Land-based and Drone Communication with Algorithms

New Mexico

Supercomputing Challenge

Final Report

April 2, 2014

Team 145

School of Dreams Academy and Deming High School

Team Members

Danielle Garcia

Eneyda Ramos

Seth Howe

Teacher

Creighton Edington

Project Mentor

Talyssa Ogas

Abstract

Technological advancements are occurring at an exponential pace. Technology that was the far-fetched ideas in science fiction novels are now part of our everyday lives. With cheaper and cheaper robots being developed because the enhancement of manufacturing abilities, robots have started to be ubiquitous in our modern day society. From robots that help with household chores to robots that help with search and rescue operations and military robots. This is the way into the future for better and cheaper manufacturing. In addition to cheaper manufacturing, Moore's Law has created cheaper and faster processing power for computers over the last several decades. Cheap robots plus more computing power has started moving the world into the era that will dominated by the presence of autonomous robots.

This is a proof-of-concept project, which entails the use of an AR Drone, iRobot Create, and a computer. With technology improving, there will be more aerial vehicle and land-based robot communication. Wildlife firefighters, botany surveyors, search and rescue, biologists and military operation will need more assistance in finding their target in the act of duty. There will be a maze in the project to represent the obstacles that are to be faced in the real world, such as trees and buildings. The project will go through a series of trials in order to prove this concept.

Table of Contents

List of Figures

List of Tables

1 Introduction

Each year technology is constantly improving. The military has many programs that develop our technology further and further. One of the many projects that has and is currently being worked on is unmanned autonomous vehicles. The US Navy uses Aerial Vehicles in there research. Other branches of the military use autonomous land-based vehicles to help with their research. This project is to combine both assets to create a proof-of-concept that has aerial vehicles and land-based robots to communicate together to work as a system.

1.1 Purpose

The purpose of this experiment is understanding the physical application of autonomous unmanned vehicles' communication. With this experiment involving the physical application, the overall resolution is to develop a proof-of-concept. In this proof-of-concept project, it was ideal to get an understanding of how to program and intertwine multiple programs together to work as one system.

1.2 Design Criteria

The main design for this project is to have an iRobot Create, AR Drone, and a computer to work together to go through a maze. The iRobot Create will have Proximity Sensors attached at both aides and in the front of the vehicle. The AR Drone is the first version of the commercial product line. The computer, used in this project, is a personal laptop. Using these three elements, the overall product will have a system that works together.

1.3 Materials

This project is not fully a computer simulation, but involves physical applications. The following materials in Table 1.1 are the materials used in this project.

Hardware	Software
iRobot Create	OpenCv
AR Drone	KISS-C
A laptop	Netlogo
Cardboard (boxes)	
Orange Butcher Paper (1.53 x 1.53m)	
Packaging Tape	
20.32 x 25.4 cm Construction Paper (4-green, purple, pink, yellow)	
3 Proximity Sensors	
3 (2.54 x 3.81 cm) wood blocks	
1 x 10 Strapping (2)	
1 x 19 Strapping (4)	
Create Brackets (2)	
Chassis Bracket (1)	
Create Connect Cable	
Chummby Bot Controler (2)	

Table 1.1 Materials

2 Research

Our research was more directed towards military operations, search and rescue, and research from high regard universities, such as Carnegie Mellon University and Georgia Institute of Technology. Most of the research was on naval research on unmanned vehicles. “Unmanned aerial vehicles, drones, are being used by the Navy more and more each day. The Navy is planning to…… You might deploy a remotely manned underwater generator that sits on the bottom in a secure area, which is a secure location where your forward-deployed vehicles might come back and recharge,” (Department of Defense 25).

In A Roadmap for US Robotics from Internet to Robots, the information gathered was how the application of autonomous robot communication is being applied to different fields. This report lead to search more into the application of our project. The great pacific garbage patch and biological monitoring were a few that were looked into.

Figure 2.2
Russian Olive (invasive species) in New Mexico

The system could be applied into getting counts of plastic through the aerial vehicle and uses the gaps in the plastic for the ocean-based robot to drive around to help clean (Image 2.1). When biological motoring was considered, there were many different options. The system could be used to motor plant life and weed clearing (Image 2.2), maintaining ocean life with coral reef rebuilding (Image 2.3), and keeping count of birds or other animal species and their habitats.

When it comes to traditional applications such as search and rescue, then the aerial vehicle can use thermal detection to find lost humans in avalanches or in earthquake disasters. This system can be applied to many fields. Computer science is developing and being integrated into multiple fields, and this proof-of-concept is to showcase how technology is going to be used.

Figure 2.3
Coral Reef ecosystem

3 Procedure

1. Build the Create.
    a. Mount metal brackets and strapping.
    b. Mount the Proximity sensors to the wood blocks, and mount the wood blocks to the metal strapping
2. Test colors on Netlogo
    a. Take pictures of Construction paper on different surfaces (black and grey) and import them into Netlogo.
    b. Once imported, test which colors stand out more on screen.
    c. Use the colors that stand out the most as most as the colors that will be used to detect the maze.
3. Create the maze solving program.
    a. Create a pseudo code
    b. Develop variables that will be tested within the program
4. Modify the previous program to be used with images
5. Make mock mazes with construction paper, and take image use a regular camera
    a. Import image to test on screen image
    b. Use these images to test the maze solving program
    c. Test how well this part of the experiment works until maze takes the shortest time to solve the maze.
6. Build the 1:1 maze using the Cardboard boxes with orange butcher paper, magenta and yellow construction paper.
7. Create a program on iRobot Create to navigate through the walls, using the Proximity Sensors.

    a. Testing the Proximity Sensors so that they are at the most accurate distance.
    b. Using the results from the Netlogo code, develop a program that indicates the direction the robot needs travel.
8. Create a program on OpenCV to have the AR Drone take flight and capture a still image.
    a. Send image to the laptop on which OpenCV is running on, and save in the same location as the Netlogo Code
9. The Image will be converted to a PNG file and imported into Netlogo to solve the maze.
10. The solved maze will be sent to the CBC on the iRobot Create to go through the maze on the ground.
11. Test this system, multiple times

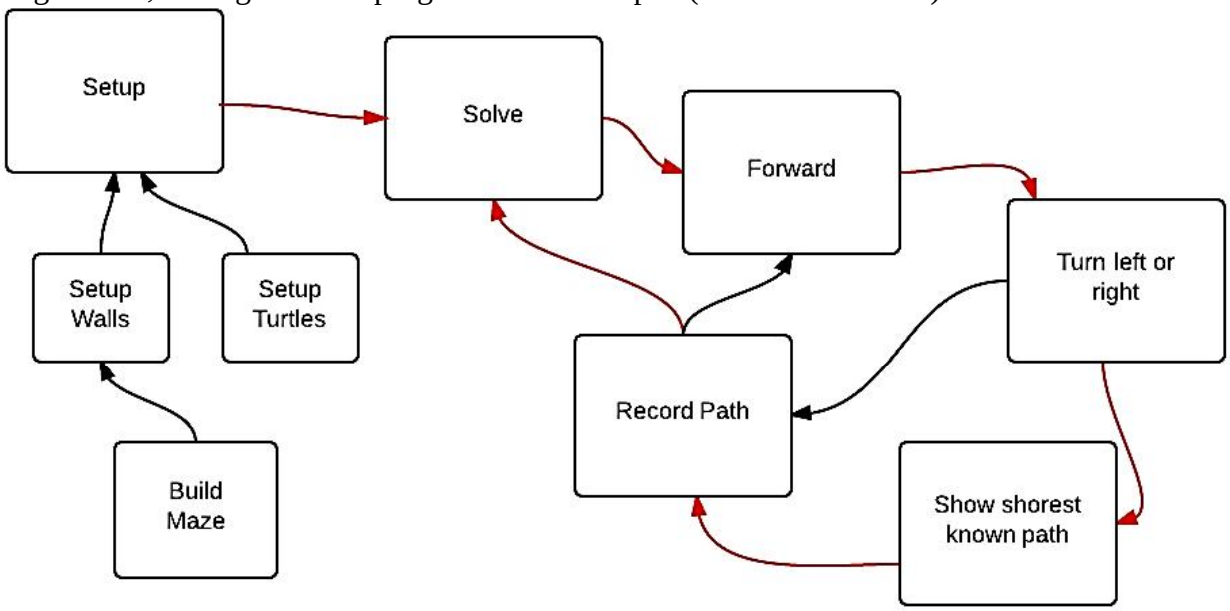
4 Code

When programming for this project there was two different languages used. Netlogo and KISS-C programming languages were used. Netlogo gave the project the platform to do agent-based modeling, and print out our results. KISS-C's platform helped with the control of the iRobot Create.

4.1 Netlogo

4.1.1 Concept Maze

The programming software used to solve the maze is Netlogo. The Netlogo code was in two stages. The first stage is the development of our maze algorithm. This code was used to help determine what type of algorithm, and to what extent was needed. The team looked into two different algorithms; one that was a trial and error and remembered dead ends or an algorithm that is a strict left-rule. With research, the team decided on the trial-and-error algorithm. With this algorithms, the logic for the program was developed (See Flowchart 4.1).



Flowchart 4.1

Logical flow of how code was developed and solved

4.1.2 Maze Solving with Picture

When implementing the image that the A.R Drone took into Netlogo, Danielle had to figure out how to import the image into Netlogo. While trying the different ways to import an image, Danielle decided on using the pcolors function. This was function was used so the image was a part of the patches. When the image was imported, the shadows in the image effected what was trying to be accomplished with the code.

The image was to be converted to being three solid recognizable pcolors within the code. Danielle had to take the time to see the range of the pcolors (Figure 4.1), and write in the code to include the shadows and bright spots (See Appendix). Once the image was fully converted, then the application of the concept maze was applied to the current program.

This provided some difficulty in incorporating the two programs. The problem was that the concept program, was not only what was needed. Danielle decided to create a program that acted like a simulation to how the create would run on without prior solving. The turtle would check at a radius of 10 patches ahead to see if a wall would be in the way. The turtle would then move forward till would have to turn, and thus repeat the process. Much like Flowchart 4.1, the logic of the program is similar. The program still give the shortest known path, as well as a set of direction for the iRobot Create to follow. The program prints the

Figure 4.1
Here is an example of how the decisions process was made in choosing the colors.

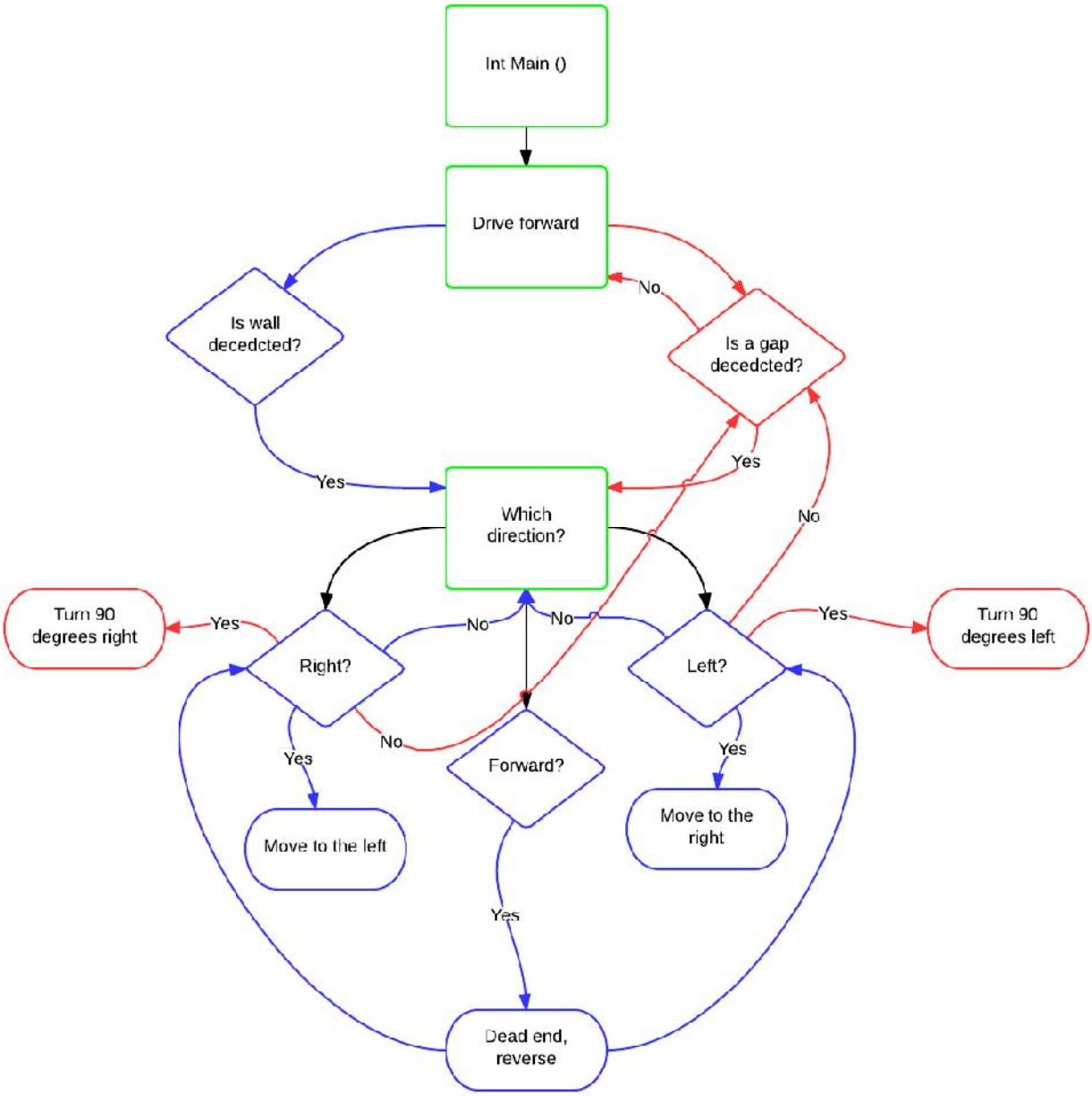
directions to turn left or right. This is then imputed into the iRobot Create Program (See Appendix).

4.2 KISS-C

The KISS-C program was created by Seth Howe. Seth developed the program to work with the Netlogo program. So the turtle from the above program, simulates how the iRobot Ceate would interact with the environment; if not for the maze solving program. The outputted directions that were printed from the Netlogo program, go into the iRobot Create program manually for now. This program and design also uses proximity sensors, as mentioned in section 5 – Design.

In the code these proximity sensors were used to detect how far away the iRobot Create is from the real life maze walls (also elaborated in section 5). These sensors are to help the Create move around the environment. The code for the Create is simple with functions in detecting the walls, gaps, and telling when the Create to turn (See Flowchart 4.2).

When first experimenting with the Create, the code found it difficult to differentiate between two gaps in the maze. It was concluded that the Create code could not do three-way intersection, so Danielle compensated in the Netlogo program. This problem was solved, by having the Netlogo code print out the directions for Seth to imput into the Create. This solution is working currently.



Flowchart 4.2

This flowchart shows the logic within the KISS-C code. The red lines represent the variable of detecting the gaps within the maze and the logic it follows. The blue line are to follow the functionality of how the Create moves around the wall. The black lines represent similar functions.

5 Design

This project is not just a computational model, but has physical application. There are two main designs; the computational design described in Section 4 and the physical design. This section will focus on the physical design of the project. This includes the design process on the iRobot Create and the development of the three-dimensional maze.

5.1 iRobot Create and A.R. Drone version 1

The iRobot Create was provided through KIPR (KISS Institute of Practical Robotics). The iRobot Create as shown is in Figure 5.1 is displayed with three proximity sensors attached with a wooden block to maintain stability. The reasoning for only using three sensors instead of four was because, there was no useful function for a sensor in the rear. If there were to be a proximity sensor in the rear, then the values would read a higher value—as if there was no wall—and cause confusion to the Create.

Figure 5.1

This is an image is how the Create is set up and how the proximity sensors are mounted.

The A.R. Drone version 1, as seen in Figure 5.2 did not go through any physical changes. The A.R. Drone v1 is the same

Figure 5.2

This is the A.R Drone model that was used for the project.

commercial product that can be found in stores or online. This A. R. Drone was provided by KIPR, but had no hardware or software modifications done.

5.2 Maze development

The maze that is used currently uses orange butcher paper with magenta and yellow construction paper to indicate the starting and end points. The reason for the colors was to create a two-dimensional image in Netlogo. The overview of the colors acts an indicator for where the walls are in the real world. In choosing the colors that would work have the best color quality in Netlogo.

Eneyda Ramos would collect sheets of colored construction and take pictures on different backgrounds, like the asphalt on a parking lot and concrete (grey) background. While Danielle and Seth would take pictures of colored construction paper on local school floor ground, like Figure 5.3. The best colors that worked was purple (magenta), orange,

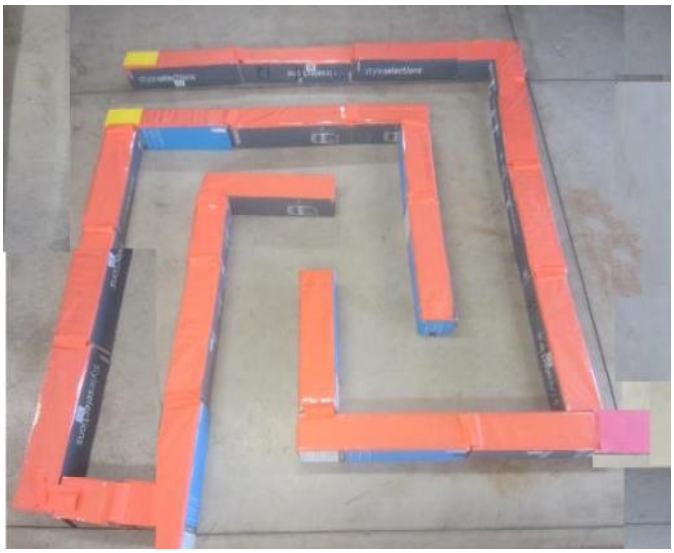


Figure 5.3

This is the A.R Drone image that was taken. Orange, purple, and yellow were the final colors choosen.

yellow, blue, and green. These colors worked great separately. When the initial testing begin, the purple construction paper worked the best once imported into Netlogo. This was the color that was going to be used with orange and green starting and end points. This proposed a problem once imported into Netlogo. The green and purple construction paper had similar shadows; therefor, the green was mistaken for the purple.

Orange was the next color that was able to be spotted easily. When testing with the orange construction paper, yellow and purple construction paper worked best. This is the final colors that were decided in using (See Figure 5.4).

5.3 Proximity Sensors

From a previous project worked on last year by former student Keva Howe and current students Denton Shaver and Danielle Garcia called Sensor Data Refinement, had research that applied to this project. The final report of last year's team 180 was included in the research and review process (See Appendix).

When building the Create and mounting the sensor about 7cm away from the initial target placement. The distance gets rid of the false peak in the raw data. Each wall had to be a total distance of 53.34cm (21 inches) apart from each other for the data recorded.

6 Processes

This section will follow the process of how all the codes interacted with one another to the physical application. The process is specifically directed towards how the code runs in the agent-based model and the physical model. All sections will be explained with a series of images and a brief explanation.

6.1 Netlogo Process

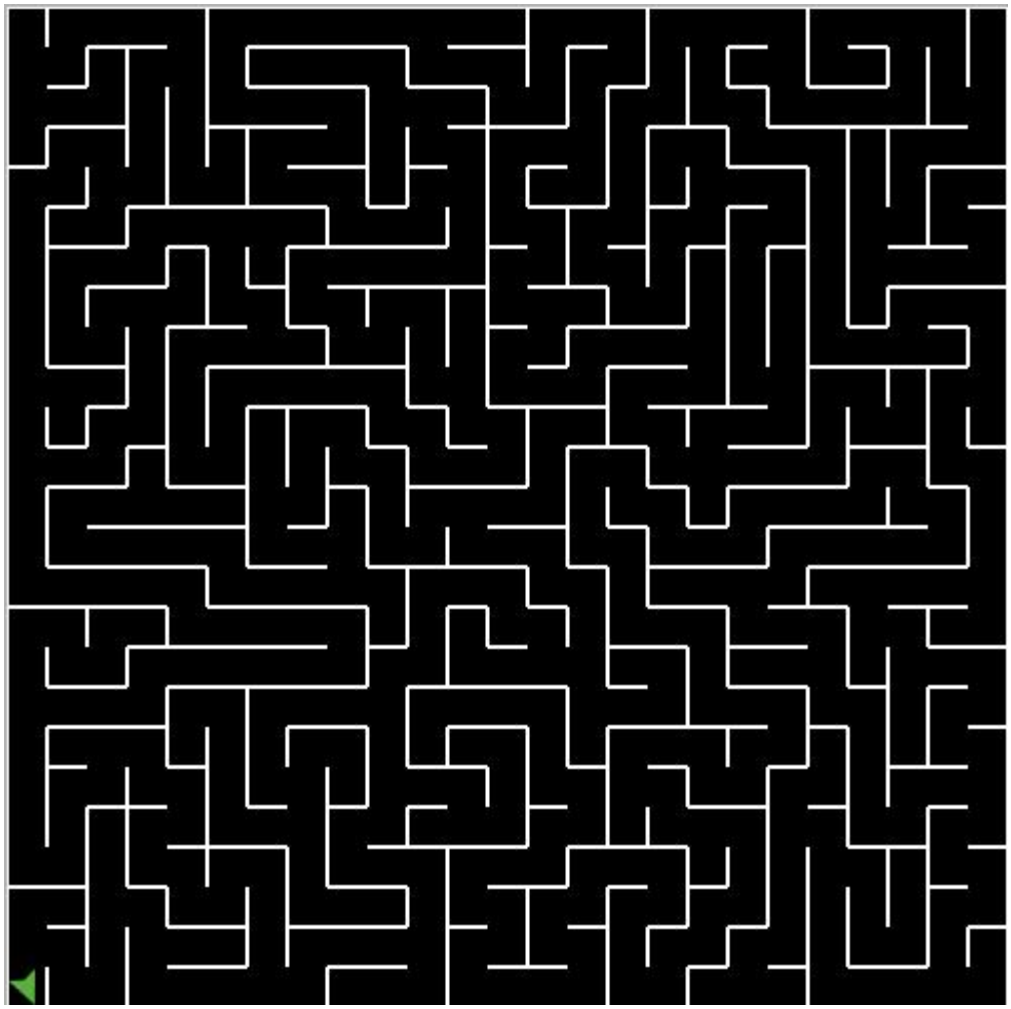


Figure 6.1

This is co the concept maze before any solving has been complete. The walls that are built within the program turn into the image above. A turtle in the lower left corner is the start of the maze.

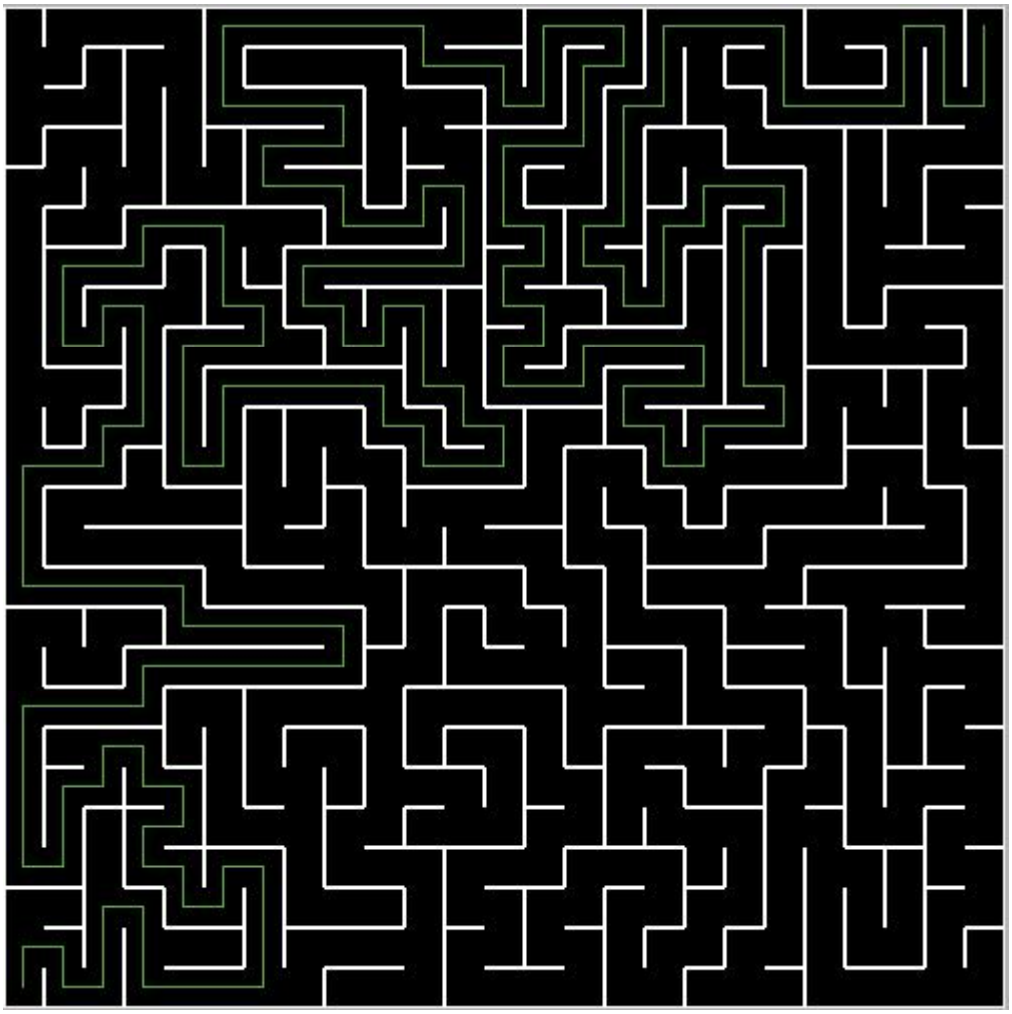


Figure 6.2

This is the solved concept maze without the dead ended paths. Every time the program ends, it saves the most current solved maze. This shortens the time it takes to solve the maze each time.

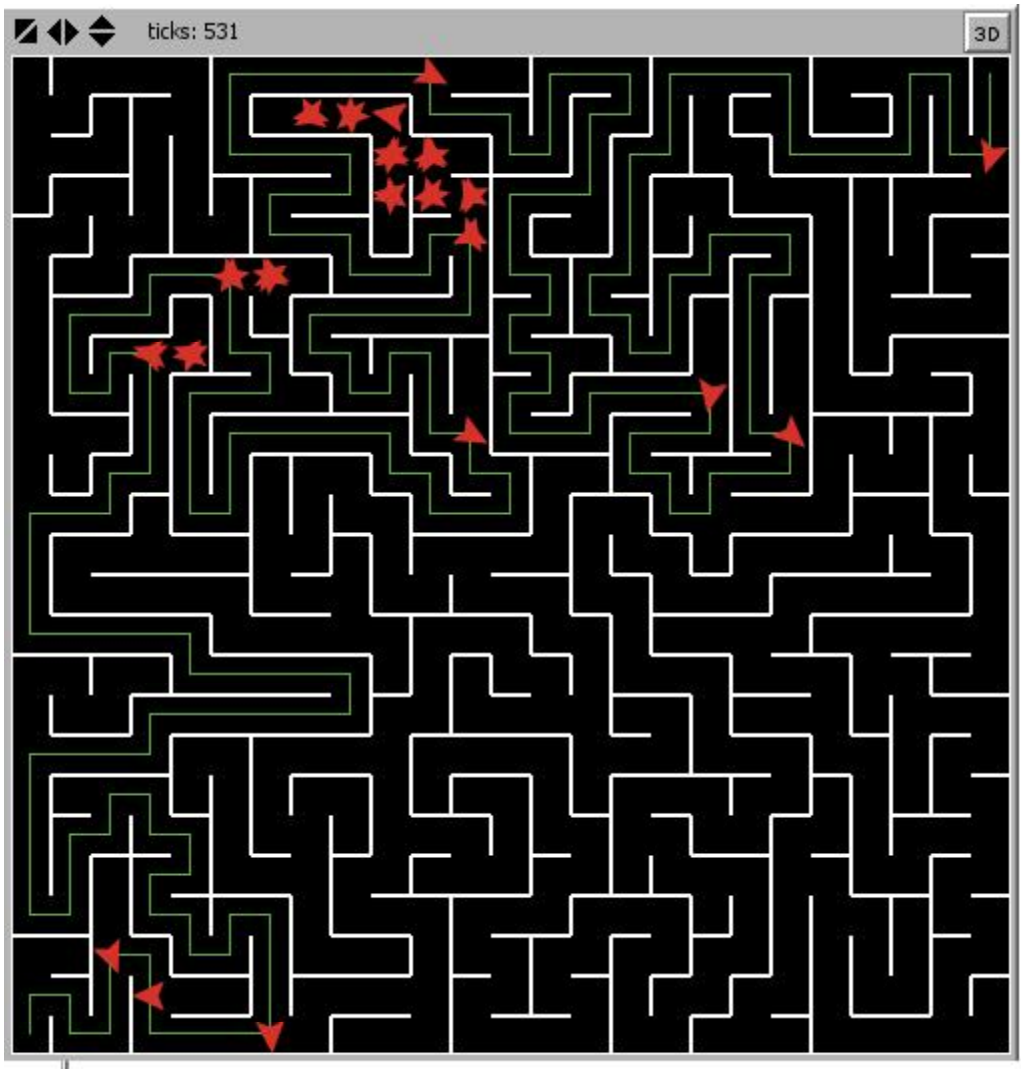


Figure 6.3

Above is the final image in the process of concept maze. This image shows the dead ends that did not work.

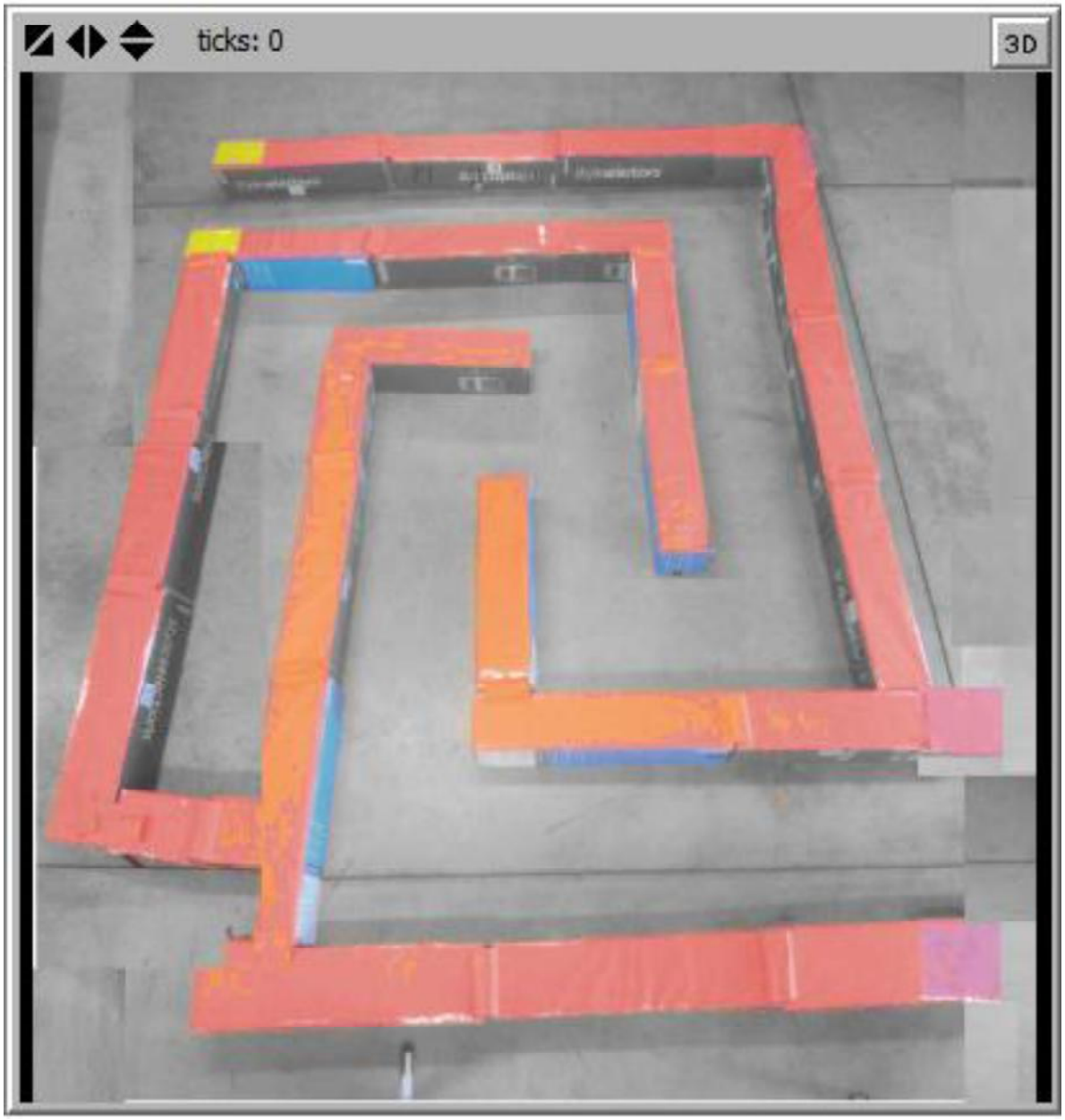


Figure 6.4

Here is the aerial view of the real life maze. The image is distorted, as anticipated. This is how the image looks before it is converted to pcolors.

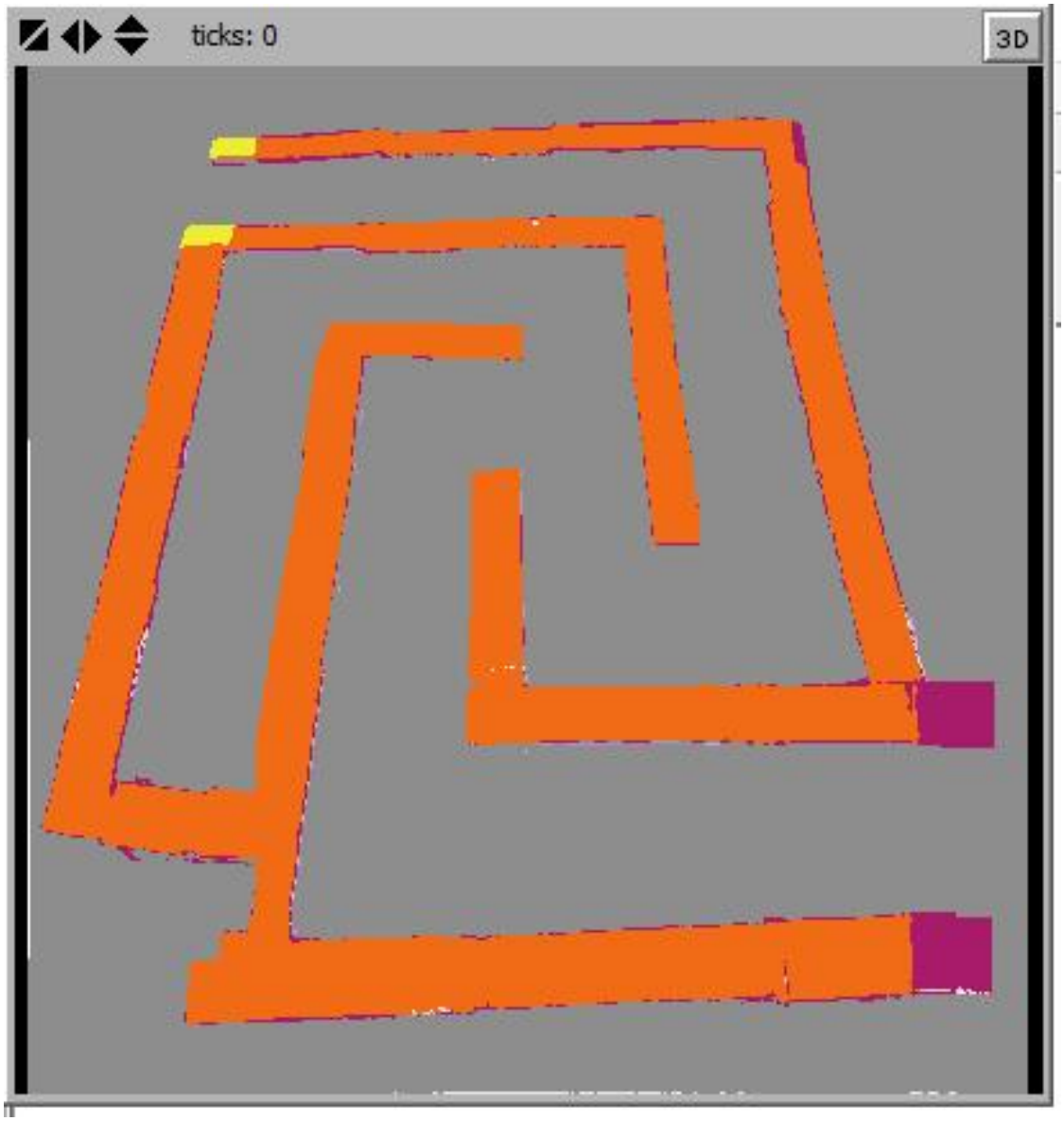


Figure 6.5

The above image is the same image from Figure 6.5 only converted into the pcolors.

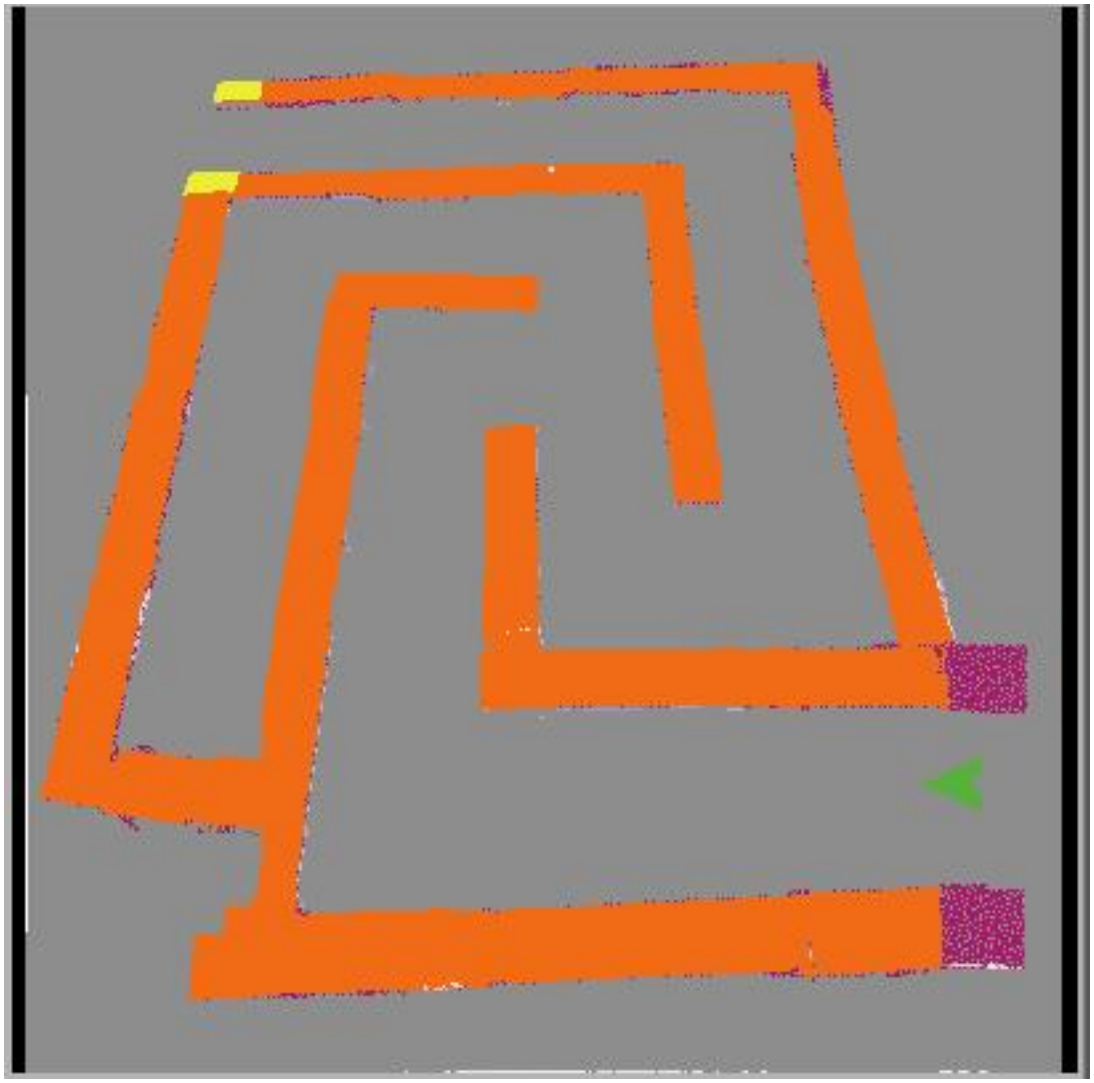


Figure 6.6

Here the start of the program solving the real-life maze. There are many patches that the turtle will turn many times while moving forward.

7 Results

The results that concluded this experiment was successful in proving this as a proof-of-concept. The image that was taken and imported into the Netlogo program acted as a simulation to what the iRobot Create would solve. The maze (Image 1) was used to work on the algorithm before Figure 2 was imported. The image is distorted, which was to be expected. Due to this distortion, the Netlogo program acts as simulation as said before. When the program finishes, it prints out the set of coordinates and which direction the iRobot Create will need to turn. From the directions that the Netlogo program provides, the iRobot Create followed the maze. Due to the Proximity Sensors, the iRobot Create was only able to detect left or right. When met at an intersection, the program would turn the iRobot Create into circles. This was fixed in the Netlogo program. The Netlog program gives direction to the iRobot Create to help reduce the confusion.

8 Data Analysis

The data for this project is a comparison between how the iRobot Create would interact with the environment without the Netlogo program (Test A) and with the Netlogo program (Test B). The results that came from Test A were that the Create took a long time to solve the maze. The Create hit dead-ends and would have to turn around. This method took a lot of time, and it was determined that the time should be cut.

When running the Create with the maze previously solved, the time was shortened by a few minutes. From this, the team predicted that with larger mazes that the time gap will be much larger.

9 Conclusions

The system for this project works effectively. The Netlogo program does solve our maze, and works well with the iRobot Create. There are still a few bugs in the program, but will soon be fixed. The AR Drone image had to be simulated. As the team is learning to use Python, Javascript, and C++ to help with the connection between all three parts of this project. The research was well proven in our experiment.

As far as our research, this helped prove our effort into other areas of science. This concept can be used in numerous situations. The maze is an analogy for an obstacle. The Netlogo program is used to solve this at its most basic level with only a single-solution maze. The main design criteria was to make a more efficient system for unmanned autonomous aerial and land-based robot communication. This design is halfway there. Once the AR Drone gets fully working to communicate with the laptop, then the project will be at its ideal state. For now, we have a proof-of-concept idea that works successfully.

10 Future Plans

This project has room for improvement. The future direction of this project to use OpenCV for all the functionality. It will be used to connect the AR Drone to the computer (the location of where the maze will be solved). The information from OpenCV will be included into the iRobot Create going through the maze. This updated program will have multi-solutions looking for the best exit, instead of a single-solution maze.

Appendix

Acknowledgements

The team would like to thank a number of people. First, and foremost we would like to thank our teacher, Mr. Creighton Edington, for his support and help on this project. Talyssa Ogas, thank you for your support and input on this project. Everyone that has shared their support, input, mentorship, and help on this project. To the wonderful people that emailed us to the judges' inputs at New Mexico Institute of Mining and Technology, we appreciate all your help and suggestions. All of the help we received went to make this project the best it could be. Lastly, the team would like their parents; who have been the support backbone in this project.

Code

Netlogo- concept maze

```
__includes ["Simple Maze Setup.nls"]

patches-own [ visited? entered-from solution?]
turtles-own [ path steps]

globals [ ending-patch who-num mylist ok-to-delete?]


;; The observer sets up the world by clearing all variables and agents, and
;; setting up the maze.

to setup
  clear-all
  reset-ticks
  set ok-to-delete? 0
  let current-random (random 2147483648) * one-of [ -1 1 ] ;; generate random
number to be used after maze is setup
  setup-maze ;;
  random-seed current-random
  ask patches
  [
```

```
    set visited? false
    set entered-from nobody
  ]
  set ending-patch ( patch max-pxcor max-pycor )

  create-turtles 1
  [
   setxy min-pxcor min-pycor
   set color green
   pen-down
   set visited? true
   set steps 0

   set who-num who ;; needed so maze-solving turtle can be directed to open .txt file
when maze is solved


   if file-exists? "current-path.txt" ;; delete old current-path file
    [
      file-close
      file-delete "current-path.txt"
    ]

    if file-exists? "current-num-of-steps.txt" ;; delete old current-path file
    [
      file-close
      file-delete "current-num-of-steps.txt"
    ]

    ifelse file-exists? "min-num-of-steps.txt" ;;
    [
      ;; do nothing - "else" is used to set high value of min-num-steps on first run
    ]
    [
      file-open "min-num-of-steps.txt"  ;;
      file-write 1000000000
      file-close
    ]
  ]
end
```

```
to solve
  record-path
  if count turtles = 0
  [
    stop
  ]
  ask turtles
  [
  if patch-here = ending-patch
  [
    record-current-num--of-steps
    record-info
    die
  ]



  let origin patch-here
  let possible-destinations (neighbors4 with [ not visited? and is-accessible? origin ])
  ifelse any? possible-destinations
   [
    face ( one-of possible-destinations )
    forward 1
    set visited? true
    set entered-from origin
   ]
   [
     set color black
     face entered-from
     forward 1
     set color green
     ask patch-here
     [
     set solution? true
     ]
   ]
  ]
  tick
  ask turtles
```

```
  [
    set steps steps + 1
  ]
  solve
end

to record-path
  file-open  "current-path.txt" ;; Opening file for writing
  ask turtles
    [ file-write xcor file-write ycor ]
  file-close
end

to record-current-num--of-steps
  file-open "current-num-of-steps.txt"  ;; Opening file for writing
    ask turtle who-num
    [  file-write steps ]
  file-close
end


to record-info
   file-open "current-num-of-steps.txt"  ;; Opening file for writing
   let current-num-of-steps file-read
   file-close

   file-open "min-num-of-steps.txt"  ;; Opening file for writing
   let min-num-of-steps file-read
   file-close


   if current-num-of-steps < min-num-of-steps
    [
      file-delete "min-num-of-steps.txt"
      file-open "min-num-of-steps.txt"  ;; Opening file for writing
      file-write current-num-of-steps
      file-close


      if file-exists? "shortest-known-path.txt" ;; delete old current-path file
      [
```

```
      file-delete "shortest-known-path.txt"
    ]

    ;;; start current to shortest

    file-open "current-path.txt"  ;; Opening file for writing
    while [not file-at-end?]
    [
      let temp-value file-read
      file-open "shortest-known-path.txt"  ;; Opening file for writing
      file-write temp-value
      file-close
      file-open "current-path.txt"
    ]
    file-close




  ]
end
to show-shortest-known-path
  clear-turtles
  file-open "min-num-of-steps.txt"  ;; Opening file for writing
  let min-num-of-turtles file-read
  file-close

  file-open "shortest-known-path.txt"  ;; Opening file for reading

  create-turtles min-num-of-turtles
  [
    set color red
    setxy file-read file-read
    if other turtles-here = true
     [
       die
     ]
  ]
    file-close
```

```
    ask turtles
    [
       ask patch-here
       [
         if solution? != true
         [
           ask turtles-here
           [
             die
           ]
         ]
       ]
    ]

end
;; need to work on this more
to show-shortest-path
  show-shortest-known-path
  ask turtles
     [
       while [ any? other turtles-here ]
        [
          ask other turtles-here
            [
              die
            ]
          die
        ]
     ]
end
```

Netlogo-Picture maze

```
turtles-own [ max-range min-range ]

globals [ scan-distance ]

to setup
  clear-all
  setup-terrain
  set scan-distance 0
  reset-ticks
end

to setup-terrain
 import-pcolors "A-Maze.png"
  ask patches [
    if pcolor = shade-of? pcolor orange
    [
      set pcolor orange
    ]

     if pcolor >= 16 and pcolor <= 29
    [
      set pcolor orange
    ]

    if pcolor = shade-of? pcolor magenta
    [
      set pcolor magenta
    ]
    if pcolor >= 127 and pcolor <= 137
    [
      set pcolor magenta
    ]
    if pcolor = shade-of? pcolor gray
    [
      set pcolor gray
    ]
    if pcolor >= 2 and pcolor <= 9
    [
      set pcolor gray
```

```
  ]

  if pcolor = shade-of? pcolor blue
  [
    set pcolor gray
  ]
  if pcolor >= 94 and pcolor <= 115
  [
    set pcolor gray
  ]
  if pcolor = shade-of? pcolor yellow
  [
    set pcolor yellow
  ]
  if pcolor >= 43 and pcolor <= 46
  [
    set pcolor yellow
  ]
  if pcolor = shade-of? pcolor brown
  [
    set pcolor gray
  ]
  if pcolor >= 35 and pcolor <= 39
  [
    set pcolor gray
  ]
]

  ask patches
  [
    if pcolor = magenta
    [
      sprout 1
      if count turtles-on neighbors < 2
      [
        set pcolor gray
      ]
    ]
  ]

  ask turtles
```

```
    [
      die
    ]

    create-turtles 1
    [
      set xcor 156
      set ycor -92
      set size 30
      set color green
      set heading 270
      set max-range 16
      set min-range 20
    ]
end

to move
  side-adjust
  ask turtles
  [
    let splotch patches in-cone scan-distance 90
    ifelse any? splotch with [pcolor = orange]
     [
       set color red
       set heading one-of [ 0 90 180 270 ]
     ]

     [
       set color green
       forward 1
     ]
  ]
end

to side-adjust ;; move to center of alley
  scan

end


to scan
```

```
let center-target 0
let delta-splotch 0
let right-x-splotch 0
let right-y-splotch 0
let left-x-splotch 0
let left-y-splotch 0

let splotch-detected 0
let splotch-360 0

ask turtles
[
  if heading = 90 or heading = 270
  [
    right 90
    let North-South-splotch patches in-cone scan-distance 90

    while [ (count North-South-splotch with [pcolor != gray ] ) < 2 ]
    [
      set scan-distance scan-distance + 1
      set North-South-splotch patches in-cone scan-distance 90
    ]

    set delta-splotch scan-distance
    set scan-distance 1


    left 180 ;; rotate to look at other side
    set North-South-splotch patches in-cone scan-distance 90

    while [ (count North-South-splotch with [pcolor != gray ] ) < 2 ]
    [
      set scan-distance scan-distance + 1
      set North-South-splotch patches in-cone scan-distance 90
    ]
    set delta-splotch delta-splotch - scan-distance
    set delta-splotch delta-splotch / 2
    set scan-distance 1

    right 90 ;; face back to origanal direction
```

```
    set splotch-360 patches in-cone scan-distance 360
    if delta-splotch >= 0
    [
      while [ ( (  count splotch-360 with [ pcolor != gray ] ) < 2) and ( scan-distance
< delta-splotch ) ]
       [
         set ycor ycor + 1
         set scan-distance scan-distance + 1
         set splotch-360 patches in-cone scan-distance 360
       ]
     ]
   ]
    if heading = 0 or heading = 180
    [
     right 90
     let East-West-splotch patches in-cone scan-distance 90

     while [ (count East-West-splotch with [pcolor != gray ] ) < 2 ]
     [
       set scan-distance scan-distance + 1
       set East-West-splotch patches in-cone scan-distance 120
     ]

     set delta-splotch scan-distance
     set scan-distance 1

     left 180 ;; rotate to look at other side
     set East-West-splotch patches in-cone scan-distance 90

     while [ (count East-West-splotch with [pcolor != gray ] ) < 2 ]
     [
       set scan-distance scan-distance + 1
       set East-West-splotch patches in-cone scan-distance 90
     ]

     set delta-splotch delta-splotch - scan-distance
     set delta-splotch delta-splotch / 2
     set scan-distance 1

     right 90 ;; face back to origanal direction
     set xcor  xcor + delta-splotch
```

```
   ]
  ]
end
```

iRobot Create- Create going through the maze

```
int min_wall_at_left = 525;       // value that is further from the wall
int max_wall_at_left = 680;              // value that is closest to the wall

int min_wall_in_front = 101;     // value that is further from the wall
int max_wall_in_front = 500;     // value that is closest to the wall

int min_wall_at_right = 460;     // value that is further from the wall
int max_wall_at_right = 790;     // value that is closest to the wall

int wall_in_front = 0;
int wall_count = 2;

void right_turn();
void left_turn();
void drive_current_angle();

int driving_angle = 720;

int main()
{
        printf("Hello, Maze!\n");

        set_each_analog_state(1, 0, 0, 0, 1, 0, 0, 1);                  // et sensors plugged
into ports 0, 4, and 7
        while(create_connect() < 0)                     // connects CBC to Create
        {
                create_connect();
                msleep(100);
        }
        msleep(100);
        printf("CBC connected to Create!\n");                                                   //
Gives time for Create and CBC to connect

        set_create_total_angle(720);
        msleep(2000);
```

```
	while(1)
	{
		drive_current_angle();

		if (analog10(4) < max_wall_in_front)                                    //
the sensor in the front is engaged
		{
			if (analog10(0) > max_wall_at_left)
	// the sensor to the left of the Create is driving till the max value is read than
movies to the right to even itself out
			{
				create_drive_direct(100, 200);
				msleep(100);
			}
			if (analog10(7) > max_wall_at_right)
	// the sensor to the right of the Create is driving till the max value is read than
movies to the left to even itself out
			{
				create_drive_direct(200, 100);
				msleep(100);
			}
			create_drive_straight(150);
			msleep(100);
		}
		if (analog10(4) > max_wall_in_front)                                    //
the sensor to in the front of the Create is driving till the max value is read than stops
		{
			create_stop();
			wall_in_front = 1;
		}

		if(wall_in_front == 1)
		{
			if(wall_count >=0)
			{
				right_turn();
			}
			else
			{
				left_turn();
```

```
                }
                wall_count = wall_count - 1;
                wall_in_front = 0;
            }

        }
        create_stop();
        return 0;
}



void right_turn()
{
        int target_angle = get_create_total_angle(.1) - 90;
        driving_angle = target_angle;
        while(get_create_total_angle(.1) > target_angle)
        {
            create_spin_CW(100);
        }
        create_stop();

        while(get_create_total_angle(.1) != target_angle)
        {
            if(get_create_total_angle(.1) > target_angle)
            {
                create_spin_CW(55);
                msleep(175);
                create_stop();
            }
            if(get_create_total_angle(.1) < target_angle)
            {
                create_spin_CCW(55);
                msleep(175);
                create_stop();
            }
        }
}

void left_turn()
{
```

```
	int target_angle = get_create_total_angle(.1) + 90;
	driving_angle = target_angle;
	while(get_create_total_angle(.1) < target_angle)
	{
		create_spin_CCW(100);
	}
	create_stop();

	while(get_create_total_angle(.1) != target_angle)
	{
		if(get_create_total_angle(.1) > target_angle)
		{
			create_spin_CW(55);
			msleep(175);
			create_stop();
		}
		if(get_create_total_angle(.1) < target_angle)
		{
			create_spin_CCW(55);
			msleep(175);
			create_stop();
		}
	}
}
void drive_current_angle()
{

	if(get_create_total_angle(.1) != driving_angle)
	{
		if(get_create_total_angle(.1) > driving_angle)
		{
			create_drive_direct(100, 200);
			msleep(50);
		}
		if(get_create_total_angle(.1) < driving_angle)
		{
			create_drive_direct(200, 100);
			msleep(50);
		}
	}
}
```

Sensor Data Refinement:

The Accuracy of the ET Sensor

Danielle Garcia

School of Dreams Academy
daniellegarcia4541@gmail.com

Sensor Data Refinement:
The Accuracy of the ET Sensor

1 ET Sensor

Every student-team that participates in Botball receives an ET sensor with their Kit of Parts. With each ET sensor comes with a warning: After the sensor values reach a certain point, the farther values will resemble the closer values. The image shown here represents the light values reflected from the surface to the sensor.[1]

In a recent project, the accuracy of the ET sensor was tested. The sensor was tested under three light conditions and with two different colored sides (black and white). The reasons for the three lighting conditions was to see if light reflectivity affects the accuracy. The different colored sides was to test the focal point of the ET Sensor.

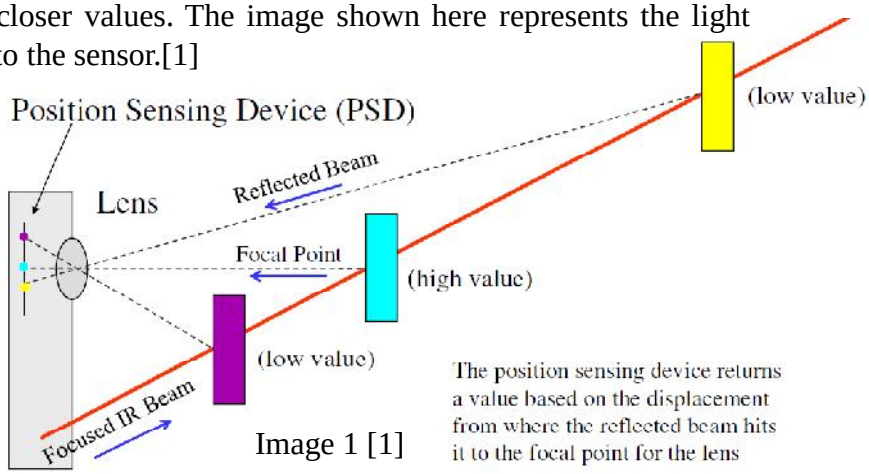


Image 1 [1]

2 Testing the ET Sensor

Testing the ET Sensor was a long process. There was a total of six tests conducted. Before the sensor was tested, research was done to make sure on the maximum distance that could be sustained (80cm) [2]. After the information was collected, the supplies was gathered: A block of wood with a black and white side, particle board, ET Sensor, and a CBC. (The KIPR LINK was purposely not used due to the project was started before our institution was provided with the KIPR LINK.) The particle board was marked at each centimeter for a total distance of 80

centimeters . The ET sensor was attached to a smaller block of wood to keep it stable. The block of wood had one side covered with white paper and another with black paper.

2.2 Lighting the Set

The three types of light that were used are as follows: Relative Lighting (classroom-type of lighting), Bright Lighting (strong florescent lights), and Dim Lighting (poor lighting). With each type of lighting, there was two tests conduced; one for the black and the white side. Below is one

of the tables from the Relative Lighting-White Side:

Key

-values that are similar

-peak values

Distance away from wood board (cm)	Value of Sensor
1	400
2	420
3	500
4	500
5	700
6	940
7	950
8	900
9	800
10	710
11	640
12	570
13	520
14	480
15	450
16	425
17	390
18	370
19	350
20	330
21	320
22	305
23	290
24	275
25	270
26	255
27	250
28	235
29	230
30	220
31	200
32	200
33	200

76	75
77	75
78	76
79	80
80	80
34	185
35	180
36	175
75	80
37	180
38	165
39	160
40	160
41	155
42	150
43	150
44	140
45	145
46	140
47	140
48	135
49	130
50	120
51	115
52	120
53	118
54	115
55	120
56	117
57	113
58	110
59	107
60	109
61	105
62	107
63	102
64	97
65	99
66	93

67	96
68	96
69	90
70	85
71	81
72	81
73	83
74	88

This table is the representation of the first test that was completed. There were five other test similar to the one above.

3 Results

The results from all six tests are in the following graph.

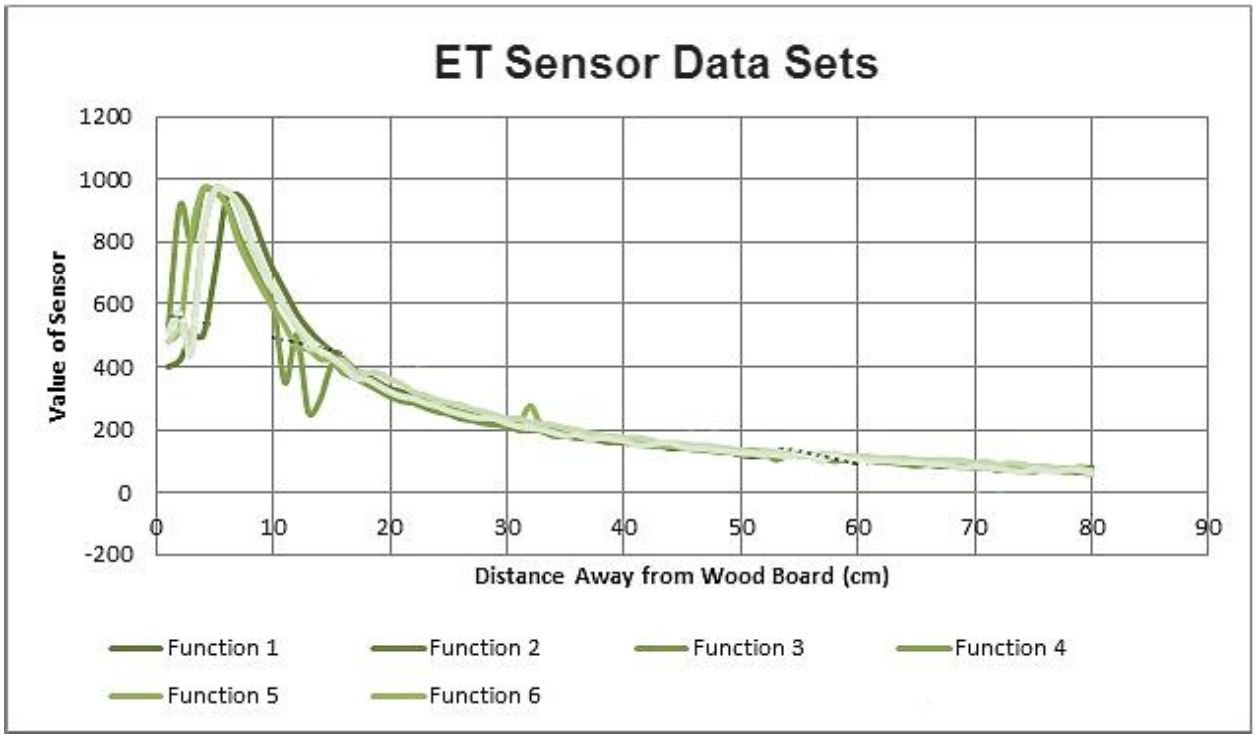


Image 2

4 Conclusions

Originally, the plan was to develop an equation to solve the inaccuracy of ET sensor, but it was soon realized that a mechanical solution was a much more plausible solution for now. During the testing, the brightly lit area and the dim lit area were close in data values; while the relative lit areas tended to fluctuate more. All the results provided a much needed outcome, they all peak at about seven centimeters. A simple solution for accurate readings would be to put a small Lego piece or foam that is 6 to 8 centimeters to create the most accurate reading for the sensor during game play.

Figure Table

1 (2013). Optical Rangefinder. *Botball '13 Workshop , 1*(4.6), 167. Retrieved December 17, 2012, from the KIPR database.

References

2 Botball 2011. (n.d.). *Sensor and Motor Manual.* Received Jan. 13, 2013

Bibliography

Garcia, Danielle. Sensor Data Refinement: The Accuracy of the ET Sensor. Tech. Norman: KISS Institute of Practical Robotics, 2013. Print.

Winnefeld, James A., Jr., and Frank Kendall. "Unmanned Systems Integrated Roadmap."*Higher Logic Download*. U.S. Department of Defense, n.d. Web. 13 Nov. 2013.

"A Roadmap for US Robotics: From Internet to Robotics." *US Robotics*. Ed. Georgia Institute of Technology, University of Southern California, John Hopkins University, Carnegie Mellon University, and Etc. Computing Research Association, 01 May 2009. Web. 13 Nov. 2013.

Adam, Neil J., and Etc. "Autonomous Vehicles in Support of Naval Operations." *Higher Logic Download*. National Academy of Science, 2005. Web. 4 Dec. 2013.

"Autonomous UAV Mission System (AUMS)." *Autonomous UAV Mission System (AUMS)*. Ed. System Center Pacific. United States Navy, n.d. Web. 16 Oct. 2013.