GEARGIS KAN Maze Runner

Ashley Hale
Member of Team GEARGIS KAN
198 Riverside Street, Lowell, MA
Ashley_Hale@student.uml.edu

Kevin Dibble
Member of Team GEARGIS KAN
198 Riverside Street, Lowell, MA
Kevin_Dibble@student.uml.edu

Nicholas Forsyth
Member of Team GEARGIS KAN
198 Riverside Street, Lowell, MA
Nicholas_Forsyth@student.uml.edu

ABSTRACT

This paper will describe the project Maze Runner by team GEARGIS KAN. It will go over what the project was including the project's goals, implementation, and analysis of the results. This project is about robot navigation through a real world maze via cue card directions. The robot has coded behavior that dictates its reactions to the maze environment. The focus of this application is a user's ability to move the cue cards and alter the robots maze navigation.

AUTHOR KEYWORDS

Mobile robotics, behavioral robotics, interactive robotics, Braitenburg robot, UMass Lowell

INTRODUCTION

The goal of the Maze Runner project was to create a robot that would traverse a pre-determined maze by following a set of cards that were placed down at intersections and corners by a person guiding the robot.

The purpose of the robot was to experiment in developing a user-guided and behavior based robot that would allow interactive learning with minimal foreknowledge. This is accomplished by a very simplified user interaction system. The cards that direct the robot can be read from anywhere and work in any orientation, so all the user has to do is do is place cards in a path for the robot to follow. We found this method of interaction to be very natural for us as designers, but were not able to get enough testers to conclude if the card following system was intuitive over a large range of people.

PROJECT DESCRIPTION

The robot was built out of Legos and had three sensors: one camera and two top hats. The cards that are placed have one blue circle and one red circle right next to it. The robot would use the camera, attached to its underside, to read the cards and determine which direction it should turn by calculating the angle between the centers of the two circles. The top hat sensors were placed on either side of the robot, outside of the two wheels, and would correct the robot if it started to travel too close to a black "wall" of the maze.

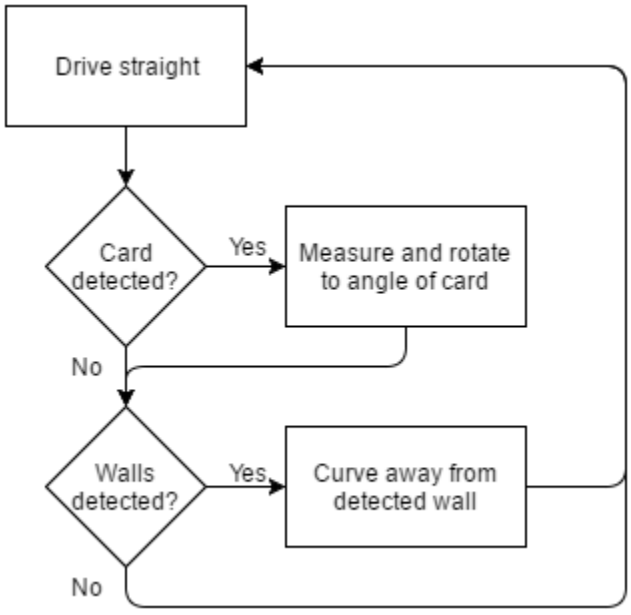

Figure 1. System Flow Chart

To make the robot as accurate as possible, we constructed our own environment, a maze, for the robot to traverse. The maze is made from white boards with thick black lines acting as the walls. The robot would work in a non-structured environment; however it would be far less accurate.

Another piece of environment structure was the addition of a self-contained light source to the robot. Due to the relative inaccuracy and low quality of the camera, steps had to be taken to normalize the card illumination levels. Adding a light source to illuminate the underside of the robot provided a stable illumination and drastically improved the accuracy of the card detection.

In order for the robot to traverse the maze we had to develop a function that would keep the robot inside the walls. Without the function the robot could easily run over the lines and miss the cards on the intended path. The function is made up of a series of if else statements which use the two top hat sensors on either side of the robot to sense when the robot is going over the line. If only one of the sensors sees black, then it will turn away from the line. If neither of the sensors sees black, then the robot will go straight. If both sensors see black then it just drives forward.

ANALYSIS OF RESULTS

We were able to create this robot and it traverses the maze successfully 4 out of 5 times. We know that the robot works because it was able to complete the maze. Improvements that could be made to increase accuracy would be to increase the field of view of the camera, include driving adjustments based on where in the frame the card was detected, or increased the number of card "commands" recognized by the robot. With improvements, the robot could have been run faster and thereby decrease the time taken to complete a course. Additionally, adding mapping functionality to the robot to allow it to retrace steps would greatly increase application opportunities.

DISCUSSION

This project taught us several algorithms including calculating the angle between two objects and tracking those objects. It also taught us the importance of debugging and testing with consistent conditions. In addition, it is extremely important to plan out clear project details and follow through with them by a specific deadline. This project also improved our ability to improvise.

This project was a learning experience for all involved. The project taught us several algorithms including calculating the angle between two objects and tracking those objects and line

following to assist maze navigation. It also taught us the importance of debugging and testing with consistent conditions. In addition, it is extremely important to plan out clear project details and follow through with them by a specific deadline. This project also improved our ability to improvise solutions.

CONCLUSION

Overall, our project is a success, but it could be improved. We could make the robot autonomous and learn from its mistakes like the robot NightFury which participated in the competition NERC in 2012. As an ambitious goal, sometime in the future the robot could work with a drone to help it traverse the maze by using the drone as its eyes. Or the drone could be used to lay down a path for the robot to follow.

The paper, *An Interactive Maze Scenario with Physical Robots and Other Smart Devices*, describes a similar project that focuses on a educational application of controlling a lego robot to navigate a maze. PDAs were used to simulate the physical environment for development of a set of rules to allow the robot to navigate the real and simulated mazes. The approach of the project allowed for students to go through an iterative process of development and testing for the robot's physical maze navigation. This relates closely with our robot's functionality and application as the people interacting with our robot, using its pre-existing rules, plot its navigation through our maze. There is a learning process for people who have not used the robot before and develop a methodology for plotting the robot's navigation. Their approach introduces a structured approach to the maze navigation learning scenario which could be used to structure our project for reproducibility.

ACKNOWLEDGMENTS

In this project Ashley Hale constructed the robot chassis, programmed the code for the robot to stay within the "walls," created the cards, and helped build the maze. Kevin Dibble was the main programmer and developed the algorithms for detecting, reading, and following the cards. Nicholas Forsyth mounted the sensors on the robot, constructed the maze, and developed tests to challenge the robot and fine tune the code.

We would like to thank Professor Fred Martin for letting us create this robot and use the robotics lab.

The work described in this paper was conducted as part of a Spring 2016 robotics course, taught in the Computer Science department of the University of Massachusetts Lowell by Professor Fred Martin.

REFERENCES

7VoltCrayon. "NightFury: Maze Traversing Robot NERC 2012." *YouTube*. YouTube. 21 Dec. 2014. Web. 3 May 2016.

M. Jansen, M. Oelinger, K. Hoeksema, U. Hoppe (2004). An Interactive Maze Scenario with Physical Robots and Other Smart Devices. In: Proc. of WMTE 2004, Los Alamitos, California (USA), pp 83-90

APPENDIX

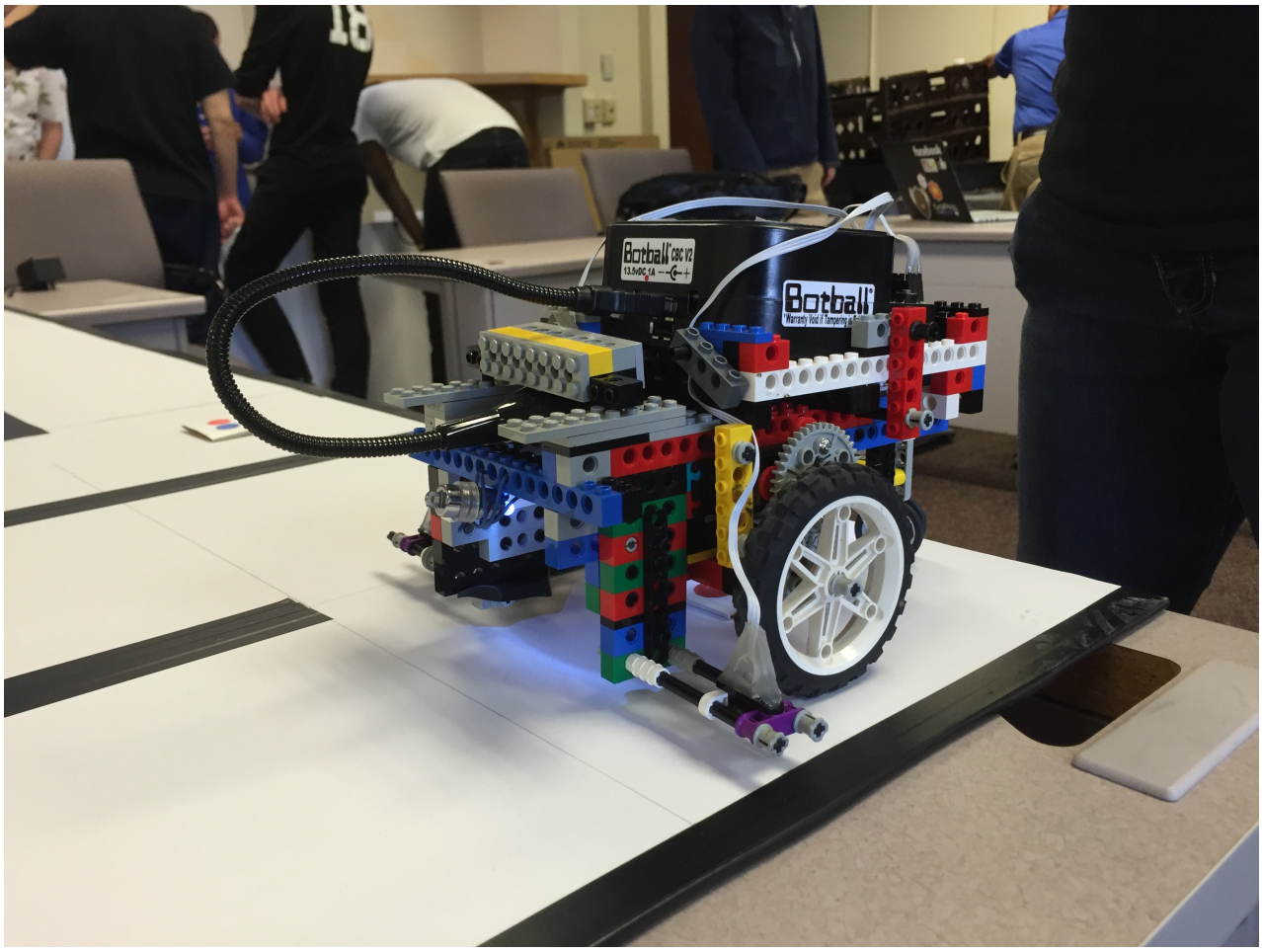

Figure 2. Maze Runner Robot

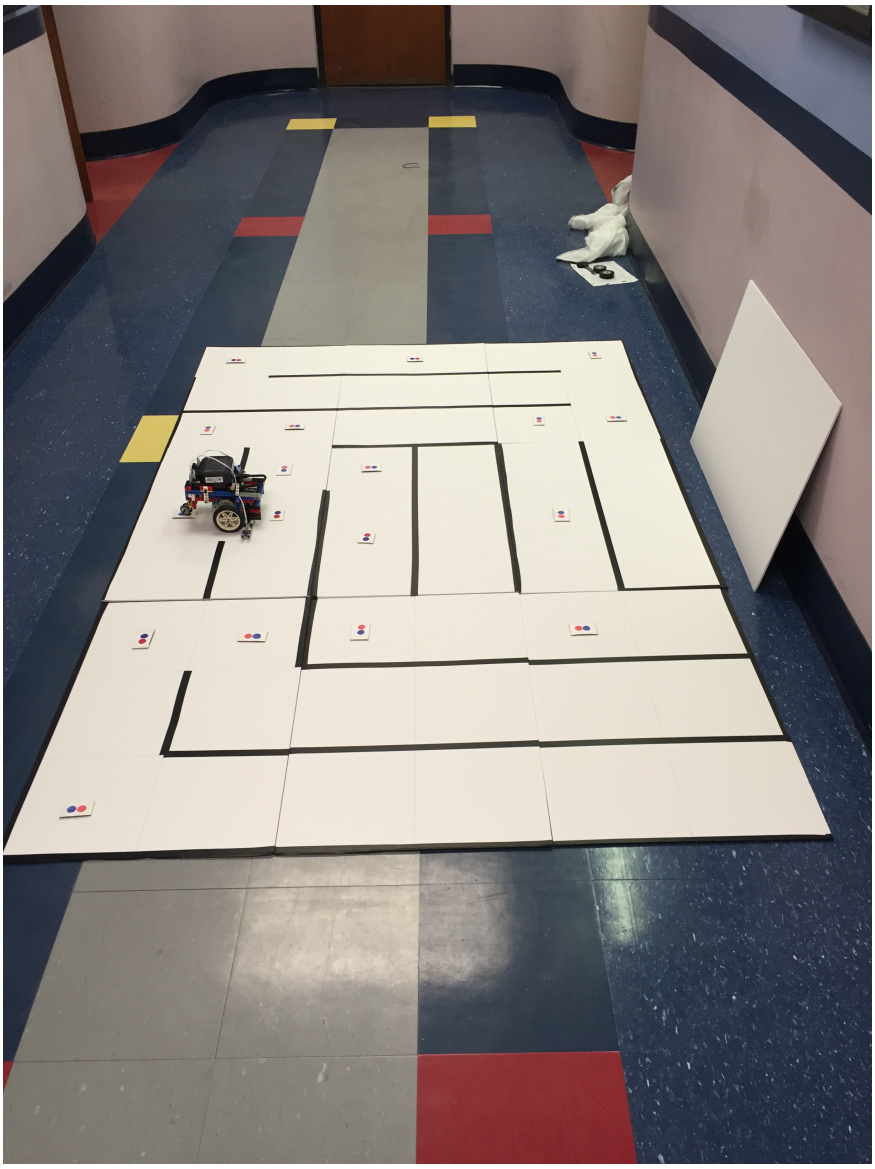

Figure 3. Maze Runner going through the maze

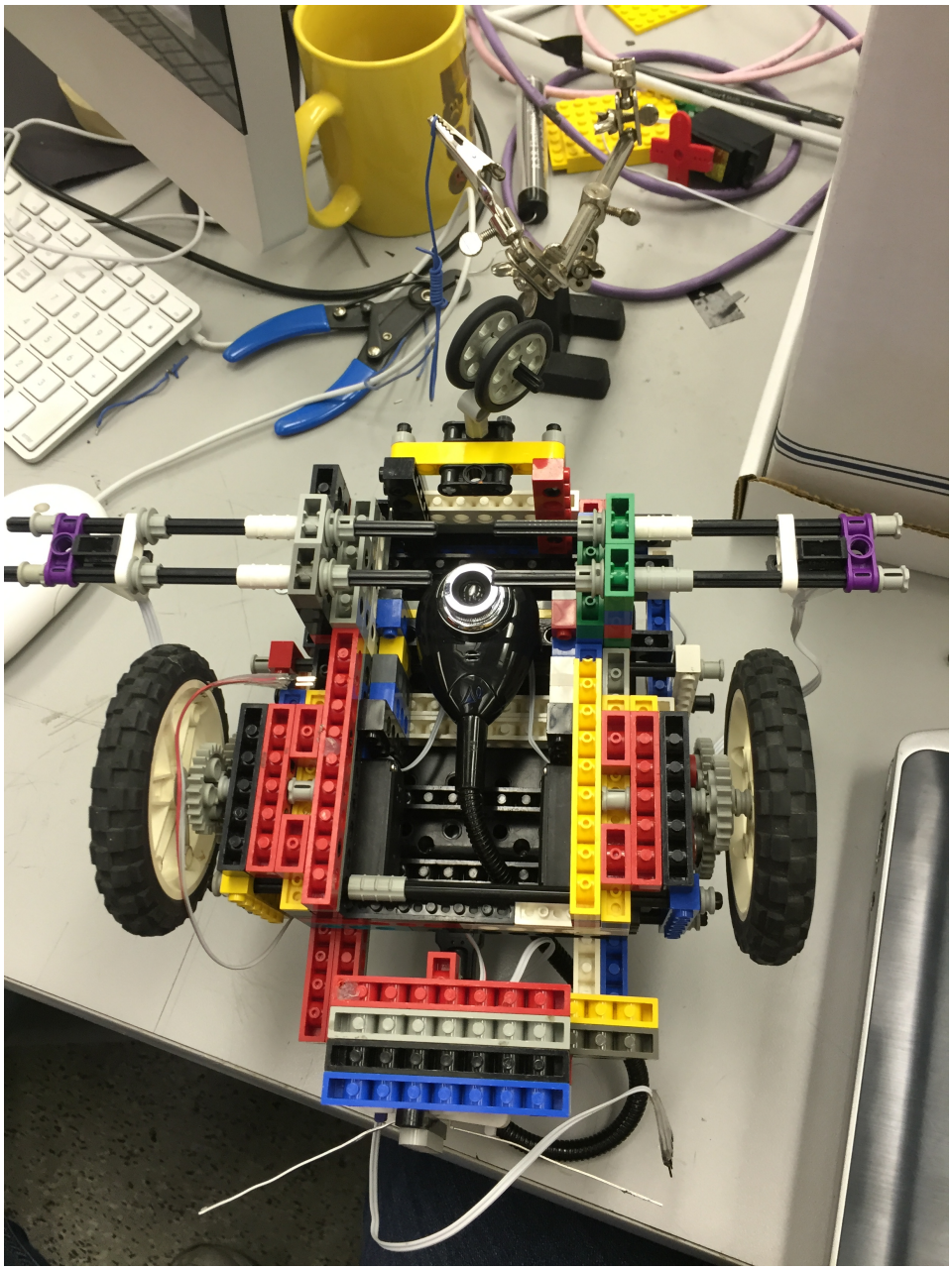

Figure 4. Underside of Maze Runner

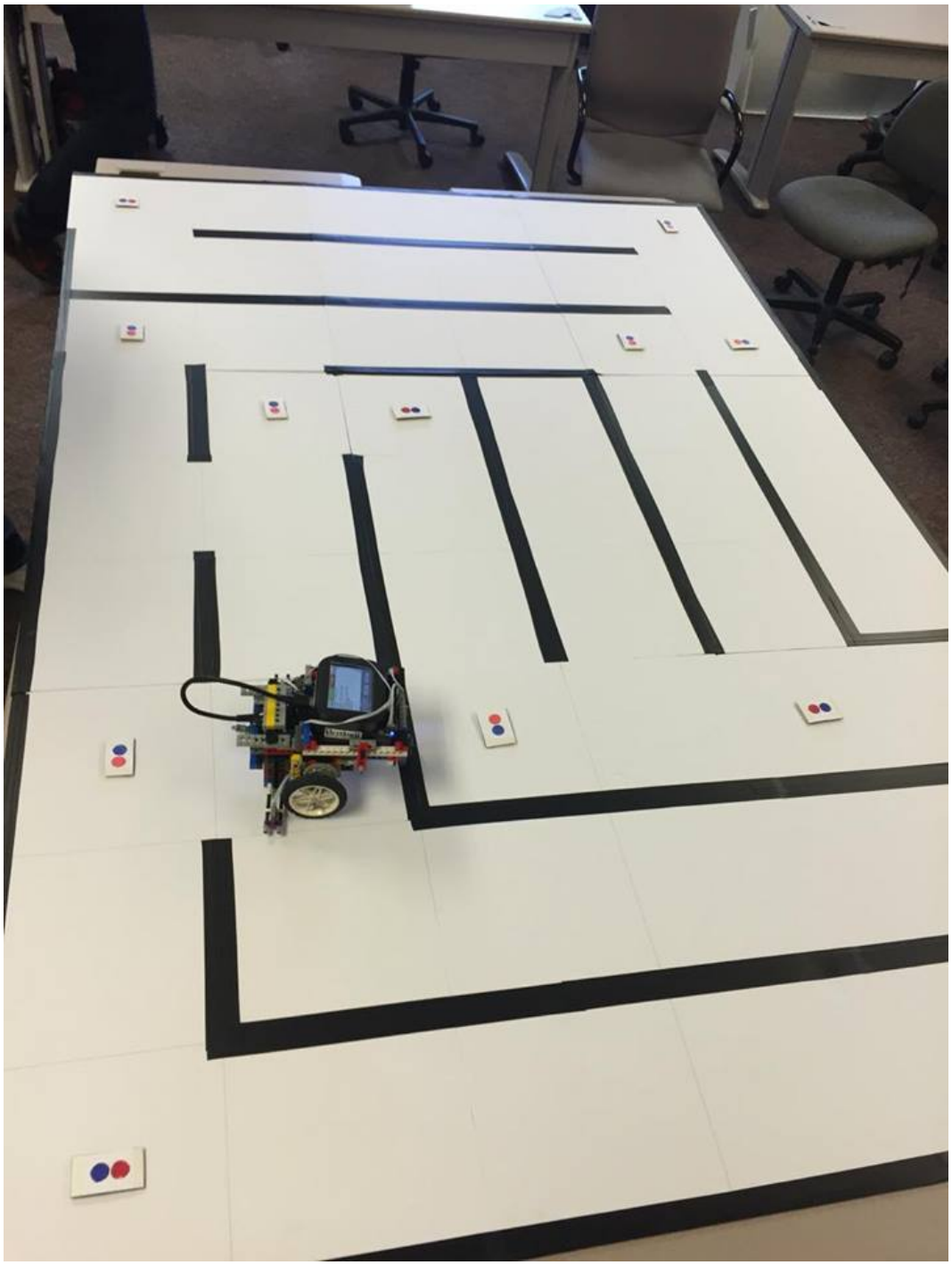

Figure 5. Maze Runner rounding a corner of the maze

Figure 6. GEARGIS KAN group members from left to right:
Ashley Hale, Kevin Dibble, Nicholas Forsyth

Video:

https://www.youtube.com/watch?v=EDUWYasbZIA&feature=youtu.be

```c
/*****************************************************************************
*********  This comment area should be filled with a description
*********        of your program and what it does.
*****************************************************************************/
/* Libraries used in Botball are automatically included, but list any additional includes below
*/
#include <stdlib.h>

/* #defines and constants go below.*/
#define RMOTOR 0
#define LMOTOR 3
#define RBUMP 8
#define LBUMP 9
#define RLIGHT 1
#define LLIGHT 0
#define ARRAY_SIZE(x) (sizeof(x)/sizeof(*x))
#define FAST 800
#define MEDIUM 600
#define SLOW 100

#define RED 0
#define BLUE 3
#define BIG_ENOUGH 473
#define PI 3.14159265358979323846
#define ANGLE_OFFSET_RAD (PI / 2)
// x = 75 y = 14 is right in front of the robot

//typedef enum {false, true} bool;
typedef enum bool {false, true} bool;
typedef enum color {black, white} color;

/* Global variables go below (if you absolutely need them).*/

/*Function prototypes below*/
bool checkBumpers(int* whichBumper);
void directionalBumpReact(int whichBumper, int time);
void setSingleSensorToFloating(int whichSensor);
void findWall();
void getSonarData(int sensorPort, int* distances);
void rotateBot(int largestDistanceIndex);
int largestValue(int* distances);
void chaseColor();
void rotate(int angle);
int findAngle(int x1, int y1, int x2, int y2);
void followLine(int lightSensorPort1, int lightSensorPort2);

int main()
{
    int x1, x2, y1, y2, w1, w2;
    int theta;

    //loop
    int x;
    while (1) {
        //go straight, staying inside the "walls" with the tophats
        followLine(LLIGHT, RLIGHT);
        track_update();
        if (track_count(RED) > 0 && track_count(BLUE) > 0) {
            w1 = track_size(BLUE, 0);
            w2 = track_size(RED, 0);
            printf("blue size: %d, red size: %d\n", w1, w2);
            // stop if we see a blue and red blob of sufficient size
            if (w1 > BIG_ENOUGH && w2 > BIG_ENOUGH) {
                ao();
                x1 = track_x(BLUE, 0);
                y1 = track_y(BLUE, 0);
```

```
                x2 = track_x(RED, 0);
                y2 = track_y(RED, 0);
                //printf("y2 %d, y1 %d, x2 %d, x1 %d\n", y2, y1, x2, x1);

                // Get the angle offset between the two blobs and invert to make positive CCW
                and negative CW
                theta = findAngle(x1, y1, x2, y2);
                printf("  FOUND CARD\n");
                printf("  theta %d\n\n", theta);
                mav(RMOTOR, 100);
                mav(LMOTOR, 100); // move forward just a bit to get the card in the center of
                rotation
                msleep(1000);
                ao();
                rotate(theta); // rotate to that angle
                mav(RMOTOR, SLOW);
                mav(LMOTOR, (int) SLOW * 1.1);
                msleep(2000);
            }
        }
        msleep(100);
    }//endloop
}

void followLine(int lightSensorPort1, int lightSensorPort2) {
    // lightSensorPort1 is left top hat
    // lightSesnorPort2 is right top hat
    int blackCutoff = 800;
    if (analog10(lightSensorPort1) > blackCutoff && analog10(lightSensorPort2) > blackCutoff) {
        // go straight
        printf("both\n");
        mav(RMOTOR, SLOW);
        mav(LMOTOR, SLOW);
    } else if (analog10(lightSensorPort1) > blackCutoff) {
        // arc right
        printf("turn right\n");
        mav(RMOTOR, 50);
        mav(LMOTOR, SLOW);
    } else if (analog10(lightSensorPort2) > blackCutoff) {
        // arc left
        printf("turn left\n");
        mav(RMOTOR, SLOW);
        mav(LMOTOR, 50);
    } else {
        printf("neither\n");
        mav(RMOTOR, SLOW);
        mav(LMOTOR, SLOW);
    }
}

int radToDegrees(double rad) {
    return (180 / PI) * rad;
}

int findAngle(int x1, int y1, int x2, int y2) {
    double O = x2 - x1, A = y2 - y1, H = sqrt(A*A + O*O);
    double oldTheta = asin(O / H), newTheta;
    //printf("oldTheta %.5f\n", oldTheta);
    //newTheta = oldTheta - ANGLE_OFFSET;
    newTheta = abs(radToDegrees(oldTheta));
    //printf("newTheta %.5f\n", newTheta);
    if(A < 0) {
        newTheta = 180 - newTheta;
        //printf("delta y was > 0\nnewTheta %.5f\n", newTheta);
    }
    if(O < 0) {
```

```
        newTheta *= -1;
        //printf("delta x was < 0\nnewTheta %.5f\n", newTheta);
    }
    /*newTheta *= newTheta;
    printf("newTheta %.5f\n", newTheta);
    newTheta = fmod(newTheta, 2.0 * PI);*/
    //printf("theta %.5f\n\n", newTheta);
    return (int) newTheta;
}

/**
 * degrees to rotate. Positive is left, negitive is right
 */
void rotate(int angle) {
    // convert angle to time.
    double conversion = 16; // generated via experimentation at speed 200 (16 worked well)
    int speed = 200;
    if (angle > 0) {
        mav(RMOTOR, speed);
        mav(LMOTOR, -1 * speed);
    } else if (angle < 0) {
        mav(RMOTOR, -1 * speed);
        mav(LMOTOR, speed);
    }
    msleep((int) conversion * abs(angle));
    ao();
}
```