2010

Department of Computer Science and
Engineering of BRAC University

Taufiq Abdur Rahman 05210010
Auninda Rumy Saleque 06101003
Hasan Ameen Salahuddin 04101010

Under the Supervision

of

Dr. Md. Khalilur Rhaman

[CARTOGRAHER]

Cartography comes from the combination of two Greek words: chartis meaning map and graphine meaning to write. Cartography is the study and practice of making maps, and hence a cartographer is an entity – human or machine – that possesses the ability of charting an area and presenting it in a visual form. In this case, it is an autonomous system that moves around on its own in a given area, avoiding obstacles, and charting the whole area into a map including the said obstacles as parts of the mapped area.

CARTOGRAPHER

A Thesis

Submitted to the Department of Computer Science and Engineering

of

BRAC University

by

TaufiqAbdurRahman

Student ID: 05210010

Auninda RumySaleque

Student ID: 06101003

HasanAmeenSalahuddin

Student ID: 04101010

Under the supervision

of

Dr. Md. Khalilur Rhaman

In Partial Fulfillment of the

Requirements for the Degree

of

Bachelor of Science in Computer Science and Engineering

December 2010

DECLARATION

We hereby declare that this thesis is based on the results found by ourselves. Materials of work found by other researcher are mentioned by reference. This Thesis, neither in whole nor in part, has been previously submitted for any degree.

Signature of
Supervisor

Md. Khalilur Rhaman

Dr. KhalilurRhaman

Signature of
Authors

TaufiqAbdurRahman

Auninda

Auninda RumySaleque

HasanAmeenSalahuddin 8th Dec'10

ACKNOWLEDGMENTS

We would like to begin by thanking our thesis supervisor Dr.KhalilurRhaman for allowing us to work on this thesis under his supervision and for his continuous support and guidance. It was a wonderful experience for us, and we shall remain grateful to him forever.

We would like to thank Dr. BelalBhuiyan for extending every possible help when asked for, and giving his valuable time to look up documents, related to our thesis, and forwarding them to us.

We would like to specially thank Jonayet Hossain (Student ID: 09221060), whose support and unfaltering dedication made it possible to fabricate the micro-controller circuit.

We would also like to thank our family members, especially our parents for their support and belief in us all throughout.

Last but not the least; we would like to thank Almighty Allah for His blessings, without which nothing is possible.

ABSTRACT

Cartography comes from the combination of two Greek words: *chartis*meaning map and *graphine*meaning to write. Cartography is the study and practice of making maps, and hence a cartographer is an entity – human or machine – that possesses the ability of charting an area and presenting it in a visual form. In this case, it is an autonomous system that moves around on its own in a given area, avoiding obstacles, and charting the whole area into a map including the said obstacles as parts of the mapped area.

Keywords: Cartographer, Automated Area Mapping, Image Processing, Obstacle Detection Using Laser Diodes, EmguCV, Shortest Path Calculation, A* Search using Manhattan Distance.

Table of Contents

Contents

INTRODUCTION

There was a time when men started to map areas in order to keep track of the regions they visited. This method soon became popular and came to be known as cartography or charting a map. As time passed by, through advancements of technologies, there have been many attempts to make automated devices that would do the map making work instead of a human being. Some attempts neared to success and some led to progressive state but, the whole process still falls under research category as it is not yet capable of handling a full-fledged real world scenario.

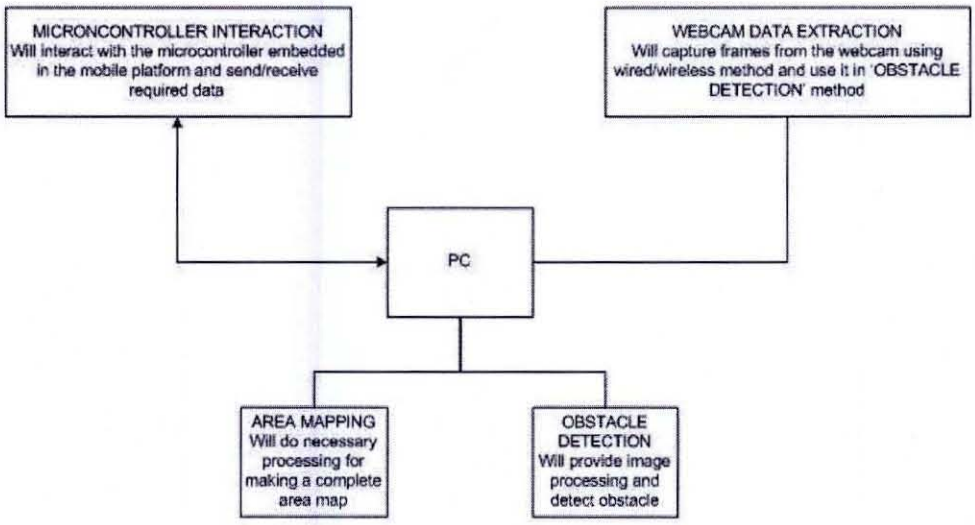
Here, we tried to follow previous researchers' footsteps and tried implementing a similar automated platform of our own. Due to hardware implementation limitations, our mapped area also needed to be following some restrictions.

PROJECT OVERVIEW

The basic idea surrounding the project was to develop a platform that possessed the ability to move around on its own in a given random space and to map the area for future references. The platform would be able to

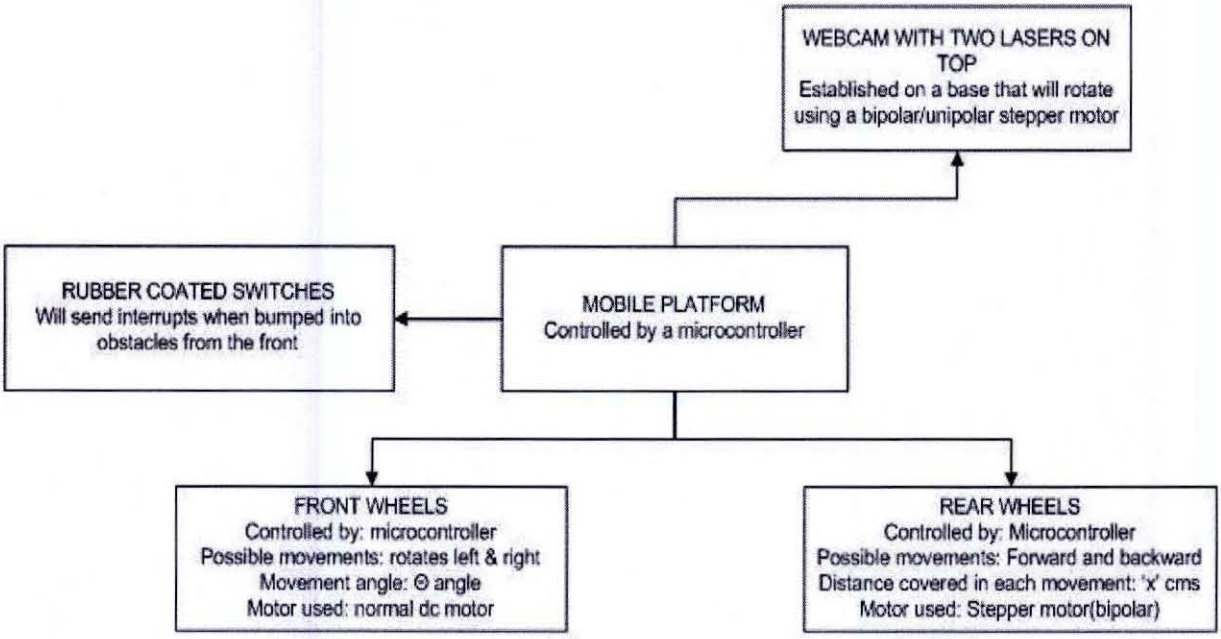
- Move in 6 directions, namely:
    - Front,
    - Front-Left
    - Front-Right
    - Back
    - Back-Left
    - Back-Right
- Detect obstacles in its path
- Estimate its distance from an obstacle
- Represent a 2d map of the environment
- Map the area using gridding techniques

The platform would then be able to send the data acquired to a remote system that would instruct the platform to perform the next series of tasks, which would lead to the traversal of the given environment in whole. The whole project is shown using a block diagram below



Block Diagram of the software part

Fig – 1: Block Diagram of the Software Side



Block Diagram of the Hardware Part

Fig – 2: Block Diagram of the Hardware Side

BASIC CONCEPTS

A* Search

A* is a computer algorithm that is widely used in path finding and graph traversal, the process of plotting an efficiently traversable path between points, called nodes. Performance and accuracy wise it is a well-known widespread method. It achieves better performance by using heuristic.

Manhattan Distance

Manhattan distance between two points is the sum of the absolute differences of their coordinates.

Image Processing

In the perspective to our work here, image processing can be explained as taking an image or video as input and extracting as much information as possible regarding that image/video and further use that data for our purpose.

Hardware Interfacing

As most of the logical tasks are complicated, it is no way possible to embed them in a microcontroller. So all the complicated tasks are handled by the core computer and the microcontroller is only used to perform the movement tasks. So, to sync both the computer and the microcontroller, we used serial communication to communicate with each other.

Occupancy Grid Mapping

The basic idea of the occupancy grid is to represent a map of the environment as an evenly spaced field of binary random variables each representing the presence of an obstacle at that location in the environment. Occupancy grid algorithms compute approximate posterior estimates for these random variables.

SOFTWARE IMPLEMENTATION

Hardware and Software Integration:

The movement of the car is handled using a PIC micro controller (**PIC16F877A**) and the necessary commands are passed to the controller through the pc. For handling this communication we utilized the serial communication channel and used **RS232 SerialCable** to sync the micro controller with the pc's software. The codes are written in **C#** using the '**System.IO.Ports**' library.

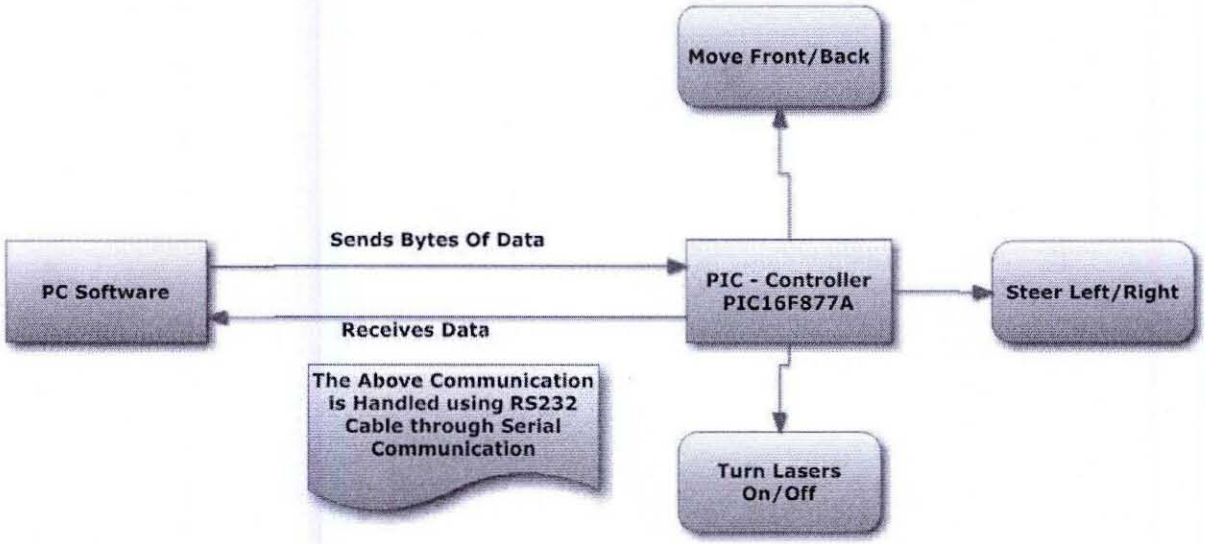


Fig - : A flow Chart Showing the Communication between the PC and the Micro Controller

OBSTACLE DETECTION

For detecting obstacles we used the '**EMGUCVAPI**' that provided us with necessary functions in implementing the obstacle detection component. Here, we looked for the brightest points in the view area that has the highest Red channel value with Blue and Green channel values much lower. This did help us to find the red laser pointers but it also distracted us from getting the real laser pointers as the laser pointers had larger area effect with higher red channel values.

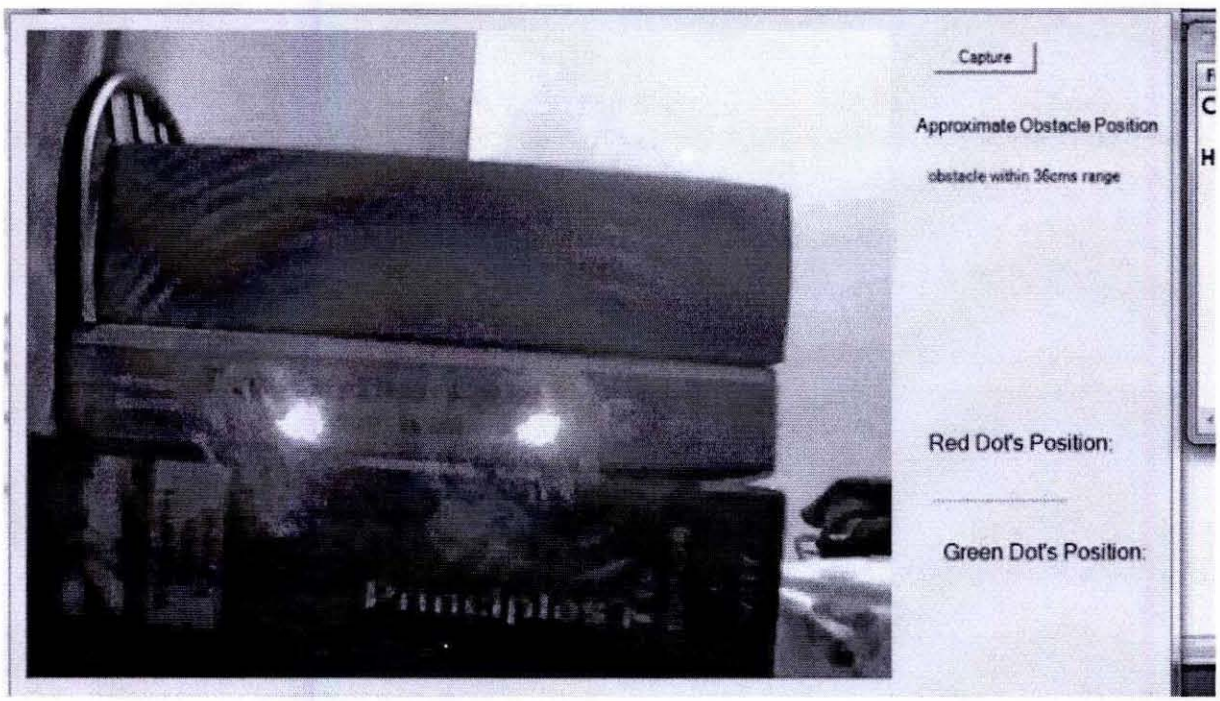


Fig – 3: Obstacle Detection Using Laser Pointers

To overcome this problem, we then detected the brightest points in the picture i.e. looked for pixel positions with white(RGB value of(255,255,255)) values. Now, this will easily detect the center white pointer as well as detect other white values too within the view area. Pointed in the below picture using black pointers.

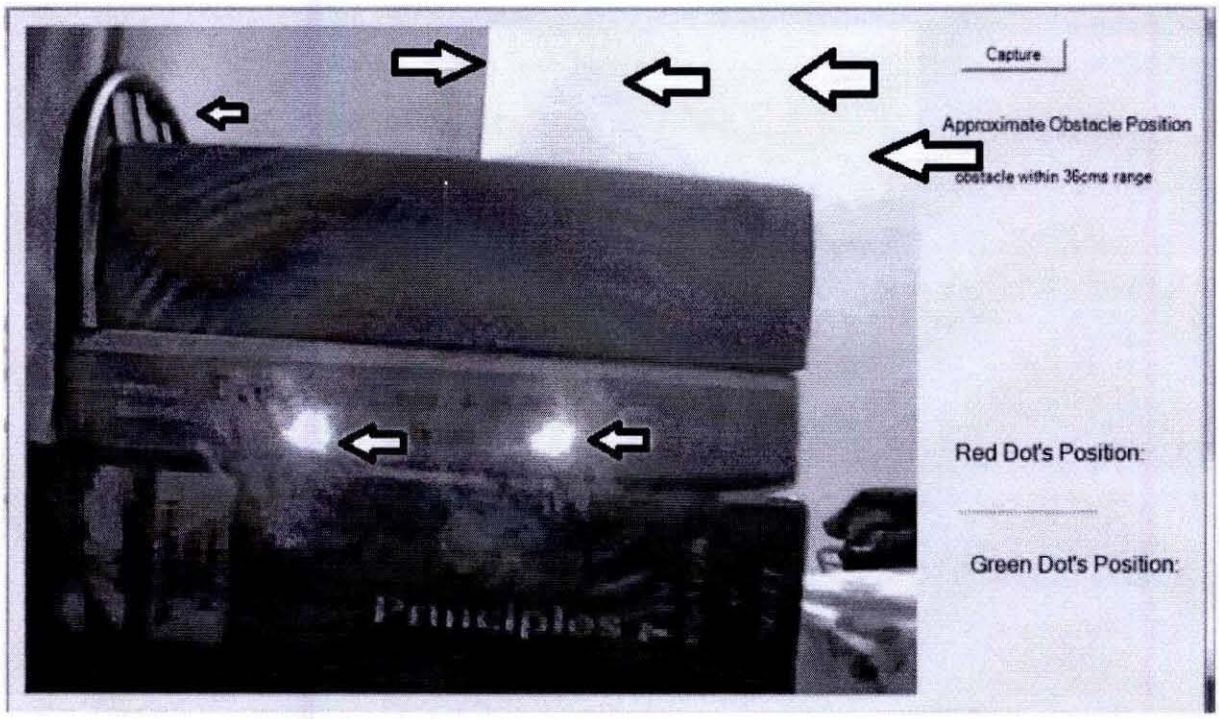


Fig – 4: Obstacle detection part - 2

To avoid such consequence we then refined our search of the pointer by first looking for a white value in the pixel and then looking within a pixel range just above that location for a higher red value where other two channels have really low values. This helped us to find the laser positions a bit more precise avoiding taking fake laser pointers. Pointed in the next figure the white pixel followed by a red pixel.

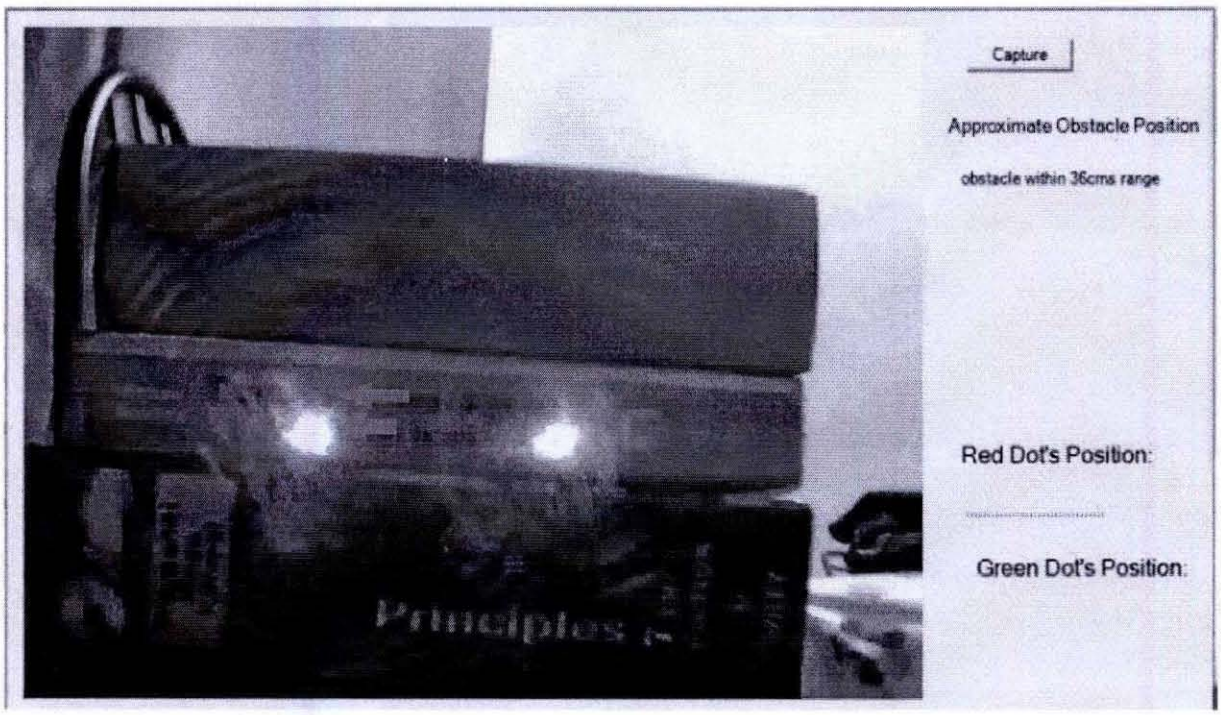


Fig - 5: Obstacle Detection part - 3

AREA MAPPING TECHNIQUE

For mapping the area we are utilizing the modified version of occupancy grid mapping technique. According to this technique, the mobile platform counts it's starting position as the initial state of the map and then gradually increases the mapped area while moving through the target enclosed area. Each cell of the mapped area holds certain information. They are:

- X // this is the x coordinate of the cell
- Y // this is the y coordinate of the cell
- M // This stores a value between 0-3 indicating if its unexplored, empty, visited or occupied

= Explored Cell

= Occupied Cell

= Visited Cell

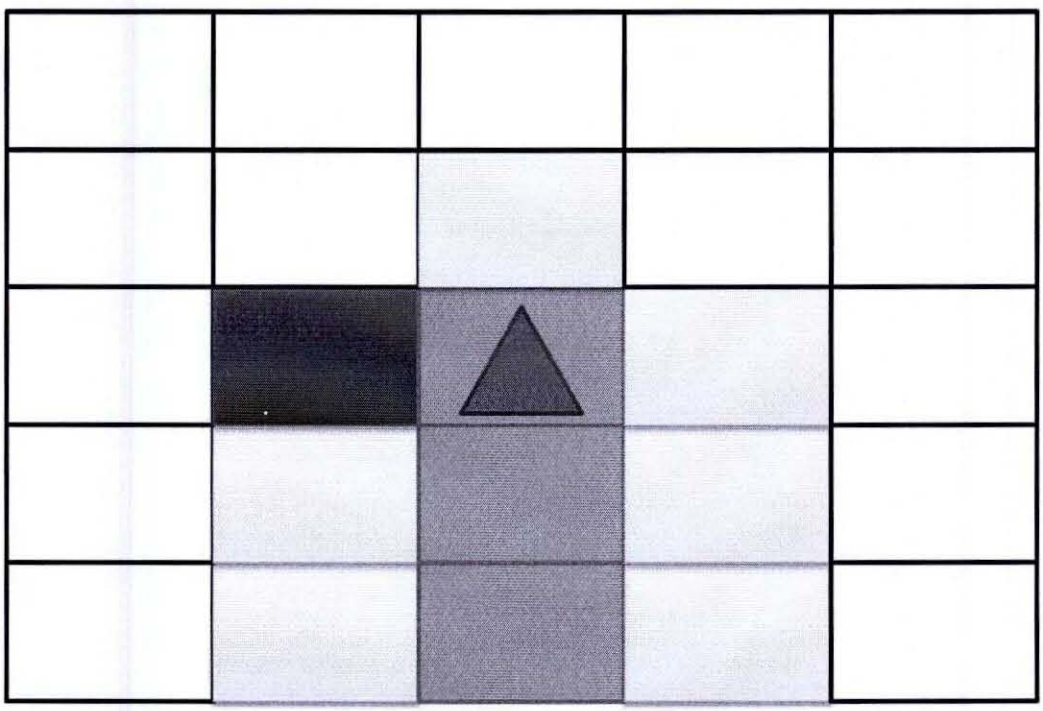


Fig – 6: Grid Mapping Technique

AUTOMATED VEHICLE MOVEMENT PATTERN

The vehicle follows a certain pattern while moving and mapping the area. The movement pattern is set according to a priority level. The levels are shown below from highest to lowest priority.

- Front
    - Front-Left
    - Front-Right
- Back
    - Back-Left
    - Back-Right

While moving according to this pattern, the vehicle always keeps track of its direction in perspective to its starting direction so that sync is not lost among the cells of the mapped area.

If the vehicle finds out that all its surrounding cells are already visited, then it sweeps its memory map to check if any cell is left unvisited or if it missed some spot to explore. If found then the vehicle uses the shortest path method to find the best way to reach the target position. If no more reachable and unexplored cells found, the vehicle stops mapping and shows the output.

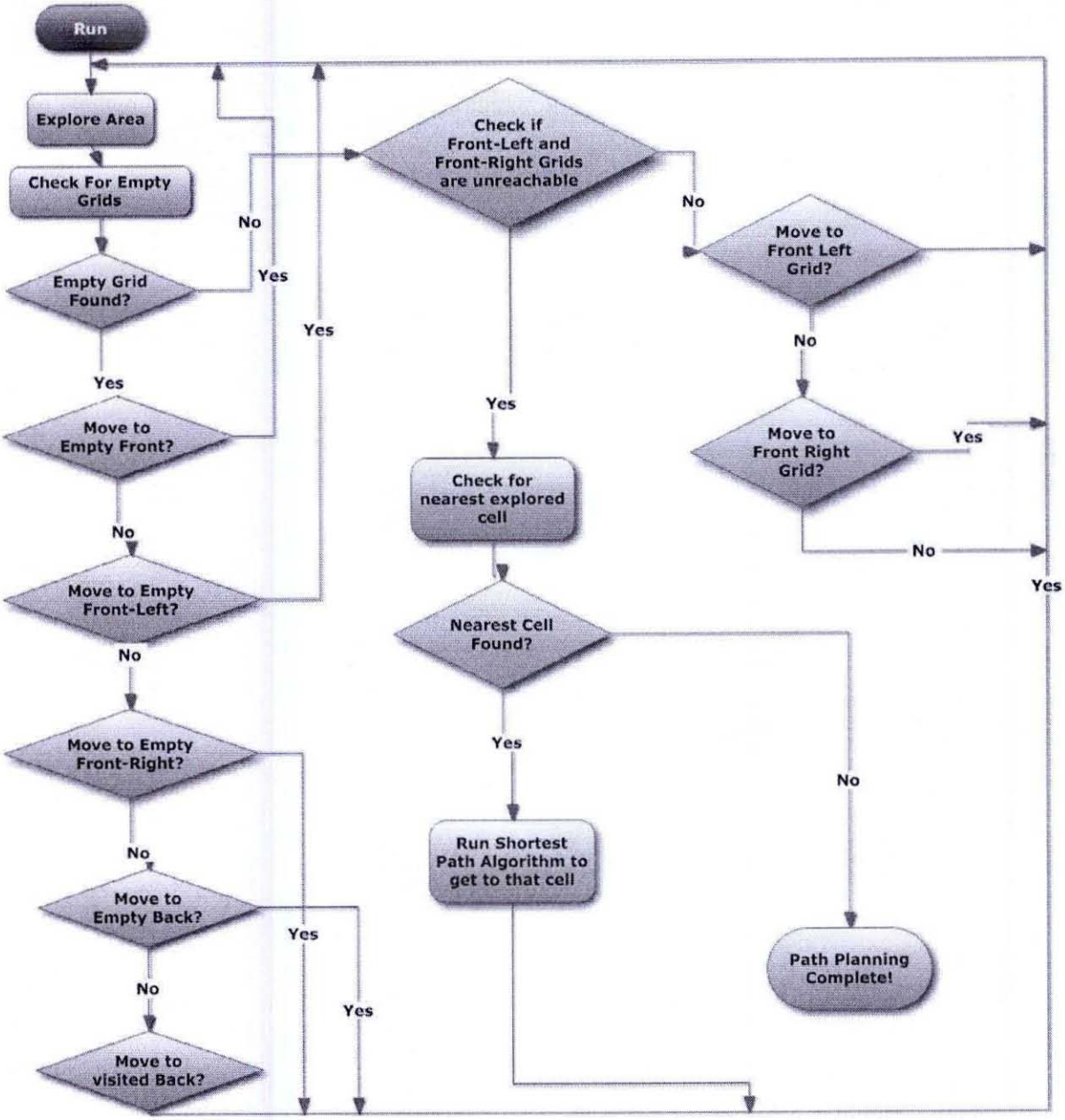


Fig – 7: (The Movement Pattern Flow Chart)

SHORTEST PATH CALCULATION

Shortest path calculation is used only for finding the path from a specific cell to a target cell using only the visited cells. We have utilized the A* search algorithm using heuristic method that is developed from Manhattan distance calculation. According to this method, we are using:

- A Priority Queue
- A Closed List

Initially we start by entering the specific grid into the **Priority Queue**and then removing it and placing it into a **Closed List**while entering all the neighboring cells of the specified cell to the priority queue setting their parent to be the specified cell. While that we are entering the grid cells into priority queue, each grid has some certain information stored within it for accomplishing the shortest path calculation. They are:

- Parent Node // This stores an integer value pointing to the cell's parent placed in the closed list
- Direction // This is used to store the car direction while in this grid
- G // This is the general jump cost from one cell to another
- H // This is the heuristic cost found using the Manhattan method
- F // This is the total cost calculated by summing up G and H

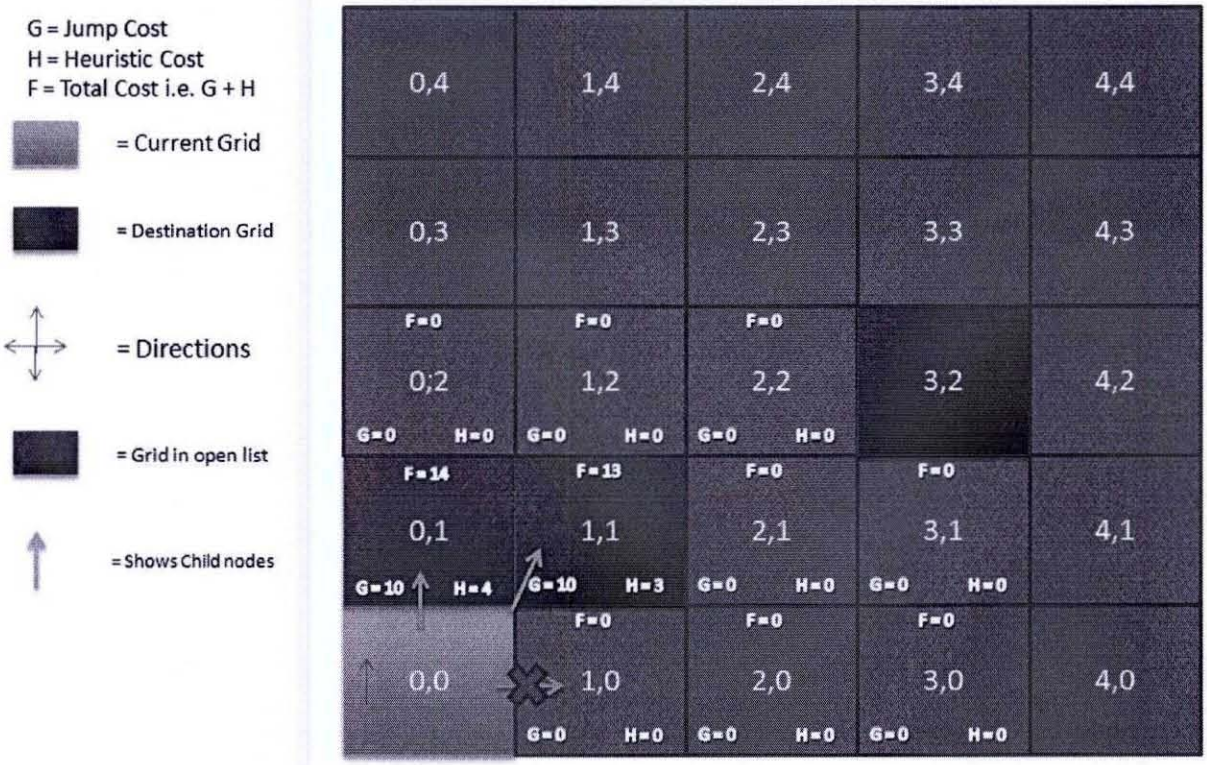


Fig – 8: A* Search for Calculating Shortest Path

HEURISTIC CALCULATION (MANHATTAN METHOD)

Here we calculated the heuristic cost by calculating the total number of squares moved horizontally and vertically to reach the target square from the current square, ignoring diagonal movement, and ignoring any obstacles that may be in the way.

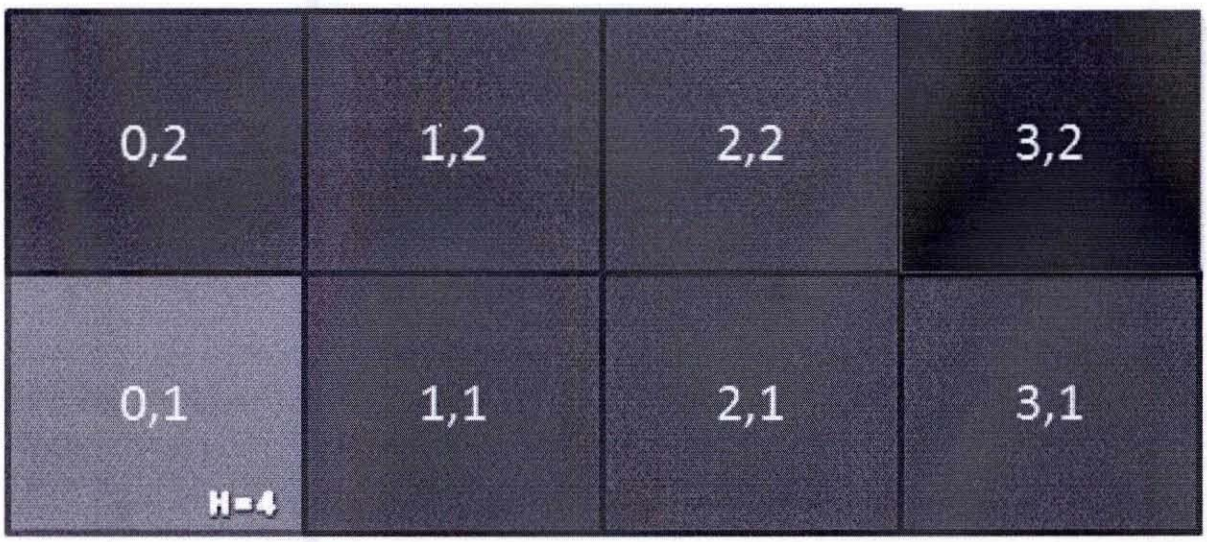


Fig – 9: Shortest Path Calculation (Initial State)

Now, check from the priority queue which element has the lowest 'F' cost, then remove it from the queue and add it to the closed list while again finding out the adjacent cells and adding them to the priority queue. Now, in this case, we will have to check a few things before adding the cell/cells to the priority queue. They are:

- Check if the cell is reachable from the parent cell or not.
- If reachable then check if the cell is already in the priority queue.
- If not in the priority queue and also not present in the closed list, then add it to the queue while calculating all the necessary fields shown above.
- In case the cell already exists in the queue, we will have to check if the 'G' cost to go from the current location to that cell is smaller than it's previous 'G' cost in perspective to its previous parent.
    - If the cost is bigger, then no change is needed.
    - In case the cost is smaller, then we will change the cell's parent to be the current cell and recalculate the 'G' cost while recalculating the direction.
- The Algorithm stops searching for further paths when it gets to the target cell and adds it to the closed list or the priority queue is empty and no target cell found. In case it fails to find a path, then it will return 'false' while providing no closed list.
- Now, that we have the target cell in the closed list, we back track from that cell to find its parent node and keep on back tracking till we can hit the initial starting cell.
- This will give us the **shortest path** from the starting position to the target cell.

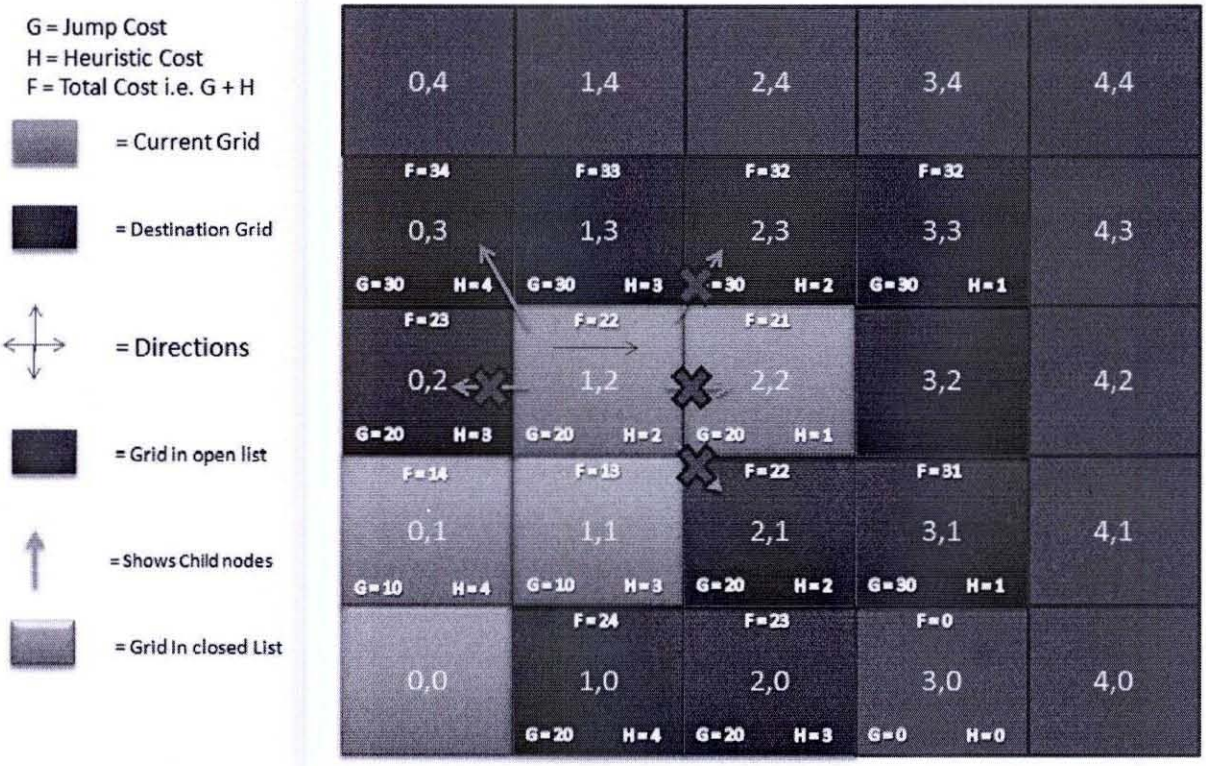


Fig – 10: Shortest Path Calculation

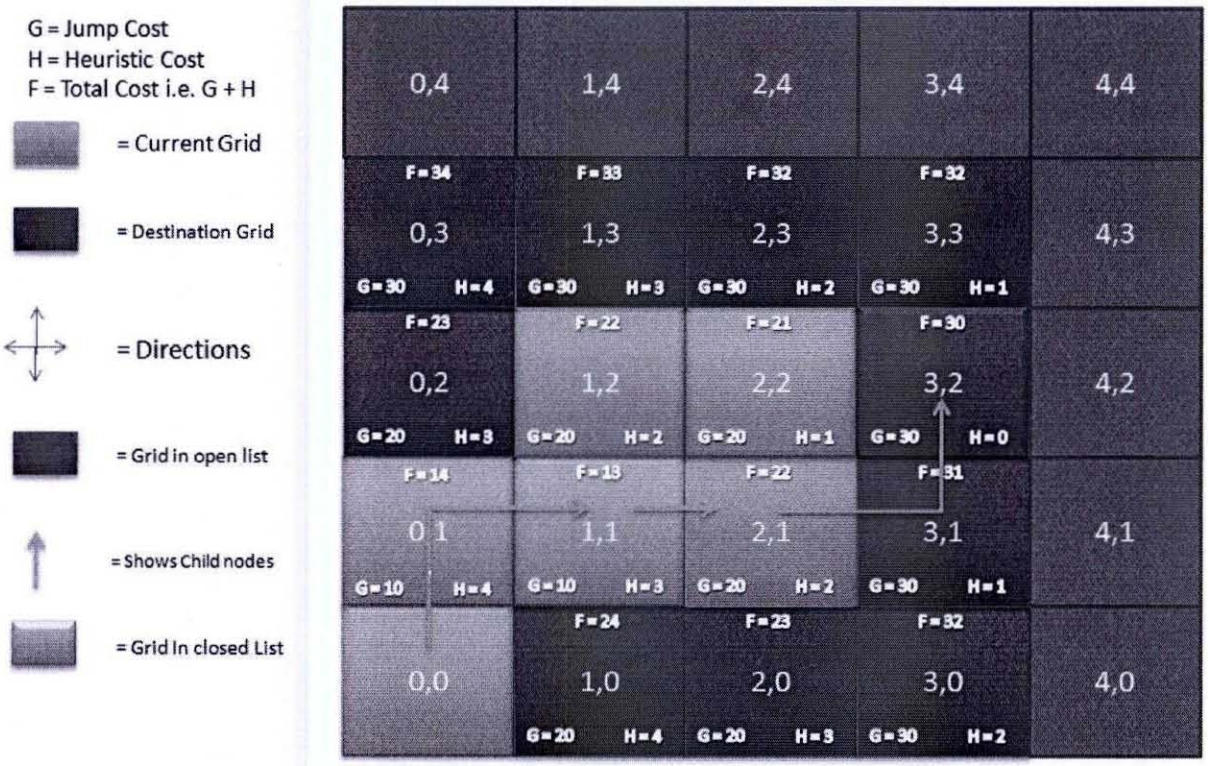


Fig – 11: Final Result of the Shortest Path Calculation

LIMITATION OF OUR CARTOGRAPHER

There might be obstacles that are not square shaped in our target environment. For example we may have cylindrical or multi cornered objects placed in the environment. Again the environment the vehicle is placed in can be of a different shape. In such cases, our car will not efficiently plan out the area rather it may miss out some spots of the environment to be treated as obstacles.

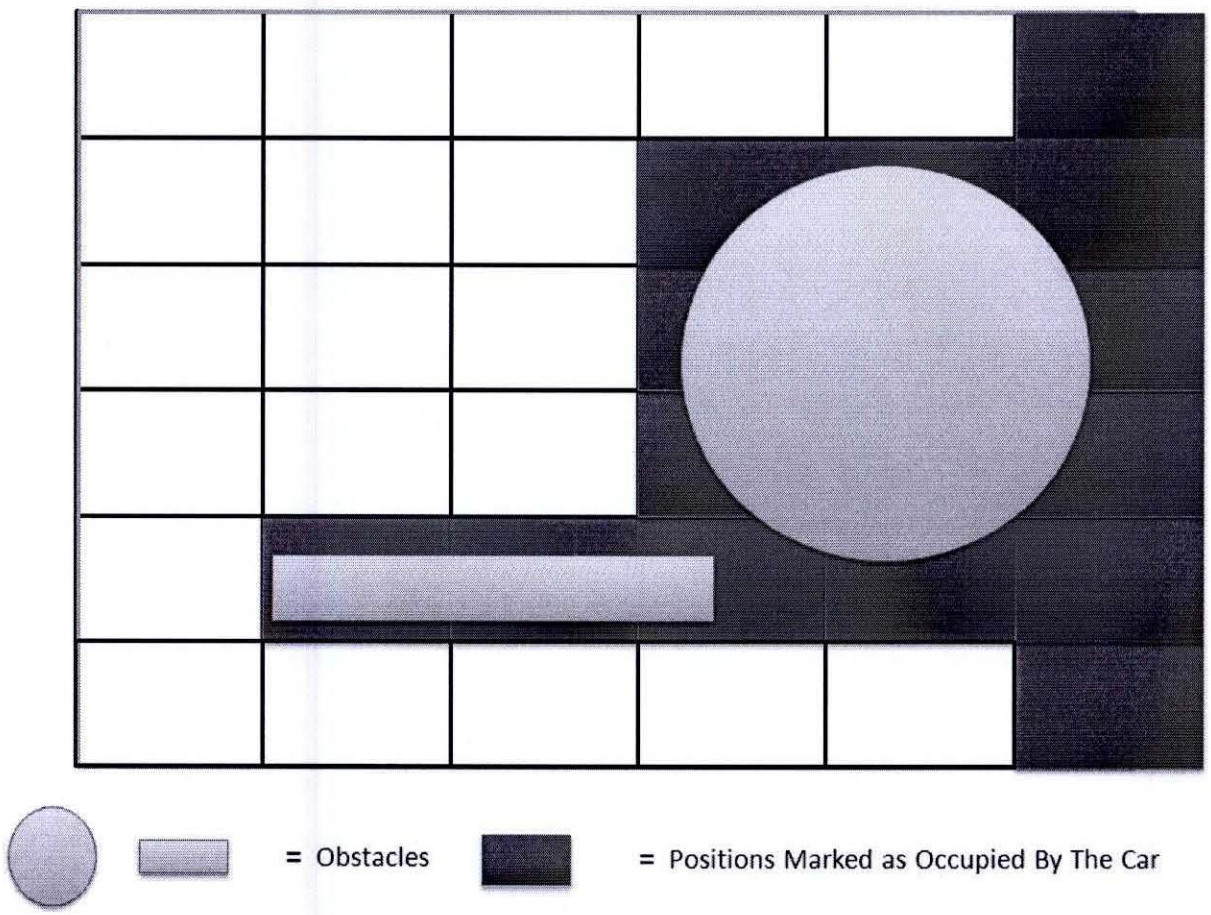


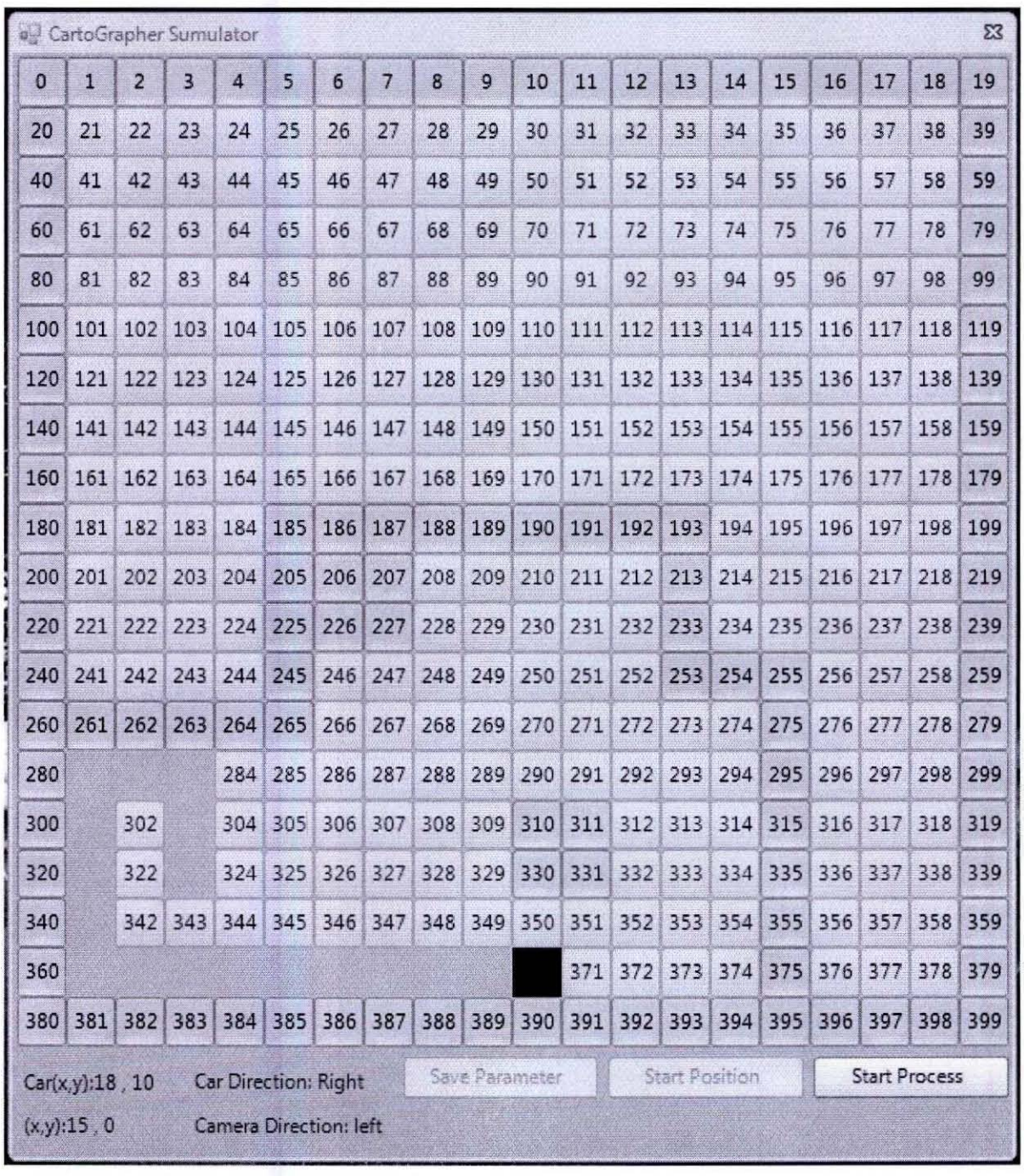
Fig – 12: Limitations Faced while Mapping Using Gridding Technique

SOFTWARE UI


CartoGrapher
CartoGrapher
Captured Image
Control
Start Camara
Stop Camara
Capture Time: 20
Com Port Name:
Baud Rate:
Parity:
Stop Bits:
Set Com Port
Start Process
Start Lisining to Sumulator
Frame Rate: 0
Description
Log
Message received from client...(done) Message sent to client...(camera,left) Message received from client...(2)
Message sent to client...(Get Obstacle) Message received from client...(done) Message sent to client...(camera,up)
Message received from client...(done) Message sent to client...(move,Front) Message received from client...(1)
Message sent to client...(Get Obstacle) Message received from client...(done) Message sent to client...(camera,right)
Message received from client...(2) Message sent to client...(Get Obstacle) Message received from client...(done)
Message sent to client...(camera,left) Message received from client...(2) Message sent to client...(Get Obstacle)
Message received from client...(done) Message sent to client...(camera,up) Message received from client...(done)
Message sent to client...(move,Front) Message received from client...(1) Message sent to client...(Get Obstacle)
Message received from client...(done) Message sent to client...(camera,right) Message received from client...(2)
Message sent to client...(Get Obstacle) Message received from client...(done) Message sent to client...(camera,left)
Description
Description

SOFTWARE SIMULATOR

As we were having trouble getting the hardware implemented according to our specifications, we developed a simulator that simulated possible real world cases with obstacles placed and interacted with our main software to simulate how our main software would react to a real world scenario. This software and the main software and uses a TCP connection to connection to each other and uses simple instructions to communicate with each other.



TECHNOLOGIES USED

- Visual Studio 2010
    - Used C# for coding the main interface and the simulator
- Netbeans IDE 6.9
    - For testing components of the software separately while under development.
- Utilized **Threading** for avoiding the hang ups of the GUI component & also the interactions between the main software and the simulator.
- Used **Com Port API** for handling the interaction between the microcontroller and the main software.
- Utilized **TCP Connection API** for handling the interaction between the server (core software) and the client (simulator)
- Used a C# based generic class collection called **C5**.
- Used '**EMGUCV**' which is a wrapper class used to wrap the image processing library '**OPEN SOURCE COMPUTER VISION**' also known as '**OPENCV**' that we used for handling image processing of our project

CONCLUSION

Successes and Failures

From the very beginning, our primary goal was to implement an autonomous vehicle that will be able to move around and map an enclosed area. We successfully implemented the software side that can plan an area using occupancy grid mapping technique. Due to hardware limitations we had to make changes to our grid mapping technique i.e. instead taking a probabilistic value if a cell is occupied or not, we are taking a direct integer value that defines the cell is occupied or empty. Again, we faced problems while getting the hardware part to work properly due to hardware setup failures. Again, we successfully implemented shortest path algorithm that helped us to get the shortest path to an unvisited cell & also successfully embedded a way to get the final state when the car will know that it has mapped the entire reachable area. To overcome the failure in successfully implementing the hardware in planning a specific area, we implemented a simulator with which our main software interacts and plans a virtual area to show that it is working perfectly.

Future Work

We further plan on expanding our work according to some sequence. First of all, the current method we used for detecting obstacles has a major chance of being interfered by noise. Which we plan on to overcome by using a more systematic way using more complex algorithms like Hough Transform. For area mapping we plan on improving our current technique by adding additional sensors and taking overall average from all the sensors for getting a probabilistic value of an obstacle being present or not. Currently our software works for an enclosed area with a static environment. We further plan on developing this software to handle real world scenarios where dynamic objects are present.

REFERENCES

[1] Sebastian Thrun, An online Mapping Algorithm for Teams of Mobile Robots.

[2] Feng Lu and EvangelosMilios, Robot Pose Estimation in Unknown Environments by Matching 2D Range Scans

[3] TjioHokHoo&Mohd Rizal Arshad, A Simple Surface Mapping Technique using Laser Triangulation Method

[4] Klancnik, S.;Balic, J. &Planinsic, P. ; Obstacle Detection With Active Laser Triangulation

[5] Stefano Carpin, HolgerKenn, Andreas Birk, Autonomous Mapping in the Real Robots Rescue League

[6] Sebastian Thrun, Robotic Mapping: A Survey

[7] M. C. SACCHEITIN, J. J. LOPES, D.F. WOLF, J. L. SILVA and E. MARQUES, ANALYSIS AND IMPLEMENTATION OF LOCALIZATION AND MAPPING ALGORITHMS FOR MOBILE ROBOTS BASED ON RECONFIGURABLE COMPUTING

[8] http://en.wikipedia.org/wiki/Occupancy_grid_mapping

[9] http://en.wikipedia.org/wiki/Kalman_filter

[10] http://en.wikipedia.org/wiki/Baysian_filter

[11] http://en.wikipedia.org/wiki/Expectation-maximization_algorithm

[12] http://en.wikipedia.org/wiki/A*_search_algorithm

[13] http://www.policyalmanac.org/games/aStarTutorial.htm

[14] http://www.emgu.com/wiki/index.php/Main_Page

[15] http://www.jmst.org.tw/marine/5/95-99.pdf

[16] http://www.dreamincode.net/forums/

Appendix:

AI.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using C5;
//using System.Collections.Generic;
namespace CartoGrapher.Class
{
publicclassAI
    {

publicMap map;
publicbool moveBackOnce = false;
public AI()
        {
Global.GridSize = 1;//initialize a size of Grid
// When the A.I is Initialized it will assume that the car is at location (0,0)
//and is facing Up Parallel to the border and is Visited
map = newMap();


        }
publicvoid StartProcess()//Start
        {
//Class.Global.LogMessage = "A.I. process started";

while (true)
            {
ExploreArea();
if (map.CurrentGrid.TopGrid.GridLocationStatus != Global.locationStatus.Empty &&
map.CurrentGrid.LeftGrid.GridLocationStatus != Global.locationStatus.Empty &&
map.CurrentGrid.RightGrid.GridLocationStatus != Global.locationStatus.Empty &&
map.CurrentGrid.BottomGrid.GridLocationStatus != Global.locationStatus.Empty
                    )
                {
if (!map.MoveTo(Global.moveList.LeftFront, Global.locationStatus.Empty, false) &&
                    !map.MoveTo(Global.moveList.RightFront, Global.locationStatus.Empty,
false))
                    {
                        {
Grid grid = map.FindNearestGrid(map.CurrentGrid, Global.locationStatus.Empty);

if (grid != null)
                            {
grid = map.FindNearestGrid(grid, Global.locationStatus.Visited);
MoveToGrid(grid);
```

```
                            }
else
                            {
Global.tcpserver.TCPWriter("Complete");
Global.searchDone = true;
return;

                            }
                        }
                    }
else
                    {
if(!MoveTo(Global.moveList.LeftFront, Global.locationStatus.Empty))
if (!MoveTo(Global.moveList.RightFront, Global.locationStatus.Empty))
                            {
                            }
                    }
                }
else
                {
if (!MoveTo(Global.moveList.Front, Global.locationStatus.Empty))
if (!MoveTo(Global.moveList.LeftFront, Global.locationStatus.Empty))
if (!MoveTo(Global.moveList.RightFront, Global.locationStatus.Empty))
if (!MoveTo(Global.moveList.Back, Global.locationStatus.Empty))
if (!MoveTo(Global.moveList.Back, Global.locationStatus.Visited))
                                    {

                                    }
                }
            }
        }

publicvoid MoveToGrid(Grid toGrid)// Move to a specific grid using the shortest path
        {
List<Global.moveList> shortestPathList = ShortestPastToGrid(toGrid);
for (int i = 1; i < shortestPathList.Count;i++ )
            {
MoveTo(shortestPathList[i], Global.locationStatus.Visited);
            }

        }

privateList<Global.moveList> ShortestPastToGrid(Grid toGrid)//find the shortest path
between current
//and given grid....
        {
            Class.Search.Solver aStarSearch = newSearch.Solver(map, toGrid);

List<Global.moveList> shortestPathList = newList<Global.moveList>();
shortestPathList = aStarSearch.SolvePuzzel();
return shortestPathList;

        }

publicbool MoveTo(Global.moveList direction, Global.locationStatus status)// move to a
specified direction
        {
switch (direction)
```

```
            {
case (Global.moveList.Front):

if (map.MoveTo(Global.moveList.Front, status, true))//if front grid is empty move up
                    {
Control.Move(Global.moveList.Front, Global.GridSize);
returntrue;
                    }
break;
case (Global.moveList.Back):
if (map.MoveTo(Global.moveList.Back, status, true))//if back grid is empty move down
                    {
Control.Move(Global.moveList.Back, Global.GridSize);
returntrue;
                    }
break;
case (Global.moveList.LeftFront):

if (map.MoveTo(Global.moveList.LeftFront, status, true))//if front grid is empty move
LeftFront
                    {
Control.Move(Global.moveList.LeftFront, Global.GridSize);
returntrue;
                    }
break;
case (Global.moveList.RightFront):

if (map.MoveTo(Global.moveList.RightFront, status, true))//if front grid is empty move
RightFront
                    {
Control.Move(Global.moveList.RightFront, Global.GridSize);
returntrue;
                    }
break;
case (Global.moveList.LeftBack):

if (map.MoveTo(Global.moveList.LeftBack, status, true))//if front grid is empty move
LeftBack
                    {
Control.Move(Global.moveList.LeftBack, Global.GridSize);
returntrue;
                    }
break;
case (Global.moveList.RightBack):

if (map.MoveTo(Global.moveList.RightBack, status, true))//if front grid is empty move
RightBack
                    {
Control.Move(Global.moveList.RightBack, Global.GridSize);
returntrue;
                    }
break;
default:
returnfalse;

            }

returnfalse;
```

```
        }

privatevoid ExploreArea()//Explore the Four Sides of the Area
        {
//Global.LogMessage = "Exploring Area....";

Grid grid = Explore(Global.Direction.Up);
map.AddGridByDirection(Global.Direction.Up, grid);


grid = Explore(Global.Direction.Left);
map.AddGridByDirection(Global.Direction.Left, grid);


grid = Explore(Global.Direction.Right);
map.AddGridByDirection(Global.Direction.Right, grid);


grid = Explore(Global.Direction.Down);
map.AddGridByDirection(Global.Direction.Down, grid);



        }
privateGrid Explore(Global.Direction direction)//Explore given Side Of current grid
        {

Grid newGrid;

newGrid = map.IsGridExplored(map.CurrentGrid.GetGridX(direction,
map.CurrentDirection),//Check is the Grid is Explored Before
map.CurrentGrid.GetGridY(direction, map.CurrentDirection));

if (newGrid != null)
            {
//Global.LogMessage = "Grid " + map.CurrentGrid.GetGridX(direction, map.CurrentDirection)
+ "," + map.CurrentGrid.GetGridY(direction, map.CurrentDirection) + "already
explored...";
return newGrid;

            }
newGrid = newGrid(map.CurrentGrid.GetGridX(direction, map.CurrentDirection), //Get the
grid location of the Relative to direction of the direction of the car
map.CurrentGrid.GetGridY(direction, map.CurrentDirection));

Control.RotateCamara(direction); //Rotate Camera
//Get Location Status
            newGrid.CheckedFrom = map.CurrentGrid;
            newGrid.GridLocationStatus =
Global.CheckForGridOccupance(Control.GetObstacleDistance());
map.AddGridToMap(newGrid);// add the grid to the map....
            map.CurrentGrid.CheckedFrom = map.CurrentGrid;

return newGrid;
        }
```

```
    }
}
```

Control.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace CartoGrapher.Class
{
publicstaticclassControl
    {
```

```
privatestaticint camaraAngle=0;
privatestaticThread thread;
privatestaticGlobal.Direction cameraDirection;
privatestaticbool flag = false;
publicstaticGlobal.Direction CameraDirection
        {
get { returnControl.cameraDirection; }
set { Control.cameraDirection = value; }
        }
publicstaticvoid initiateCommunication()
        {
Global.tcpserver.MessageRecieved += newSystem.EventHandler(MessageReceived);
thread = newThread(newThreadStart(Global.tcpserver.StartListener));
thread.Start();
//Global.LogMessage = "Server Started";

        }
publicstaticvoid MessageReceived(object obj, EventArgs e)
        {
if (Global.tcpserver.RecievedFromClient == "done")
            {
flag = true;
            }
        }
publicstaticvoid RotateCamara(Global.Direction move)
        {
int moveAngle = 0;
if (move == Global.Direction.Up)
            {
Global.tcpserver.TCPWriter("camera,up");
Global.tcpserver.TCPReader();
if(camaraAngle!=0)
                {
if(camaraAngle >= 180)
                    {
moveAngle = 360 - camaraAngle;
                    }
else
                    {
moveAngle = 0-camaraAngle;
                    }
                }
camaraAngle = 0;
            }
elseif (move == Global.Direction.Down)
            {
Global.tcpserver.TCPWriter("camera,down");
Global.tcpserver.TCPReader();

if (camaraAngle != 180)
                {
if (camaraAngle > 180)
                    {
moveAngle =  180-camaraAngle;
                    }
else
                    {
```

```
moveAngle = camaraAngle-180;
                    }
                }
camaraAngle = 180;
            }
elseif (move == Global.Direction.Left)
            {
Global.tcpserver.TCPWriter("camera,left");
Global.tcpserver.TCPReader();
if (camaraAngle != 270)
                {
if (camaraAngle < 90)
                    {
moveAngle = -90;
                    }
else
                    {
moveAngle = 270-camaraAngle;
                    }
                }
camaraAngle = 270;
            }
elseif (move == Global.Direction.Right)
            {
Global.tcpserver.TCPWriter("camera,right");
Global.tcpserver.TCPReader();
if (camaraAngle != 90)
                {
if (camaraAngle == 270)
                    {
moveAngle = 180;
                    }
elseif (camaraAngle == 0)
                    {
moveAngle = 90;
                    }
else
                    {
moveAngle = -90;
                    }

                }
camaraAngle = 90;
            }
            CameraDirection = move;
        }
publicstaticdouble GetObstacleDistance()
        {
Global.tcpserver.TCPWriter("Get Obstacle");
Global.tcpserver.TCPReader();
returndouble.Parse(Global.tcpserver.RecievedFromClient);

        }
publicstaticdouble GetMoveDistance()
        {
//
//string a = "asdas";
return 0;
```

```
//this function not needed
        }
publicstaticvoid Move(Global.moveList move, double Distace)
        {

switch(move){
caseGlobal.moveList.Front:
Global.tcpserver.TCPWriter("move,Front");
Global.tcpserver.TCPReader();
break;
caseGlobal.moveList.Back:
Global.tcpserver.TCPWriter("move,Back");
Global.tcpserver.TCPReader();
break;
caseGlobal.moveList.LeftFront:
Global.tcpserver.TCPWriter("move,LeftFront");
Global.tcpserver.TCPReader();
break;
caseGlobal.moveList.RightFront:
Global.tcpserver.TCPWriter("move,RightFront");
Global.tcpserver.TCPReader();
break;
caseGlobal.moveList.LeftBack:
Global.tcpserver.TCPWriter("move,LeftBack");
Global.tcpserver.TCPReader();
break;
caseGlobal.moveList.RightBack:
Global.tcpserver.TCPWriter("move,RightBack");
Global.tcpserver.TCPReader();
break;
            }

        }

    }
}
```

Global.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Forms;
```

```
namespace CartoGrapher.Class
{
publicstaticclassGlobal
    {
publicstaticbool searchDone= false;
publicstaticTCPserver tcpserver = newTCPserver();

privatestaticTextBox txtbox;

publicstaticTextBox Txtbox
        {
get { returnGlobal.txtbox; }
set { Global.txtbox = value; }
        }
privatestaticstring logMessage;

publicstaticstring LogMessage
        {
get
            {
returnGlobal.logMessage;
            }
set
            {
Global.logMessage = value;
Log();
            }
        }
delegatevoidLogCallback();
privatestaticvoid Log()
        {


if (Txtbox.InvokeRequired)
            {
LogCallback d = newLogCallback(Log);
Txtbox.Invoke(d, newobject[] { });
            }
else
            {
                Txtbox.Text = logMessage + " \n " + Txtbox.Text;
            }
        }
publicenummoveList
        {
            Front = 1,
            Back = 2,
            LeftFront = 3,
            RightFront = 4,
            LeftBack = 5,
            RightBack = 6

        }

privatestaticdouble gridSize;

publicstaticdouble GridSize
        {
```

```
get { returnGlobal.gridSize; }
set { Global.gridSize = value; }
        }

publicenumDirection
        {
            Up = 0,
            Down = 1,
            Left = 2,
            Right = 3,
        }
publicenumlocationStatus
        {
            Empty = 0,
            Unknown = 1,
            Occupied = 2,
            Visited = 3
        }
publicstaticlocationStatus CheckForGridOccupance(double distance)
        {
if (distance >= gridSize * 2)
returnlocationStatus.Empty;
else
returnlocationStatus.Occupied;
        }


    }
}
```

Grid.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using C5;
namespace CartoGrapher.Class
{
publicclassGrid
    {
```

```
privateint locationX;

publicint LocationX
        {
get { return locationX; }
set { locationX = value; }
        }

privateint locationY;

publicint LocationY
        {
get { return locationY; }
set { locationY = value; }
        }


privateGlobal.locationStatus gridLocationStatus = Global.locationStatus.Occupied;

publicGlobal.locationStatus GridLocationStatus
        {
get { return gridLocationStatus; }
set { gridLocationStatus = value; }
        }

privateGrid checkedFrom;

publicGrid CheckedFrom
        {
get { return checkedFrom; }
set { checkedFrom = value; }
        }
privateGrid topGrid;

publicGrid TopGrid
        {
get { return topGrid; }
set { topGrid = value; }
        }
privateGrid bottomGrid;

publicGrid BottomGrid
        {
get { return bottomGrid; }
set { bottomGrid = value; }
        }

privateGrid leftGrid;

publicGrid LeftGrid
        {
get { return leftGrid; }
set { leftGrid = value; }
        }

privateGrid rightGrid;
```

```
publicGrid RightGrid
        {
get { return rightGrid; }
set { rightGrid = value; }
        }

public Grid(int x, int y)
        {
            LocationX = x;
            LocationY = y;
        }
publicint GetGridX(Global.Direction gridDirection)
        {
if (gridDirection== Global.Direction.Up)
return LocationX;
elseif(gridDirection== Global.Direction.Left)
return LocationX-1;
elseif (gridDirection == Global.Direction.Right)
return LocationX+1;
else
return LocationX;
        }
publicint GetGridY(Global.Direction gridDirection)
        {
if (gridDirection == Global.Direction.Up)
return LocationY+1;
elseif (gridDirection == Global.Direction.Left)
return LocationY - 1;
elseif (gridDirection == Global.Direction.Right)
return LocationY + 1;
else
return LocationY-1;
        }
publicint GetGridX(Global.Direction gridDirection, Global.Direction
directionOfOjbectRelativeFromGrid)
        {
if (directionOfOjbectRelativeFromGrid == Global.Direction.Up)
            {
if (gridDirection == Global.Direction.Up)
return LocationX;
elseif (gridDirection == Global.Direction.Left)
return LocationX - 1;
elseif (gridDirection == Global.Direction.Right)
return LocationX + 1;
else
return LocationX;
            }
elseif (directionOfOjbectRelativeFromGrid == Global.Direction.Left)
            {
if (gridDirection == Global.Direction.Up)
return LocationX-1;
elseif (gridDirection == Global.Direction.Left)
return LocationX;
elseif (gridDirection == Global.Direction.Right)
return LocationX;
else
return LocationX+1;
            }
```

```
elseif (directionOfOjbectRelativeFromGrid == Global.Direction.Right)
            {
if (gridDirection == Global.Direction.Up)
return LocationX + 1;
elseif (gridDirection == Global.Direction.Left)
return LocationX;
elseif (gridDirection == Global.Direction.Right)
return LocationX;
else
return LocationX - 1;
            }
else
            {
if (gridDirection == Global.Direction.Up)
return LocationX ;
elseif (gridDirection == Global.Direction.Left)
return LocationX+1;
elseif (gridDirection == Global.Direction.Right)
return LocationX-1;
else
return LocationX;
            }



        }
publicint GetGridY(Global.Direction gridDirection, Global.Direction
directionOfOjbectRelativeFromGrid)
        {

if (directionOfOjbectRelativeFromGrid == Global.Direction.Up)
            {
if (gridDirection == Global.Direction.Up)
return LocationY+1;
elseif (gridDirection == Global.Direction.Left)
return LocationY;
elseif (gridDirection == Global.Direction.Right)
return LocationY;
else
return LocationY-1;
            }
elseif (directionOfOjbectRelativeFromGrid == Global.Direction.Left)
            {
if (gridDirection == Global.Direction.Up)
return LocationY;
elseif (gridDirection == Global.Direction.Left)
return LocationY-1;
elseif (gridDirection == Global.Direction.Right)
return LocationY+1;
else
return LocationY;
            }
elseif (directionOfOjbectRelativeFromGrid == Global.Direction.Right)
            {
if (gridDirection == Global.Direction.Up)
return LocationY;
elseif (gridDirection == Global.Direction.Left)
return LocationY+1;
```

```
elseif (gridDirection == Global.Direction.Right)
return LocationY-1;
else
return LocationY;
            }
else
            {
if (gridDirection == Global.Direction.Up)
return LocationY-1;
elseif (gridDirection == Global.Direction.Left)
return LocationY ;
elseif (gridDirection == Global.Direction.Right)
return LocationY ;
else
return LocationY+1;
            }
        }

    }
}
```

Map.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace CartoGrapher.Class
{
publicclassMap
    {


privateList<Grid> gridList = newList<Grid>();
```

```
privateint count = 0;

publicint Count
        {
get { return gridList.Count; }

        }


privateGrid currentGrid = newGrid(0, 0);

publicGrid CurrentGrid
        {
get { return currentGrid; }
set { currentGrid = value; }
        }

privateGlobal.Direction currentDirection = Global.Direction.Up;

publicGlobal.Direction CurrentDirection
        {
get { return currentDirection; }
set { currentDirection = value; }
        }

publicGrid IsGridExplored(int x, int y)
        {

for (int i = 0; i <= gridList.Count - 1; i++)
            {
if (gridList[i].LocationX == x && gridList[i].LocationY == y)
                {
return gridList[i];
                }
            }
returnnull;
        }
publicGrid FindNearestGrid(Grid sourceGrid,Global.locationStatus status)//checks for
closest empty grid
        {
int x;
Grid tempGrid = null;
int distance = 1000000;
for (x = 0; x < gridList.Count; x++)
            {
if (gridList[x].GridLocationStatus == status)
                {
if (tempGrid == null)
                    {
tempGrid = gridList[x];
                    }
else
                    {
int cDistance = Math.Abs(gridList[x].LocationX - sourceGrid.LocationX) +
Math.Abs(gridList[x].LocationY - sourceGrid.LocationY);
if (cDistance < distance)
                        {
```

```
distance = cDistance;
tempGrid = gridList[x];
                        }
                    }
                }

            }
return tempGrid;
        }
publicvoid AddGridToMap(Grid grid)
        {
gridList.Add(grid);
        }
public Map()
        {
currentGrid = newGrid(0, 0);
            currentGrid.GridLocationStatus = Global.locationStatus.Visited;
gridList.Add(currentGrid);
        }
public Map(Map map)
        {
currentGrid = map.currentGrid;
            currentGrid.GridLocationStatus = currentGrid.GridLocationStatus;
currentDirection = map.currentDirection;
//gridList.Add(currentGrid);
gridList.Clear();
map.gridList.ForEach(delegate(Grid grid)
            {
gridList.Add(grid);
            });

        }
publicvoid Copy(Map map)
        {
currentGrid = map.currentGrid;
            currentGrid.GridLocationStatus = currentGrid.GridLocationStatus;
//gridList.Add(currentGrid);
currentDirection = map.currentDirection;
gridList.Clear();
map.gridList.ForEach(delegate(Grid grid)
            {
gridList.Add(grid);
            });
        }

publicvoid AddGridByDirection(Global.Direction direction, Grid grid)
        {
if (CurrentDirection == Global.Direction.Up)
            {
if (direction == Global.Direction.Up)
                    CurrentGrid.TopGrid=grid;
elseif (direction == Global.Direction.Down)
                    CurrentGrid.BottomGrid=grid;
elseif (direction == Global.Direction.Left)
                    CurrentGrid.LeftGrid= grid;
elseif (direction == Global.Direction.Right)
                    CurrentGrid.RightGrid=grid;
```

```
            }
elseif (CurrentDirection == Global.Direction.Down)
            {
if (direction == Global.Direction.Up)
                    CurrentGrid.BottomGrid = grid;
elseif (direction == Global.Direction.Down)
                    CurrentGrid.TopGrid = grid;
elseif (direction == Global.Direction.Left)
                    CurrentGrid.RightGrid= grid;
elseif (direction == Global.Direction.Right)
                    CurrentGrid.LeftGrid = grid;

            }
elseif (CurrentDirection == Global.Direction.Right)
            {
if (direction == Global.Direction.Up)
                    CurrentGrid.RightGrid = grid;
elseif (direction == Global.Direction.Down)
                    CurrentGrid.LeftGrid = grid;
elseif (direction == Global.Direction.Left)
                    CurrentGrid.TopGrid = grid;
elseif (direction == Global.Direction.Right)
                    CurrentGrid.BottomGrid = grid;

            }
else
            {
if (direction == Global.Direction.Up)
                    CurrentGrid.LeftGrid = grid;
elseif (direction == Global.Direction.Down)
                    CurrentGrid.RightGrid = grid;
elseif (direction == Global.Direction.Left)
                    CurrentGrid.BottomGrid = grid;
elseif (direction == Global.Direction.Right)
                    CurrentGrid.TopGrid = grid;
            }
        }

publicGrid GetGridByDirection(Global.moveList direction)
        {
if (CurrentDirection == Global.Direction.Up)
            {
if (direction == Global.moveList.Front)
return CurrentGrid.TopGrid;

elseif (direction == Global.moveList.Back)
return CurrentGrid.BottomGrid;

elseif (direction == Global.moveList.LeftFront)
if (CurrentGrid.LeftGrid.TopGrid != null)
return CurrentGrid.LeftGrid.TopGrid;
else
return CurrentGrid.TopGrid.LeftGrid;

elseif (direction == Global.moveList.RightFront)
if (CurrentGrid.TopGrid.RightGrid != null)
return CurrentGrid.TopGrid.RightGrid;
else
```

```
return CurrentGrid.RightGrid.TopGrid;

elseif (direction == Global.moveList.LeftBack)
if (CurrentGrid.BottomGrid.LeftGrid != null)
return CurrentGrid.BottomGrid.LeftGrid;
else
return CurrentGrid.LeftGrid.BottomGrid;

elseif (direction == Global.moveList.RightBack)
if (CurrentGrid.BottomGrid.RightGrid != null)
return CurrentGrid.BottomGrid.RightGrid;
else
return CurrentGrid.RightGrid.BottomGrid;

            }
elseif (CurrentDirection == Global.Direction.Down)
            {
if (direction == Global.moveList.Front)
return CurrentGrid.BottomGrid;
elseif (direction == Global.moveList.Back)
return CurrentGrid.TopGrid;
elseif (direction == Global.moveList.LeftFront)
if (CurrentGrid.BottomGrid.RightGrid != null)
return CurrentGrid.BottomGrid.RightGrid;
else
return CurrentGrid.RightGrid.BottomGrid;

elseif (direction == Global.moveList.RightFront)
if (CurrentGrid.BottomGrid.LeftGrid != null)
return CurrentGrid.BottomGrid.LeftGrid;
else
return CurrentGrid.LeftGrid.BottomGrid;

elseif (direction == Global.moveList.LeftBack)
if (CurrentGrid.TopGrid.RightGrid != null)
return CurrentGrid.TopGrid.RightGrid;
else
return CurrentGrid.RightGrid.TopGrid;

elseif (direction == Global.moveList.RightBack)
if (CurrentGrid.TopGrid.LeftGrid != null)
return CurrentGrid.TopGrid.LeftGrid;
else
return CurrentGrid.LeftGrid.TopGrid;

            }
elseif (CurrentDirection == Global.Direction.Left)
            {
if (direction == Global.moveList.Front)
return CurrentGrid.LeftGrid;

elseif (direction == Global.moveList.Back)
return CurrentGrid.RightGrid;

elseif (direction == Global.moveList.LeftFront)
if (CurrentGrid.BottomGrid.LeftGrid != null)
return CurrentGrid.BottomGrid.LeftGrid;
else
```

```
return CurrentGrid.LeftGrid.BottomGrid;

elseif (direction == Global.moveList.RightFront)
if (CurrentGrid.LeftGrid.TopGrid != null)
return CurrentGrid.LeftGrid.TopGrid;
else
return CurrentGrid.TopGrid.LeftGrid;

elseif (direction == Global.moveList.LeftBack)
if (CurrentGrid.BottomGrid.RightGrid!= null)
return CurrentGrid.BottomGrid.RightGrid;
else
return CurrentGrid.RightGrid.BottomGrid;

elseif (direction == Global.moveList.RightBack)
if (CurrentGrid.TopGrid.RightGrid != null)
return CurrentGrid.TopGrid.RightGrid;
else
return CurrentGrid.RightGrid.TopGrid;
            }
else
            {
if (direction == Global.moveList.Front)
return CurrentGrid.RightGrid;
elseif (direction == Global.moveList.Back)
return CurrentGrid.LeftGrid;
elseif (direction == Global.moveList.LeftFront)
if(CurrentGrid.TopGrid.RightGrid!=null)
return CurrentGrid.TopGrid.RightGrid;
else
return CurrentGrid.RightGrid.TopGrid;

elseif (direction == Global.moveList.RightFront)
if (CurrentGrid.BottomGrid.RightGrid != null)
return CurrentGrid.BottomGrid.RightGrid;
else
return CurrentGrid.RightGrid.BottomGrid;

elseif (direction == Global.moveList.LeftBack)
if (CurrentGrid.TopGrid.LeftGrid != null)
return CurrentGrid.TopGrid.LeftGrid;
else
return CurrentGrid.LeftGrid.TopGrid;

elseif (direction == Global.moveList.RightBack)
if (CurrentGrid.BottomGrid.LeftGrid != null)
return CurrentGrid.BottomGrid.LeftGrid;
else
return CurrentGrid.LeftGrid.BottomGrid;

            }
returnnull;
        }

//Move car to a certain direction on the map
publicbool MoveTo(Global.moveList move, Global.locationStatus status,bool doMove)
        {
switch (move)
```

```
            {
caseGlobal.moveList.Front:
caseGlobal.moveList.Back:
if (GetGridByDirection(move).GridLocationStatus == status)
                    {
if (doMove == true)
                        {
                            CurrentGrid = GetGridByDirection(move);
ChangeDirection(move);
                            CurrentGrid.GridLocationStatus =
Global.locationStatus.Visited;
                        }
returntrue;
                    }
break;
caseGlobal.moveList.LeftFront:
caseGlobal.moveList.RightFront:
if (GetGridByDirection(Global.moveList.Front).GridLocationStatus ==
Global.locationStatus.Visited)
if (GetGridByDirection(move).GridLocationStatus == status)
                        {
if (doMove == true)
                            {
                                CurrentGrid = GetGridByDirection(move);
ChangeDirection(move);
                                CurrentGrid.GridLocationStatus =
Global.locationStatus.Visited;
                            }
returntrue;

                        }
break;
caseGlobal.moveList.LeftBack:
caseGlobal.moveList.RightBack:
if (GetGridByDirection(Global.moveList.Back).GridLocationStatus ==
Global.locationStatus.Visited)
if (GetGridByDirection(move).GridLocationStatus == status)
                        {
if (doMove == true)
                            {
                                CurrentGrid = GetGridByDirection(move);
ChangeDirection(move);
                                CurrentGrid.GridLocationStatus =
Global.locationStatus.Visited;
                            }
returntrue;
                        }
break;

            }


returnfalse;
        }


//Change of Direction of the car relative to the grid
privatevoid ChangeDirection(Global.moveList moveDirection)
```

```
        {
if (moveDirection == Global.moveList.LeftBack || moveDirection ==
Global.moveList.RightFront)
            {
if (currentDirection == Global.Direction.Up)
currentDirection = Global.Direction.Right;
elseif (currentDirection == Global.Direction.Down)
currentDirection = Global.Direction.Left;
elseif (currentDirection == Global.Direction.Right)
currentDirection = Global.Direction.Down;
else
currentDirection = Global.Direction.Up;
            }
elseif (moveDirection == Global.moveList.RightBack || moveDirection ==
Global.moveList.LeftFront)
            {
if (currentDirection == Global.Direction.Up)
currentDirection = Global.Direction.Left;
elseif (currentDirection == Global.Direction.Down)
currentDirection = Global.Direction.Right;
elseif (currentDirection == Global.Direction.Right)
currentDirection = Global.Direction.Up;
else
currentDirection = Global.Direction.Down;
            }

        }
    }
}
```

TCPserver.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net.Sockets;
using System.Net;
using System.Net.Mime;
using System.Net.Mail;
using System.IO;
namespace CartoGrapher.Class
{
publicclassTCPserver
    {
publicEventHandler MessageRecieved;
```

```
privatestring recievedFromClient = null;

publicstring RecievedFromClient
        {
get { return recievedFromClient; }
set
            {
recievedFromClient = value;
this.MessageRecieved(this, newEventArgs());
            }
        }
privateTcpListener listener = null;

publicTcpListener Listener
        {
get { return listener; }
set { listener = value; }
        }
privateTcpClient client = null;

publicTcpClient Client
        {
get { return client; }
set { client = value; }
        }
privateNetworkStream stream = null;

publicNetworkStream Stream
        {
get { return stream; }
set { stream = value; }
        }
privateBinaryWriter writer = null;

publicBinaryWriter Writer
        {
get { return writer; }
set { writer = value; }
        }
privateBinaryReader reader = null;

publicBinaryReader Reader
        {
get { return reader; }
set { reader = value; }
        }
public TCPserver()
        {

listener = newTcpListener(newIPAddress(newbyte[] { 127, 0, 0, 1 }), 1200);
listener.Start();
        }
publicvoid TCPReader()
        {
reader = newBinaryReader(stream);
            RecievedFromClient = reader.ReadString();
```

```
Global.LogMessage = "Message received from client...(" + RecievedFromClient + ")";
stream.Flush();
        }
publicvoid StartListener()
        {
while (true)
            {
if (client == null)
                {
client = listener.AcceptTcpClient();

stream = client.GetStream();
return;
                }
            }
        }
publicvoid TCPWriter(string message)
        {
if (client != null)
            {
if(client.Connected==true)
                {
writer = newBinaryWriter(stream);
writer.Write(message);
Global.LogMessage = "Message sent to client...(" + message + ")";

                }
            }
        }
    }


}
```

ComPortInterface.cs

```
using System;
using System.Text;
using System.Drawing;
using System.IO.Ports;
using System.Windows.Forms;
//Author::PsychoCoder
//Source::http://www.dreamincode.net/forums/topic/35775-serial-port-communication-in-
c%23/
//

namespace CartoGrapher.Class
{
publicclassComPortInterface
    {

publicenumTransmissionType {
```

```
            Text,
            Hex
        }
publicenumMessageType
        {
            Incoming,
            Outgoing,
            Normal,
            Warning,
            Error
        }
privatestring baudRate = string.Empty;
privatestring parity = string.Empty;
privatestring stopBits = string.Empty;
privatestring dataBits = string.Empty;
privatestring portName = string.Empty;
privateTransmissionType transType;

//global manager variables
privateSerialPort comPort = newSerialPort();
privateRichTextBox _displaywindow;
privateColor[] MessageColor = { Color.Blue, Color.Green, Color.Black, Color.Orange,
Color.Red };
publicstring BaudRate
        {
get { return baudRate; }
set { baudRate = value; }
        }

publicstring Parity
        {
get { return parity; }
set { parity = value; }
        }


publicstring StopBits
        {
get { return stopBits; }
set { stopBits = value; }
        }


publicstring DataBits
        {
get { return dataBits; }
set { dataBits = value; }
        }


publicstring PortName
        {
get { return portName; }
set { portName = value; }
        }

publicTransmissionType CurrentTransmissionType
        {
```

```
get { return transType; }
set { transType = value; }
        }
publicRichTextBox DisplayWindow
        {
get { return _displaywindow; }
set { _displaywindow = value; }
        }

public ComPortInterface(string baud, string par, string sBits, string dBits, string name)
        {
       baudRate = baud;
       parity = par;
       stopBits = sBits;
       dataBits = dBits;
       portName = name;
               comPort.DataReceived +=
newSerialDataReceivedEventHandler(comPort_DataReceived);
        }
public ComPortInterface()
        {
       baudRate = string.Empty;
       parity = string.Empty;
       stopBits = string.Empty;
       dataBits = string.Empty;
       portName = "COM1";

comPort.DataReceived+=newSerialDataReceivedEventHandler(comPort_DataReceived);
        }
publicvoid WriteData(string msg)
        {

switch (CurrentTransmissionType)
           {
caseTransmissionType.Text:
//first make sure the port is open
//if its not open then open it
if (!(comPort.IsOpen == true)) comPort.Open();
//send the message to the port
comPort.Write(msg);
//display the message
DisplayData(MessageType.Outgoing, msg + "\n");
break;
caseTransmissionType.Hex:
try
                    {
//convert the message to byte array
byte[] newMsg = HexToByte(msg);
//send the message to the port
comPort.Write(newMsg, 0, newMsg.Length);
//convert back to hex and display
DisplayData(MessageType.Outgoing, ByteToHex(newMsg) + "\n");
                    }
catch (FormatException ex)
                    {
//display error message
DisplayData(MessageType.Error, ex.Message);
                    }
```

```
finally
                    {
                        _displaywindow.SelectAll();
                    }
break;
default:
//first make sure the port is open
//if its not open then open it
if (!(comPort.IsOpen == true)) comPort.Open();
//send the message to the port
comPort.Write(msg);
//display the message
DisplayData(MessageType.Outgoing, msg + "\n");
break;
break;
            }
        }
privatebyte[] HexToByte(string msg)
        {
//remove any spaces from the string
msg = msg.Replace(" ", "");
//create a byte array the length of the
//string divided by 2
byte[] comBuffer = newbyte[msg.Length / 2];
//loop through the length of the provided string
for (int i = 0; i < msg.Length; i += 2)
//convert each set of 2 characters to a byte
//and add to the array
comBuffer[i / 2] = (byte)Convert.ToByte(msg.Substring(i, 2), 16);
//return the array
return comBuffer;
        }
privatestring ByteToHex(byte[] comByte)
        {
//create a new StringBuilder object
StringBuilder builder = newStringBuilder(comByte.Length * 3);
//loop through each byte in the array
foreach (byte data in comByte)
//convert the byte to a string and add to the stringbuilder
builder.Append(Convert.ToString(data, 16).PadLeft(2, '0').PadRight(3, ' '));
//return the converted value
return builder.ToString().ToUpper();
        }
        [STAThread]
privatevoid DisplayData(MessageType type, string msg)
        {
            _displaywindow.Invoke(newEventHandler(delegate
            {
                _displaywindow.SelectedText = string.Empty;
                _displaywindow.SelectionFont = newFont(_displaywindow.SelectionFont,
FontStyle.Bold);
                _displaywindow.SelectionColor = MessageColor[(int)type];
                _displaywindow.AppendText(msg);
                _displaywindow.ScrollToCaret();
            }));
        }
publicbool OpenPort()
        {
```

```
try
            {
//first check if the port is already open
//if its open then close it
if (comPort.IsOpen == true) comPort.Close();

//set the properties of our SerialPort Object
                comPort.BaudRate = int.Parse(baudRate);//BaudRate
                comPort.DataBits = int.Parse(dataBits);//DataBits
                comPort.StopBits = (StopBits)Enum.Parse(typeof(StopBits), stopBits);
       //StopBits
                comPort.Parity = (Parity)Enum.Parse(typeof(Parity), parity);
       //Parity
                comPort.PortName = portName;   //PortName
//now open the port
comPort.Open();
//display message
DisplayData(MessageType.Normal, "Port opened at " + DateTime.Now + "\n");
//return true
returntrue;
            }
catch (Exception ex)
            {
DisplayData(MessageType.Error, ex.Message);
returnfalse;
            }
        }
void comPort_DataReceived(object sender, SerialDataReceivedEventArgs e)
        {
//determine the mode the user selected (binary/string)
switch (CurrentTransmissionType)
            {
//user chose string
caseTransmissionType.Text:
//read data waiting in the buffer
string msg = comPort.ReadExisting();
//display the data to the user
DisplayData(MessageType.Incoming, msg + "\n");
break;
//user chose binary
caseTransmissionType.Hex:
//retrieve number of bytes in the buffer
int bytes = comPort.BytesToRead;
//create a byte array to hold the awaiting data
byte[] comBuffer = newbyte[bytes];
//read the data and store it
comPort.Read(comBuffer, 0, bytes);
//display the data to the user
DisplayData(MessageType.Incoming, ByteToHex(comBuffer) + "\n");
break;
default:
//read data waiting in the buffer
string str = comPort.ReadExisting();
//display the data to the user
DisplayData(MessageType.Incoming, str + "\n");
break;
            }
        }
```

```
        #region SetParityValues
publicvoid SetParityValues(object obj)
        {
foreach (string str inEnum.GetNames(typeof(Parity)))
            {
                ((ComboBox)obj).Items.Add(str);
            }
        }
        #endregion

        #region SetStopBitValues
publicvoid SetStopBitValues(object obj)
        {
foreach (string str inEnum.GetNames(typeof(StopBits)))
            {
                ((ComboBox)obj).Items.Add(str);
            }
        }
        #endregion

        #region SetPortNameValues
publicvoid SetPortNameValues(object obj)
        {

foreach (string str inSerialPort.GetPortNames())
            {
                ((ComboBox)obj).Items.Add(str);
            }
        }
        #endregion

    }
}
```