COSC343 Assignment 1 - Report

Reuben Crimp, Henry Barnett, Leo

March 29, 2014

1 The Algorithms used

1.1 General

Align the robot from the start square for stage 1
Stage 1: Travel across 14 black tiles
Align the robot for stage 2
Stage 2: Travel across 8 black tiles.
Align onto the finish square

1.2 Across the black tiles

How we traverse the black tiles, is pretty much the same as everyone else. When on a black tile, measure the arc length from the current location to either side of the black square, if the arc lengths are imbalanced, then rotate the robot by some value proportional to the imbalance. Then drive straight until we hit another black tile. Rinse Repeat.

2 Problems

2.1 Motor Syncing

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

2.2 No Floats

No floating point data type for algebra i.e pi = 3.14159
So we take advantage of the int data types large range i.e (x * 4.1889) becomes (x * 41889)/10000
Which is still not ideal.

2.3 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. And we don't try and identify shades of grey, we only check if it's black or not, and even then we don't trust it completely.

3 The Code

```
  /*
  * Assignment1_v8.nxc - March 29, 2014
  * Reuben, Henry, Leo
  */
#define BLACK_TILE 1
#define WHITE_TILE 0
#define ON_TILE(a) (SENSOR_1 < 45) == a //45 = boundary between black and non-black
#define CURRENT_TILE (ON_TILE( BLACK_TILE ) ? BLACK_TILE : WHITE_TILE)

#define TURN_SPEED 40
#define STRAIGHT_SPEED 50
#define LEFT_WHEEL OUT_B
#define RIGHT_WHEEL OUT_C
#define LOCKUP_WAIT 50 //ms to wait between sequential sync motor commands

//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(LOCKUP_WAIT)

//Absolute Value
long abs(long a){ return ( a > 0 ) ? a : -a; }

//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*400)/100);
    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*25)/100, -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 flecs on the grey tile triggering black
            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(5);
    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);
     donutTurn(90, RIGHT_WHEEL);
}

//main loop
task main(){
    SetSensorLight(IN_1, true);

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