Simulated Framework for Post Disaster Robotic Response

Christina Palicelli

December 11, 2016

0.1 Introduction

The project develops a framework for distributed robotic response to the problem of mapping post disaster enviroments. In particular the emphasis is placed on creating a method to examine the impact of communication deadzones on the mapping algorithm. This project developed a frontier find algorithm for the overall mapping and use the Dijkstra algorithm to find the shortest path to the frontier. Results were obtained to determine the response of this mapping procedure to diffrent enviroments with and without communication deadzones as well as with diffrent communication ranges and number of robots.

0.2 Background

In the event of a disaster search and rescue efforts may be delayed until it is safe for humans to enter the enviroment, however this must be balanced with the fact that delays in these efforts can signifcantly lower the potential for survivors to recover. An important task in disaster response is the ability to develop a map of the disaster zone and search for hazards and/or surivivors. This is an optimal task for a robotic swarm, hazards for human first responders can be identifed and thus addressed or avoided, resources can be targeted to where they are most needed, areas which are too small for humans can be explored and using swarm methodology this process can be expidited with a large number of robots. The primary goal is for robots to completely explore the enviroment since missing a small area can result in the loss of critical information such as the location of an unstable element or a survior. Futhermore in a post disaster enviroment communication is often limited, this can be seen in both the robotic responses to 9/11 and the response to the fuckashima disaster where robotic response were limited due to communication dead zones from enviromental factors.

0.3 Setup

0.3.1 Map Development

The overall enviroment is described by a cell-based map stored in a matrix. Each cell is defined by a state; with values 1 for free space and 2 for a obstructed area. To test the simulation three maps were defined, the base map, Map 1, which is simply a walled perimiter with the center being free space. This map was used to test overall functionality of the exploration methodology without having to worry about additional edge cases from other obstructions in the enviroment.

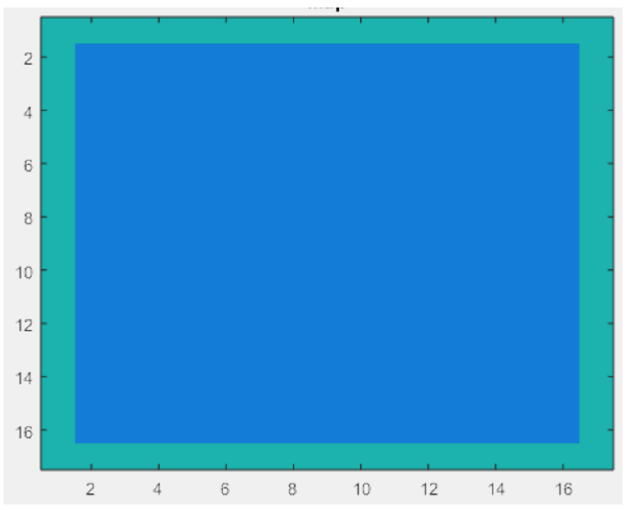


Figure 1: Map 1 - walled perimiter, center free space

The second test map was deeloped in order to explore issues with basic wall formations in the way, but with limited obstacles as to not overcomplicate the map.

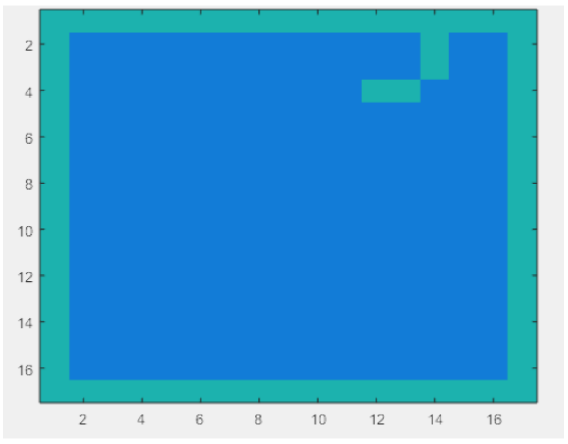


Figure 2: Map 2 - simple obstacle

The third map was developed to simulate a real world room and the potential wall configurations present in this enviroment.

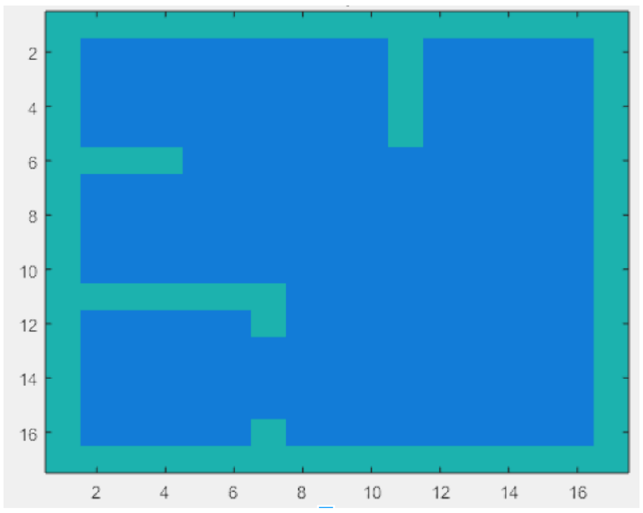


Figure 3: Map 3 - real world simulation

Although these three map setups were choosen for the simulation test any configuration could be tested. In order for the robots to know where the edge of the search area is all maps are surroundded by a wall perimiter - this can be artifically implemnted if the search distances are known.

Start Positions

Once the maps were generated a list of potential start positions, any space which is free, for the robots was generated in order of precedence from the upper right corner of the map. These is the location where the simulation will intially place the robots. Below is the code used to generate the start positions, where m and n are the dimensions of the map matrix.

```
for k= 1:1:n
    for l = m:-1:1
        if maps(k,l) == 1
            starty=[start l];
            start=[starty k];
        end
    end
end
```

Figure 4: Start location list

0.4 Implementation

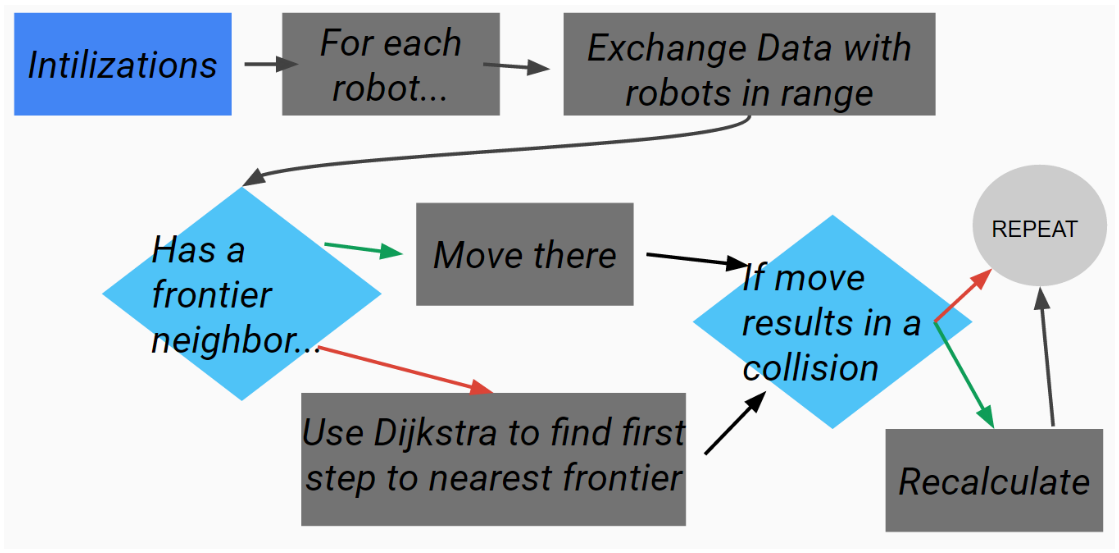

Intilizations
For each robot...
Exchange Data with robots in range
Has a frontier neighbor...
Move there
If move results in a collision
REPEAT
Use Dijkstra to find first step to nearest frontier
Recalculate

Figure 5: Flowchar of overall implementation

0.3.2 Robot Intilizations

Robots were defined as objects with the parameters of communication range, their local best known map, their current position and the position at the previous time step. An array of these robot object was created to the user inputed desired number of robots and at the user inputed desired communication range. The local best known map for each robot is intilized as all cells being unknown (value of zero).

0.4.1 Communication Information Exchange

Communication Deadzone

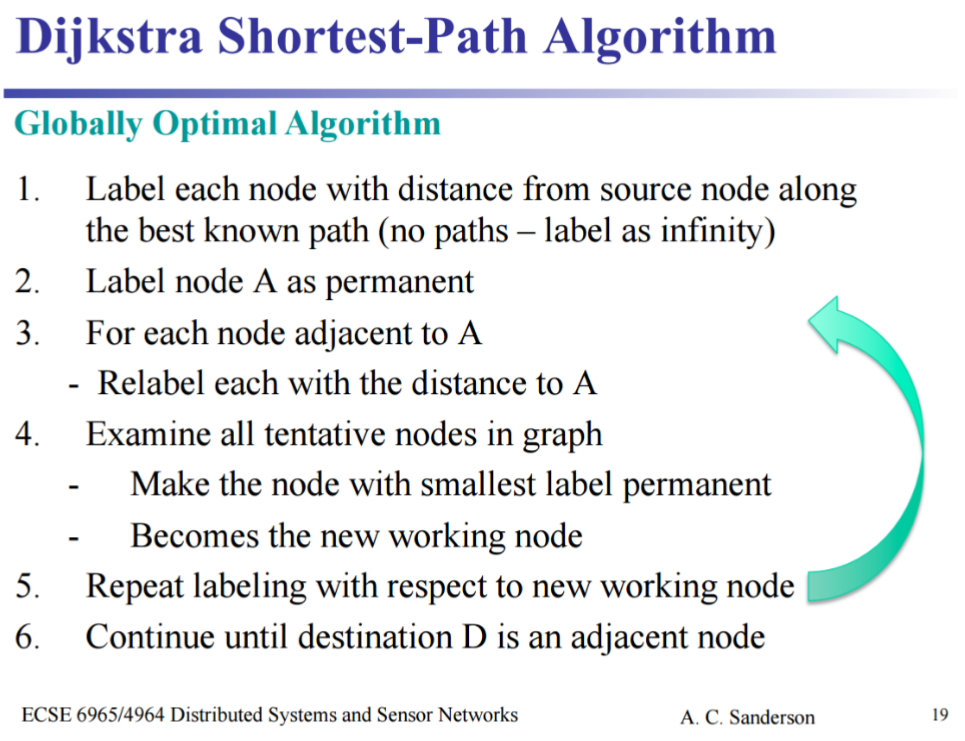
To test the robustness and effectiveness of the simulation and the choosen mapping scheme in the presence of communication deadzones a map of the same dimensions as the search zone was generated to indicate if a cell was able to be communicated out of or not. For the purpose of this simulation and test the communication dead zones were randomized, but any desired communication dead zone map could be created within this implmentation.

General Communication Procedures

Before any move is executed the robot who is implementing the move intiates communication with any robots within it's communication range, communication will fail is either robot is in a dead zone. The communication between the two robots in each others communication range consists of them updating their maps with any free, obstructed or frontier cells that the other robot has but they do not.

0.4.2 Frontier Find Algorithm

The algoithm developed to decide where a robot should move was developed based off of the algorithm inmplemented in [1]. Each move starts by setting the current position to free, and setting any unknown neighbring spaces to either frontier or wall. If a neighboring space is in the map as a frontier the robot moves there in the order of precedence; up, left, right then down. If none of the neighboring cells are labled as frontier the first step towards the nearest frontier is determined using the dijkstra algorthim between the current node and each frontier node using the known free cells as the nodes on the path. The Dijkstra algorithm was implemented as described in the lecture slides, shown below.



Dijkstra Shortest-Path Algorithm

Globally Optimal Algorithm

1. Label each node with distance from source node along the best known path (no paths – label as infinity)
2. Label node A as permanent
3. For each node adjacent to A
   - Relabel each with the distance to A
4. Examine all tentative nodes in graph
   - Make the node with smallest label permanent
   - Becomes the new working node
5. Repeat labeling with respect to new working node
6. Continue until destination D is an adjacent node

ECSE 6965/4964 Distributed Systems and Sensor Networks A. C. Sanderson 19

Figure 6: Overview of Dijkstra algorithm implementation [2]

0.4.3 Collision Avoidance

In order to avoid robots running into each other a collision avoidence scheme was implemented. In this scheme if the cell a robot wishes to move to after determination of the move as described above is occuppied the robots map updates that cell to be free space and then executes the procedure for determining the next place to move to again, which will give a diffrent result based on the updated map.

0.5 Results

The algorithm was run for each of the three maps with and without communication dead zones for 1,2,4 and 6 robots with a communication range of 1,2,4,6,8 and 10. These results were then plotted to determine the overall trend of preformance. In the data it is noted that as the number of robots increases the number of iterations it takes to completely map the enviroment decrases, this was an expected observation but it holds true across all trials. Additionaly it was noted that after a certain point increasing the communication range does not result in the reduction of iterations for mapping the enviroment.

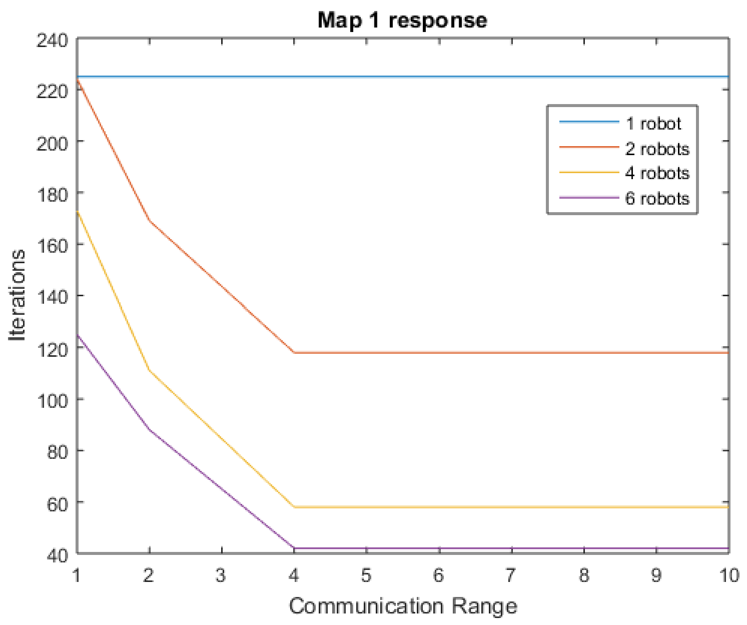


Figure 7: Map 1, no communication deadzones response

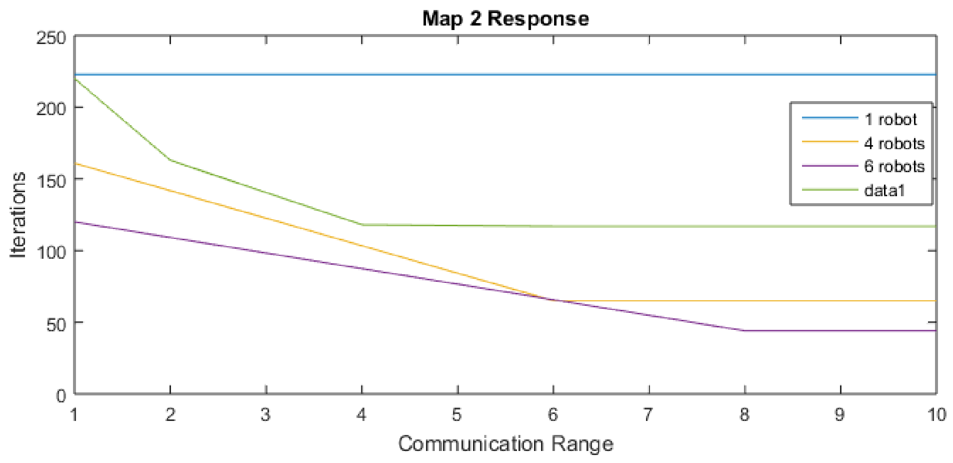


Figure 8: Map 2, no communication deadzones response

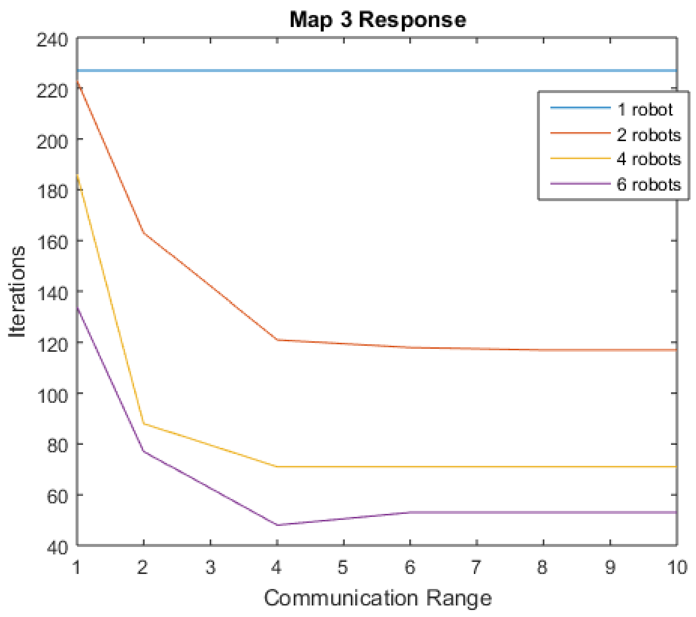


Figure 9: Map 3, no communication deadzones response

With the introduction of communication deadzones the response was slower, on average by 38.25 iterations. However the overall trends stayed the same, indicating robustness of this implementation in the face of potential communication deadzones.

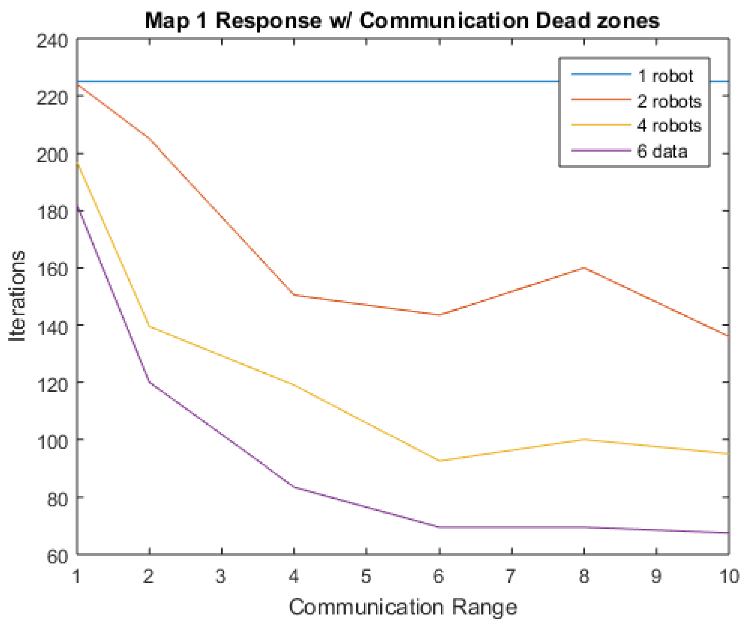


Figure 10: Map 1, with deadzones response

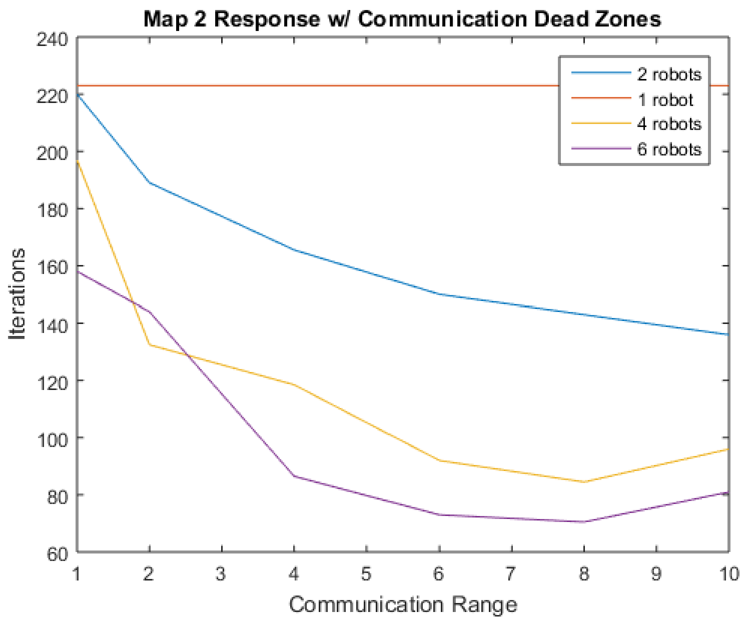


Figure 11: Map 2, wit deadzones response

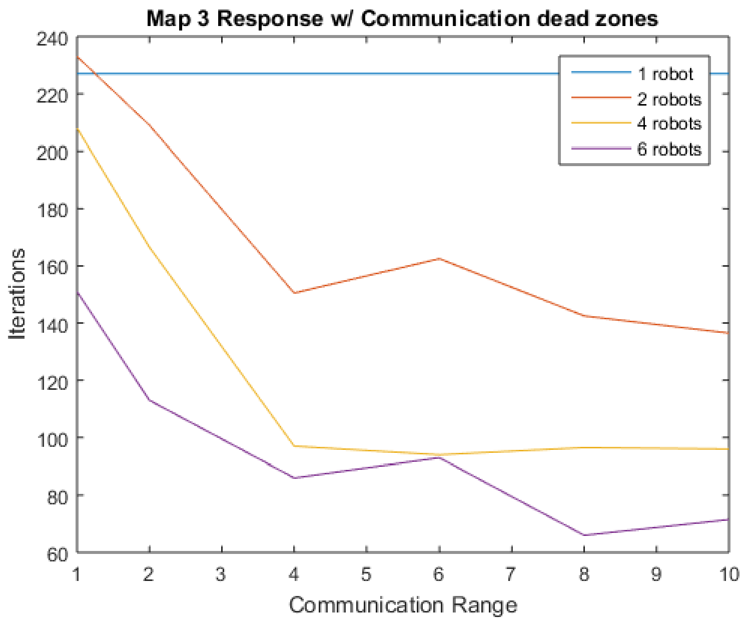


Figure 12: Map 3, nwith deadzones response

0.6 Modularity

This simulation was designed as a framework and therefore emhasis was placed on creating it in a method which is modular. In its current configuration the number of robots, communication range, maps and deadzone are easy to change. At the moment the communication range is homogenous amoung all robots, however with small changes to the robot intilization code it will be possible to make the communication ranges non-homogenous which mirrors what is likely to happen in the real world. It is also relativly easy to swap out the shortest path to frontier algorithm, this was done during the development of the code as the intial method was using the shortest straight path which had the issue of getting stuck in corners. Changing the overall framework is possible but would be a greater time investment.

0.7 Disscussion + Future Work

This simulation has many applications and potential future applications. The most basic expansion would be to extend this simulation to cover more cases, in particular cases which mimic real world incidents. This will be possible at least to mimic the enviroment of the fuckashma disaster as map of the enviroment have been released which will allow for the simulation to be run on these test cases. Another feature to implement into the algorithm would be as it is searching for a frontier to move towards, consider where other robots are located and move in opposite directions from them, this would hopefully allow the area to be mapped faster. It would also be intresting to implement caculation of other variabls which are intresting for this type of application such as energy consumption, although these would require more knowelege of the particular platform which would be searching the area. Currently general platforms for robotic search and rescue are not avaible and every disaster is responded to by a diffrent collection of robots, often who are not suited to the task, it is for this reason it is important to maintain a high level of flexability in the simulation. It is also possible to develop this method for implementation on a hardware platform. A last thing to do to make this simulation easier to use would be to develop a graphical user interface.

This simulation providdes a starting place from which to develop and analyze diffrent approaches to the problem of mapping and searching an unknown enviroment

0.8 Appendices

0.8.1 MatLab Code

```
clear all;
num = input('input the number of robots ') ;
width = 17;%input('input the width of the map ');
height = 17;%input('input the height of the map ');
comm_range = input('input communication range');
actual = Map(width,height);
robots = [];

start = [];
starty = [];
for k= height:-1:1
    for l = 1:1:width
        if actual(k,l) == 1
            start=[start l];
            starty=[starty k];
        end
    end
end
for i =1:1:num
    if length(start) < num
        error('there are not enough places to put the robots');
    end
    inital = [start(i) starty(i)];
    inital;
    %make robot instances
    robots = [robots Robot(width,height,inital,comm_range)];
end
figure(1)
image(actual.*15)
title('Intial␣Setup');
iteration = 0;
done = 1;
X=1;
dead = DEATH(width);
while(1)
    iteration = iteration + 1;
    for j = 1:length(robots)
        %Setup
        current = robots(j);
        FLAG = robots(j).stuck;
        map = current.map;
        pos = robots(j).position;
        prev = current.prev;
        [~,local] = Collide(robots, pos);
        map = Comm(local,pos,robots,map,j,dead);
        [move,map] = Move(actual,map,pos,local);
        %check to see if move will result in a collision
        [hit,~] = Collide(robots, move);
        if hit==1
            %make space with other robot free
            map(move(1), move(2)) = 1;
            robots(j).map = map;
            [move,map] = Move(actual, map, pos,local);
            [hit,~] = Collide(robots,move);
            if hit ==1
                move = pos;
            end
        end
        move;
%update object parameters
        map(move(1), move(2)) = 4;
        map = Comm(local,pos,robots,map,num,dead);
        robots(j).map = map;
        figure(j+1);
        image(map.*15);
        robots(j).position = move;
        robots(j).prev = pos;
        robots(j).stuck = FLAG;
        Result(robots);
    end
end
final = Result(robots);
iteration
```

Figure 13: main.m

```
function [ hit,local ] = Collide(robots,new)
    hit = 0;
    local = [];
    for robot = 1:1:length(robots)
        x1 = robots(robot).position(1);
        y1 = robots(robot).position(2);
        local(robot,1) = x1;
        local(robot,2) = y1;
        if (x1 == new(1) && y1 == new(2))
            hit = 1;
        end
    end
end
```

Figure 14: Collide.m

```
function [map] = Comm(local,pos,robots,map,num,dead)
x0 = pos(1);
y0 = pos(2);
range = robots(num).comm_range;
[x,y] = size(map);

if ~dead(x0,y0)
    for robot=1:1:length(robots)
    x1 = local(robot,1);
    y1 = local(robot,2);
    dist = sqrt((y1-y0)^2 + (x1-x0)^2 );
    new = robots(robot).map;
    if dist <= range && ~dead(x1,y1);
        for a=1:x
            for b = 1:y
```

```
                    if map(a,b) == 4
                        map(a,b) = 4;
                        new(a,b) = 1;
                    elseif new(a,b) == 4
                        new(a,b) = 4;
                        map(a,b) = 1;
                    elseif map(a,b) == 2 || new(a,b) == 2
                        map(a,b) = 2;
                        new(a,b) = 2;
                    elseif map(a,b) == 1 || new(a,b) == 1
                        map(a,b) = 1;
                        new(a,b) = 1;
                    elseif new(a,b) == 3 || map(a,b) == 3
                        map(a,b) = 3;
                        new(a,b) = 3;
                    end
                end
            end
        end
        robots(robot).map = new;
        end
end
robots(num).map = map;
map = map;
end
```

Figure 15: Comm.m

```
function s = Frontier_Find(map, pos,local)
    x = pos(1);
    y = pos(2);
    [h,w] = size(map);
    [row, col] = find(map==3);
    front = [row,col];
    free_neighbor = [];
    count = 1;
    for j = -1:2:1
        if map(x+j,y) == 1
            free_neighbor(count,1) = x+j;
            free_neighbor(count,2) = y;
            count = count + 1;
        end
        if map(x,y+j) == 1
```

```
                free_neighbor(count,1) = x;
                free_neighbor(count,2) = y+j;
                count = count+1;
            end
        end
        d = [];
        tags = [];
        %setup for Dijkstra
        nodes = [x y];
        for j = 1:h
            for i = 1:w
                if map(j,i) == 1
                    nodes = [nodes;j i];
                end
            end
        end
        nodes = [nodes;1 1];
        [num, ~] = size(nodes);
        distances = [];
        next = [];
        [A,~] = size(front);
        for a=1:A
             nodes(num,1) = front(a,1);
             nodes(num,2) = front(a,2);
            [xd,yd,ld] = dijk(nodes);
            distances = [distances ld];
            next = [next;xd yd];
        end
        next;
        [~,I]=min(distances);
         s = [next(I,1),next(I,2)];
end
```

Figure 16: FrontierFind.m

```
function [move,map] = Move(birdseye,map,position,local)
        x = position(1);
        y = position(2);
        FRONT = 0;
        map(x,y) = 1;  % Mark current cell as free
        %mark neighbors as wall or frontier
        for i = -1:2:1
        if birdseye(x,y+i)~= 2 %not a wall in global map
            if map(x,y+i) ~=1 %not open in local map
                map(x,y+i) = 3; %set to frontier
                FRONT = 1;
            end
        else
            map(x,y+i) = 2; %set to wall in local map
        end

        if birdseye(x+i,y)~= 2 %not a wall in global map
            if map(x+i,y) ~=1 %not open in local map
                map(x+i,y) = 3; %set to frontier
                FRONT = 1;
            end
        else
            map(x+i,y) = 2; %set to wall in local map
        end
    end

    %if known frontier is in "area"
    if FRONT == 1
        if map(x,y+1)==3
            position= [x,y+1];
        elseif map(x-1,y)==3
            position= [x-1,y];
        elseif map(x+1,y)==3
            position= [x+1,y];
        elseif map(x,y-1)==3
            position= [x,y-1];
        end
    else
        new = Frontier_Find(map,position,local) ;
        if isempty(new)
            new(1) = x;
            new(2) = y;
        end
        position = [new(1), new(2)];

    end
    move = position;
end
\begin{figure}[h!]
\caption{Move.m}
```

```
\label{fig:}
\end{figure}
\begin{lstlisting}[frame = single]

classdef Robot
    properties (SetAccess = public, GetAccess = public)
        comm_range
        map
        position
        prev
    end
    methods
        function R = Robot(n,m,i,comm)
            %n and m are the intial matix size
            %i is the intial location of the robot (x,y)
            %comm is the communication range
            R.map = zeros(n,m);
            R.position = i;
            R.prev = [i(1)-1, i(2)];
            R.comm_range = comm;
            R.stuck = 6;
        end
    end
    methods(Static)
    end
end
```

Figure 17: Robot.m

0.8.2 Refrences

[1]Y. Wang, A. Liang and H. Guam, "Frontier-based Multi-Robot Map Exploration Using Particle Swarm Optimization", 2011.

[2]A. Sanderson, "Lecture 10. Wireless Sensor Network: Routing", Rensselaer Polytechnic Institute, 2016.

[3]D. Stormont, "Autonomous Rescue Robot Swarms for First Responders", in IEEE International Conference on Computational Intelligence for Homeland Security and Personal Safety, Orlando, FL, 2005.

[4]M. Subhan and A. Bhide, "Study of Unmanned Vehicle (Robot) for Coal Mines", International Journal of Innovative Research in Advanced Engineering, vol. 1, no. 10,

2014.

[5]S. Grayson, ”Search and Rescue using Multi-Robot Systems”, School of Computer Science and Informatics, University College Dublin, 2014.