DESIGN AND DEVELOPMENT OF ROBOTIC CONTROL INTERFACE FOR IROBOT CREATE USING MATLAB

Aaron R. Rababaah and Kyle Barker
University of Maryland Eastern Shore, Princess Anne, Maryland, USA
arrababaah@umes.edu, knbarker1@umes.edu

ABSTRACT

Development of intelligent behaviors for robots is highly motivated for number of reasons including: extending human capabilities, replacing human operators, safety operations, etc. in this paper we introduce the development of a set of intelligent behaviors based on different sensing technologies including: mechanical contact switches, infra-red sensing and digital imaging. In this work we will present the following developed behaviors: Traversing a predefined path, following a wall, obstacle avoidance, finding colored objects, passing through widest opening and searching and reaching a tagged object. The behaviors will be developed in the ISIDE (Intelligent Systems Integrated Environment) which is built in MatLab software. The described behaviors will tested in a lab environment and observations on accuracy and repeatability will be reported and discussed.

1.0 INTRODUCTION

When MATLAB is discussed it is often thought of as a mathematic analytical tool, but this project uses MATLAB as a high-level programming language to create behaviors for iRobot Create. The behaviors programmed in MATLAB prove its ability to be used as a high-level language, and the timeframe in which they are created can give insight into the ease of using MATLAB as a programming language.Using no other sources other than MathWorks' tutorials, MATLAB's help function, the iRobot Create's Open Interface Guide, and some guidance from Dr. Rababaah I was quickly able to feel comfortable enough to begin the first behavior. Even though this was my first time using MATLAB, it was intuitive and easy to use after only a few days worth of viewing online tutorials. The iRobot's commands and responses proved to be a bigger challenge to familiarize myself with, mainly because of the lack of researchable knowledge to use MATLAB to program the robot. Most research will turn up results of programmers using a toolbox with basic behaviors already created as functions. I had to create a function for each of the robot's commands in order to understand them. It was even a challenge to have the robot reliably connect via MATLAB on my own at first. My first practice commands originally connected to the robot itself, but this evolved into running a start function when the robot is connected.

2.0 SYSTEM CONCEPT AND DESIGN

In this section we will present and discus the different algorithms developed for Create robot namely: Traverse predefined path, wall follow, box escape and reach a target. We will be presenting the aforementioned algorithms from four different aspects: description, algorithm pseudo code, Matlab code and diagram.

2.1 Traversing a Pre-Defined Path

The traversing a path behavior collects input from the user by asking how many movements it should make in its "path", what the movements are, then executes the path. Programming the iRobot to follow a predefined path is the introductory behavior presented in this project. The movements can be made at any angle, the robot will just angle itself in the direction that you tell it (forward and reverse remains straight, but left and right will angle it in that direction. Algorithm is listed as follows:

1. Collect how many movements the robot would make from the user
2. For each movement, collect the direction, distance, and angle
3. For each movement, use a switch statement to choose a case based on the direction
4. Each case statement causes the robot to move in the specified direction, and each command to the robot uses the angle (which for forward and backward will remain 0), and distance. Each movement attribute is stored in an array other than speed, which will remain constant. 'i' is used in the first for statement in order to cycle through the attribute arrays to gather the attributes, and then in the second for loop in order to match each attribute with the movement.

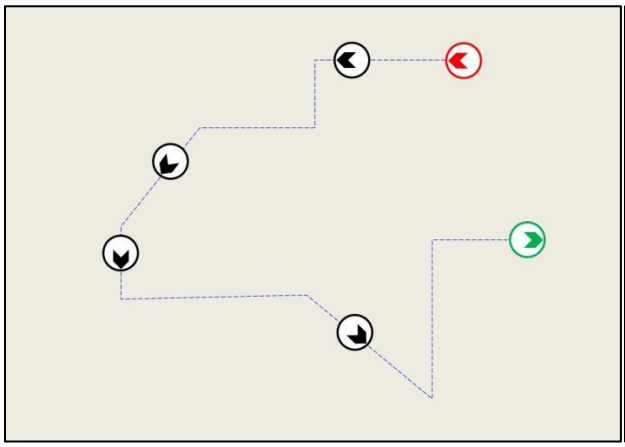

Figure 1: Traverse a Predefined Path Algorithm

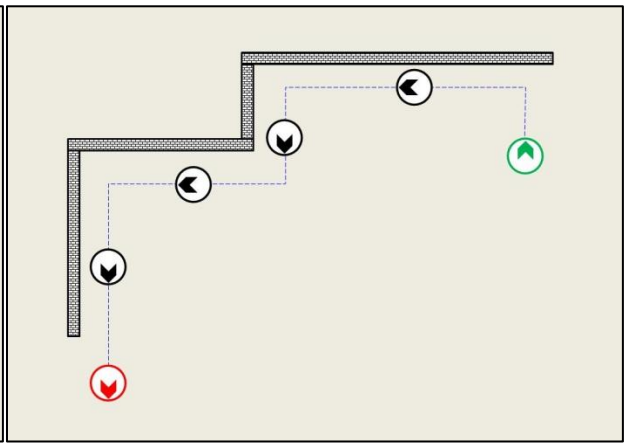

Figure 2: Follow Wall Algorithm

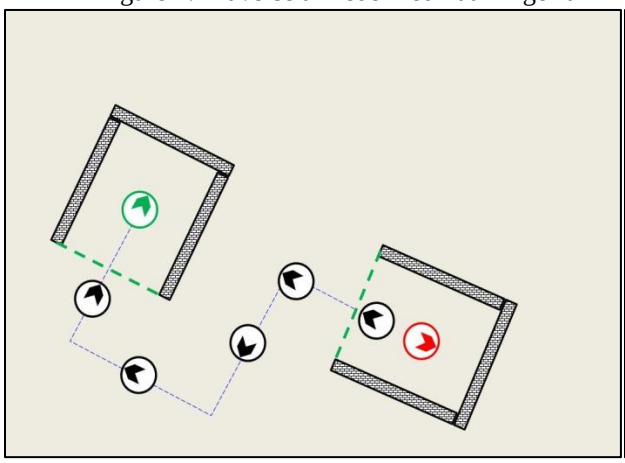

Figure 3: Box Escape Algorithm

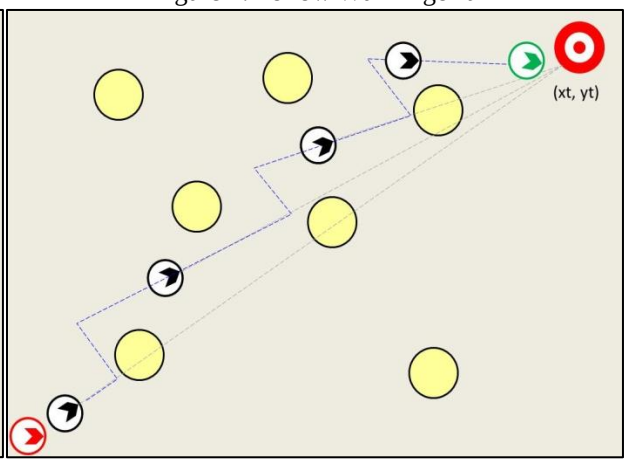

Figure 4: Reach Target Algorithm

MatLab Code

```
function create_prePath( acn, val)
global gv;
n = length(acn);
for i=1:n
    switch acn(i)
        case 'f'    % forward
create_forDist(gv.createSP, val(i), gv.createSpeed);
        case 'b'    % backward
create_backDist(gv.createSP, val(i), gv.createSpeed);
        case 'l'    % turn left CCW
create_spinCCW(gv.createSP, val(i), gv.createSpeed);
        case 'r'    % turn right CW
create_spinCW(gv.createSP, val(i), gv.createSpeed);
    end
end
```

2.2 Following Wall

Background:The following a wall behavior moves constantly along a wall using a while loop to get sensor information regarding the walls strength, and to be able to tell if the robot bumped the wall. Based on the wall's sensors strength, the robot changes its path by angling a few degrees in the required direction. If the robot bumps the wall, then it will attempt to angle away from the wall in 30 degree intervals until it has cleared the wall.

1. Move the robot forward in a straight line
2. While done = 0

3. Collect wall strength's sensor packet
4. Check if the robot has bumped a wall
5. If the robot has bumped a wall
6. Reverse 10 mm
7. Angle 30 degrees away from the wall
8. End
9. If the wall strength is greater than or equal to 50, angle the robot 2 degrees towards the wall
10. Elseif the wall strength is less than or equal to 50, angle the robot 2 degrees away from the wall
11. Else, continue the while loop
12. If the wall strength = 0 then done = 1
13. End of the while loop
14. Stop the robot

MatLab Code

```
function  create_followWall()
global gv;
done = 0;
create_forward(gv.createSP, gv.createSpeed);
while ~done
    w = create_sensors(gv.createSP, 'wall_sig', 0.5);
    w = max(w);
    b = create_sensors(gv.createSP, 'bump', 0.5);

    if any(b) % hit a wall ==> rotate 90 left and repeat algorithm
disp('bump');
create_spinCCW(gv.createSP, 90, gv.createSpeed);
create_forward(gv.createSP, gv.createSpeed);
        continue;
    end
    w
    if w <= 1
        done = 1
disp('done');
elseif w>=75
create_spinCCW(gv.createSP, 2, gv.createSpeed);
disp('w>75');
create_forward(gv.createSP, gv.createSpeed);
elseif (w<25)
create_spinCW(gv.createSP, 2, gv.createSpeed);
disp('w<25');
create_forward(gv.createSP, gv.createSpeed);
    else
        % continue ==> move with same direction if wall signal within 25-75
disp('continue');
    end
end
create_stop(gv.createSP);
```

2.3 Escaping a Box

The robot is started in a box perpendicular to any of the walls. The robot has to exit the box it is placed in and enter a different box which is placed in front of it facing a predetermined direction. The robot exits the first box by moving forward until it hits a wall, reversing back to the starting position, turning, and continues that pattern until it passes a certain distance. Once this distance has been passed then the robot knows it has exited the box and seeks the next box.

Algorithm:

1. Move continuously forward
2. While done = 0
3. Detect if a bumper was activated
4. Detect the distance travelled
5. Add the distance travelled to the total distance travelled
6. If there was a bumper hit
7. Stop the robot
8. Done = 1
9. Reverse back to the starting position
10. Turn 90 degrees left
11. Call findExit()
12. Elseif the total distance > ½ the box's length
13. Done = 1
14. Display "Exit found"
15. End
16. End
17. Flush the serial port
18. Call findExit
19. Turn left
20. Move forward the length of the box
21. Turn right
22. Move forward the length of the box
23. Turn right
24. Move forward the length of the box

Matlab Code

```
function create_escape()
global gv;
gv.createFlags(1) = 0;
for i=1:4
    if ~gv.createFlags(1)
create_forward(gv.createSP, 100);
        pause(2.5);  % 2.5 * 100 = 250mm half of the size of the assumed box
        b=[0 0];
        b = create_sensors(gv.createSP, 'bump', .1);
        if any(b)
create_stop(gv.createSP);
create_backward(gv.createSP, 100);
            pause(2.5);    % need to wait for the robot to go back to the
center of the box before attempting any more commands
create_spinCCW(gv.createSP, 90, 100);
            pause(1.5)       % need to wait for the robot to turn before
attempting any more commands
        else  % found escape door
            % in this block, the robot is just executing a must-follow
            % predefined path to the other box
gv.createFlags(1) = 1;
create_stop(gv.createSP);
```

```
create_forDist(gv.createSP, 250, 100);
create_spinCCW(gv.createSP, 90, 100);
create_forDist(gv.createSP, 500, 100);
create_spinCW(gv.createSP, 90, 100);
create_forDist(gv.createSP, 650, 100);
create_spinCW(gv.createSP, 90, 100);
create_forDist(gv.createSP, 650, 100);
        end
   end
end
```

2.4 Reaching a Target

A target is placed away from the robot with obstacles between the robot and target. The robot reaches the target by creating a grid of intervals, and the target's X and Y coordinates are given at the beginning of the behavior. The robot computes the angle and the distance away from the target by using the Pythagorean Theorem and then can find the angle by calculating the tan of the angle with the coordinates. The robot must always start at (0,0) and the coordinates cannot be negative. The robot navigates around objects by detecting a bumper hit, then reverses to its previous interval, turns 90 degrees counter clockwise, moves and interval forward, turns 90 degrees clockwise, then moves forward again. When the robot has travelled the proper distance, it will turn 90 degrees clockwise and move an interval for every bumper hit detected.

Algorithm:

1. Collect the X-Coordinate
2. Collect the Y-Coordinate
3. Calculate the distance of the object by creating a right triangle and using the distance as the hypotenuse, using the Pythagorean theorem
4. Get the distance by multiplying the hypotenuse by the unit of your intervals (must convert from mm to whatever units you are using)
5. Calculate the angle using arctan(y/x)
6. Convert the radians of the angle to degrees
7. Turn the robot to face the object at the calculated angle
8. Start the timer which calls findTimer every .5 seconds
9. Stop the robot when the timer is finished
10. If bumpCount is greater than 1
11. Turn 90 degrees right
12. For every bump
13. Move an 400 forward
14. End
15. End
16. Display "Object Found"

findTimer

1. If sensor is equal to 1
2. The distance travelled is equal to the distance travelled plus the given interval
3. Check for a bumper hit
4. End
5. If there was a bumper hit then
6. Sensor equals 0
7. Reverse back an interval
8. Turn 90 degrees left
9. Move forward 400 mm
10. Turn 90 degrees right
11. The bump counts equals the bump count plus one
12. The distance travelled equals the distance travelled minus twice the interval.
13. Continue moving towards the object

14. Sensor equal 1
15. End
16. If the distance travelled is greater than or equal to the object's distance
17. Stop the robot
18. Stop the timer
19. Set the timer to null
20. end

NOTE:
The code for this algorithm is respectively long, if the reader is interested to look at it, he/she is welcome to contact the authors. The same goes for the unreported code for Create robot basic functions.

CONCLUSIONS

MATLAB was an easy language to transition to using very little references to learn it. Mathworks was more than enough to become capable with it, and the forums will most likely already contain an answer to any errors you experience. I now recognize that MATLAB can be used outside of mathematical analysis as I thought before using it. It may not have as large of a library as other languages, but for these behaviors I never ran into a tool that I needed but did not have.The iRobot was a great way to learn how to program with. There were little to no resources online using MATLAB to program it, the Command Manual contains all the information required. Without Dr. Rababaah's help in the beginning it may have been difficult to understand some things about the robot as discussed already, but after the initial hump it was very straightforward. I will continue to update the algorithms to improve their reliability and repeatability. I plan on rewriting the Wall Follow and Test Escape algorithms to use timers instead of while loops. My goal would be to get them to run successfully without needing MATLAB and the robot to be restarted between uses, which seems to stem from the sensors being called in a while loop. I also hope to be able to continue helping student's develop behaviors for the iRobot, and to eventually incorporate an arm and webcam to add more utility via Dr. Rababaah'sISIDE.

REFERENCES

[1] David Hall and Sonya McMullen, "*Mathematical Techniques in Multi-Sensor Data Fusion*", 2004, Artech House, Inc. 685 Canton Street, Norwood, MA 02062.
[2] http://www.scientificamerican.com/article.cfm?id=a-robot-in-every-home
[3] Jennifer S. Kay, "*Robots as Recruitment Tools in Computer Science: The New Frontier or Simply Bait and Switch?*", Association for the Advancement of Artificial Intelligence (www.aaai.org), 2010.
[4] http://www.irobot.com/filelibrary/create/Create%20Manual_Final.pdf
[5] James Wolfer1 Aaron R. A. Rababaah, "Creating a Hands-On Robot Environment for Teaching Assembly Language Programming", Global Congress on Engineering and Technology Education, March, 2005, São Paulo, BRAZIL.
[6] James Wolfer1 Aaron R. A. Rababaah, "An Integrated KheperaAnd Sumo-Bot Development Environment For Assembly Language Programming", Global Congress on Engineering and Technology Education, November, 2005.
[7] SaedAmer, Amir Shirkhodaie, and Aaron Rababaah, UXO detection, characterization, and remediation using intelligent robotic systems, Proc. SPIE 6953, 69530P (2008).
[8] Aaron R. Rababaah, Emin Kuscu, Amir Shirkhodaie, Indoor Mobile Robot Localization Using IPS Cricket Technology, 2010 MTMI-NIT INTERNATIONAL CONFERENCE ON Global Issues in Business & Technology, (December 22 – December 24, 2010).
[9] http://wiki.tekkotsu.org/index.php/Main_Page
[10] www.mathworks.com
[11] Aaron R. Rababaah, "*Intelligent Systems Integrated Development Environment*", a software for machine intelligence algorithms development, University of Maryland Eastern Shore, 2011.
[12] Dr. Joel M. Esposito: http://www.usna.edu/Users/weapsys/esposito/
[13] IRobot Create Open Interface (OI) Specification. (2006). IRobot Corporation.
[14] MathWorks. (1994, January 1). Retrieved September 5, 2014, from http://www.mathworks.com/