COSC343 Assignment 1 - Report

Leo Gardiner, Henry Barnett, Reuben Crimp

March 31, 2014

1 The Algorithms used

1.1 General

Our overall code is very modularised, which made it easier to debug, and easier to read (arguable).

The final algorithm reads:

1. Get initial (inaccurate) light measurement.
2. Align the robot from the start square for stage 1.
3. Adjust light measurement to be more accurate.
4. Stage 1: Travel across 14 black tiles.
5. Align the robot for stage 2.
6. Stage 2: Travel across 8 black tiles.
7. Align onto the finish square.

1.2 Across the black tiles

How we traverse the black tiles, is very simple.
When on a black tile, measure the arc length from the current location to either side of the black tile, if the two arc lengths are imbalanced, then rotate the robot by some value proportional to the imbalance. Then drive straight until we hit another black tile. This repeats until 14 black tiles in stage 1, or 8 black tiles in stage 2, have been found.
We chose this algorithm because it is general enough to deal with both sections of the course. It also provides a level of reliability about where the robot will end up both in terms of orientation and position making it easier to manage the transition between the different stages.

2 Problems

2.1 No floating-point arithmetic

No floating-point data type, so instead we took advantage of the int data types large range.

```
  x *= 4.1889;
```

Became

```
  x = (x * 41889) / 10000;
```

Which is still not ideal but does give as much accuracy in the code as we can expect from the motors.

2.2 Ambient lighting

Since the values for white and black can vary depending on the ambient light and the sensor. We used the sensor in "active mode" so we measure reflectivity. We also don't try and identify shades of grey, we only check if it's black or not. Even then we don't trust it completely. We check multiple times before saying a tile is black to prevent black flecks on the grey tiles giving a false positive.

2.3 Servomechanism discrepancies

The servos were fairly inaccurate; testing our code on different robots produced highly unexpected results.
We overcame this issue by changing our algorithm completely.
Now it's much slower than our previous versions, and it's more aware of it's surroundings, making it less dependent on consistent hardware.
Our current build works on all robots that we could test it with.

2.4 Motor Syncing

We found that syncing the motors together had unintentional side effects, like jerky rotations.
To combat these side effects we reset both motors tachometer and waited a few milliseconds between commands, this prevented sequential motor commands conflicting with each other.

3 The Code

```
/*
* Assignment1.nxc - March 29, 2014
* Leo Gardiner, Henry Barnett, Reuben Crimp
*/
#define MAX(a,b) (a>b)?a:b

#define TURN_SPEED 40
#define STRAIGHT_SPEED 50
#define LEFT_WHEEL OUT_B
#define RIGHT_WHEEL OUT_C

#define BLACK_TILE 1
#define WHITE_TILE 0
#define ON_TILE(a) (SENSOR_1 < BlackThreshold) == a
#define CURRENT_TILE (ON_TILE( BLACK_TILE ) ? BLACK_TILE : WHITE_TILE)

#define LIGHT_TOLERANCE 5
#define GET_BLACK SENSOR_1 + LIGHT_TOLERANCE
#define UPDATE_BLACK_THRESHOLD MAX(BlackThreshold, GET_BLACK)

//these should be inline functions but meh....
#define GO_BACKWARDS OnRevSync(OUT_BC, STRAIGHT_SPEED, 0)
#define GO_FORWARDS OnFwdSync(OUT_BC, STRAIGHT_SPEED, 0)
#define STOP_MOTORS OffEx(OUT_BC, RESET_ALL); Wait(50)

int BlackThreshold; //global light value

//Move robot by dist (cm) in a straight line
sub move(int dist){
  RotateMotorEx(OUT_BC, STRAIGHT_SPEED, dist*20, 0, true, true);
  STOP_MOTORS;
}

//Rotate the robot about one wheel (pivot) by degrees.
sub donutTurn(int degrees, byte pivot){
  RotateMotor(pivot , TURN_SPEED, degrees*4);
  STOP_MOTORS;
}

//Rotate the robot about one wheel (pivot) by some tachometer count (arcLength)
void rotateArcDistance(byte pivot, long arcLength, int speed){
  int currArcLength = 0;
  ResetAllTachoCounts(pivot);
  OnFwdEx( pivot, speed, RESET_NONE);
  while(abs(MotorTachoCount(pivot)) <= abs(arcLength)){}
  STOP_MOTORS;
}
//returns the arc angle between current location and tile edge in "tochometer
    count"
int countArcDistance(byte pivot, int currentTile, int speed){
  long arcLength;
  ResetAllTachoCounts(pivot);
  OnFwdEx(pivot, speed, RESET_NONE);
  while( ON_TILE ( currentTile ) ){}
  arcLength = MotorTachoCount(pivot);
  STOP_MOTORS;
  return arcLength;
```

```
}

//aligns robot on the specified tile (usually black)
void alignOnTile(int currentTile ){
  long leftArcDist, rightArcDist, correctionArcDist;
  byte correctionPivot;

  //measure left arc distance
  leftArcDist = countArcDistance(LEFT_WHEEL, currentTile, TURN_SPEED);
  //move back to start
  rotateArcDistance(LEFT_WHEEL, leftArcDist, -TURN_SPEED);
  //measure right arc distance
  rightArcDist = countArcDistance(RIGHT_WHEEL, currentTile, TURN_SPEED);
  //rotate back to start
  rotateArcDistance(RIGHT_WHEEL, rightArcDist, -TURN_SPEED);

  //calculates the correction values
  if (rightArcDist > leftArcDist){
    correctionArcDist = rightArcDist - leftArcDist;
    correctionPivot = LEFT_WHEEL;
  } else {
    correctionArcDist = leftArcDist - rightArcDist;
    correctionPivot = RIGHT_WHEEL;
  }
  //corrects the robots position, "aligns" itself
  rotateArcDistance(correctionPivot, correctionArcDist/4, -TURN_SPEED);
}

//Traverses the specified number of black tiles
//Which are seperated by non-black tiles
void crossBlackTiles(int tileLimit){
  int tileCount = 0;
  while(tileCount < tileLimit){
    if (CURRENT_TILE == BLACK_TILE){
      PlayTone(500, 400);
      tileCount++;
      alignOnTile( BLACK_TILE );
      GO_FORWARDS; until( ON_TILE( WHITE_TILE ) );
    } else {
      GO_FORWARDS; until( ON_TILE ( BLACK_TILE ) );
      Wait(100); //prevents black flecks on grey tiles, trigging false positive
      STOP_MOTORS;
    }
  }
}

//move and rotate onto the final square
inline void alignForEnd(){
  move(15);
  STOP_MOTORS;
  donutTurn(90, RIGHT_WHEEL);
  STOP_MOTORS;
  move(5);
  STOP_MOTORS;
}
//move and rotate robot for stage 2
inline void alignForStage2(){
  GO_FORWARDS; until( ON_TILE( BLACK_TILE ) ); STOP_MOTORS;
  move(6);
  Wait(100);
  donutTurn(90, RIGHT_WHEEL);
  GO_BACKWARDS; until( ON_TILE( BLACK_TILE ) );
  GO_BACKWARDS; until( ON_TILE( WHITE_TILE ) ); STOP_MOTORS;
  GO_FORWARDS; until( ON_TILE( BLACK_TILE ) );
  Wait(100); STOP_MOTORS;
}
//move and rotate robot for stage 1
inline void alignForStage1(){
  GO_FORWARDS; until( ON_TILE( WHITE_TILE ) );
  GO_FORWARDS; until( ON_TILE( BLACK_TILE ) );
  PlayTone(500, 400);
  move(4);
  BlackThreshold = UPDATE_BLACK_THRESHOLD; //update light value
  donutTurn(90, RIGHT_WHEEL);
}

//main loop
task main(){
  SetSensorLight(IN_1, true);
  Wait(100);
  BlackThreshold = GET_BLACK; //get initial light value

  alignForStage1();
  crossBlackTiles(13);
  alignForStage2();
  crossBlackTiles(8);
  alignForEnd();
}
```