven;
            this->loopCount++;
        }
        this->loopCount = 0;

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

        // perform any actions needed
        this->action();
    }

    int drive(int t, bool correction) {
        char innerWheel, outerWheel, innerSpeed, outerSpeed;

        // reset inner distance to theta
        this->innerDist = (int)(2 * PI * (this->radius - WHEELDIST) * ((float)abs(t) / 360));

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

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

        // if correction, reduce the speed for greater accuracy
        if (correction) {
            omega *= 0.2;
        }

        Serial.print("Omega: ");
        Serial.println(omega);

        // loop through driving until one of the distances is over it's limit
        while (individualDistance(innerWheel) <= this->innerDist) {
            // Set wheels to spin at different rates
            Wire.beginTransmission(MD25ADDR);
            Wire.write(MODE);
            Wire.write(MODESEPERATE);
            Wire.endTransmission();

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

            // Set outer wheel speed
            Wire.beginTransmission(MD25ADDR);
            Wire.write(outerSpeed);
            Wire.write((char)(128 + (this->radius + WHEELDIST) * omega));
            Wire.endTransmission();

            // set inner wheel speed
            Wire.beginTransmission(MD25ADDR);
            Wire.write(innerSpeed);
            Wire.write((char)(128 + (this->radius - WHEELDIST) * omega));
            Wire.endTransmission();
        }

        int innerDriven = individualDistance(innerWheel);
        // get the angle driven through
        int ang = 360 * innerDriven / (2 * PI * (this->radius - WHEELDIST));
        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
    int dist = clicks * CLICKSTOMM;

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

// reset distance encoders between legs
inline void resetEncoders() {
    Wire.beginTransmission(MD25ADDR);
    Wire.write(CMD);
    Wire.write(CLEARENCODERREGISTERS);
    Wire.endTransmission();
    delay(50);
}

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

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

    // return the average
    return (int)((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 2             // linear tolerance for accuracy in straight line
#define ANGULARTOL 0.5
#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 spin
#define SERVOPIN 10             // servo pin

#endif
```