Path Planning for Autonomous Mobile Robots

Dissertation Submitted in partial fulfillment of

the requirement for the degree of

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

By

Saumya Saurabh (121075)

Jayesh Raghav (121122)

UNDER THE GUIDANCE OF

Dr. RAJIV KUMAR

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

JUNE 2016

TABLE OF CONTENTS

DECLARATION

I hereby declare that the work reported in the B-Tech thesis entitled **"Path Planning for Autonomous Mobile Robots"** submitted at **Jaypee University of Information Technology, Waknaghat, India,** is an authentic record of my work carried out under the supervision of Dr**. RAJIV KUMAR**. I have not submitted this work elsewhere for any other degree or diploma.

SAUMYA SAURABH JAYESH RAGHAV

Department of Electronics and Communication Engineering

Jaypee University of Information Technology, Waknaghat, India

26th May 2016

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY

WAKNAGHAT

SOLAN, HIMACHAL PRADESH

Date: 26th May 2016

CERTIFICATE

This is to certify that the work reported in the B-Tech. thesis entitled **"Path Planning for Autonomous Mobile Robots"**, submitted by **Saumya Saurabh** and **Jayesh Raghav at Jaypee University of Information Technology, Waknaghat, India,** is a bonafide record of their original work carried out under my supervision. This work has not been submitted partially or wholly to any other university or institution for award of this or any other degree program.

ACKNOWLEDGEMENT

The realisation of this project depended upon the shared efforts from several experts of knowledge and sources of information.

I am indebted to management of JUIT to allow me to carry out this project. I express my sincere gratitude and indebtedness to Dr. Rajiv Kumar, ASSISTANT PROFESSOR (ECE), arranging necessary facilities for the project and providing me an opportunity to involve myself in the topic of the project and with the working environment and people of JUIT.

I owe my sincere gratitude to my course coordinator Dr Rajiv Kumar, associate professor(ECE), for his valuable and inspiring guidance towards the subject matter and progress on the topic 'Path Planning for Autonomous Mobile Robots' and providing valuable information for progress of the project.

ABSTRACT

In this present work, we present an algorithm for path planning to a target for mobile robot in unknown environment. The proposed algorithm allows a mobile robot to navigate through static obstacles, and finding the path in order to reach the target without collision. This algorithm provides the robot the possibility to move from the initial position to the final position (target). The proposed path finding strategy is designed in a grid-map form of an unknown environment with static unknown obstacles. The robot moves within the unknown environment by sensing and avoiding the obstacles coming across its way towards the target. When the mission is executed, it is necessary to plan an optimal or feasible path for itself avoiding obstructions in its way and minimizing a cost such as time, energy, and distance. The proposed path planning must make the robot able to achieve these tasks: to avoid obstacles, and to make one's way toward its target. The algorithms are implemented in Matlab, afterwards tested with Matlab GUI; whereby the environment is studied in a two dimensional coordinate system. The simulation part is an approach to the real expected result; this part is done using Matlab to recognize all objects within the environment and since it is suitable for graphic problems. Taking the segmented environment issued from Matlab development, the algorithm permits the robot to move from the initial position to the desired position following an estimated trajectory using Maps in Matlab GUI.

Also we have created an autonomous robot for hardware implementation. The obstacle avoidance robotic vehicle uses ultrasonic sensors for its movements. A UNO of Arduino family is used to achieve the desired operation. The motors are connected through motor driver IC to Arduino Uno. The ultrasonic sensor is attached in front of the robot.

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In robotic navigation, path planning is aimed at getting the optimum collision-free path between a starting and target locations. The planned path is usually decomposed into line segments between ordered sub-goals or way points. In the navigation phase, the robot follows those line segments toward the target. The navigation environment is usually represented in as configuration space. Depending on the surrounding environment and the running conditions, the optimality criterion for the path is determined. For example, in most of indoor navigation environments, the optimum path is the safest one, i.e. being as far as possible from the surrounding obstacles, whereas for outdoor navigation, the shortest path is more recommended.

The aim of this project is to compute the optimum path between a start and a target point in a given navigation map.

The idea of the project is to represent every obstacle as a charge that has a repulsive potential. In the other hand the target represent a charge has an attractive potential.

By combining these potentials, a new map will be generated with an optimum path.

CHAPTER 2

SOFTWARE IMPLEMENTATION USING MATLAB GUI

2.1 THE ALGORITHM

The algorithm deals with every obstacle as a point source of repulsive potential effect on the robot with an inverse proportional of the distance square between them. This force can be computed by:

W/(J-y) ^2+(I-x) ^2

Where I and J represent all points in the map. X and Y represent the center of the obstacle
W represent the weight of the charge

This generate a matrix map and every element of this matrix carry the amount of potential found on (I, J) coordinates. This map will have large values at the obstacles centers and boundaries.

The other forces are generated by the target and it can be represented as a source of attractive potential and it is directly proportional with the distance with the robot. This force can be computed by:

W*sqrt((J-GoalY)^2+(I-GoalX)^2)

Where GoalY, GoalX is the target coordinates

This will generate a matrix map that has a minimum value at the target and a maximum at the robot position.

Then by summing the two forces map we will get a map with repulsive and attractive forces. These forces with each other will draw the path of the robot.

Figure below shows the two forces and as we can see the repulsive forces was drawn as a huge mount but the attractive one as a valley.

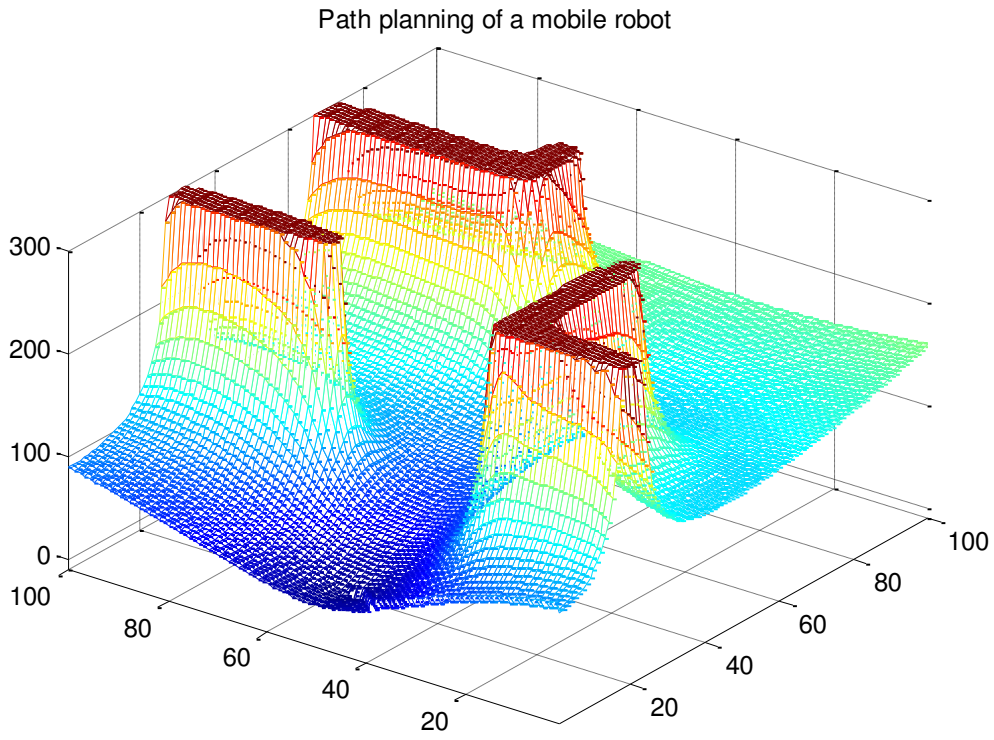

Fig 2.1.1

The table below is a sample of this map and show that the 300 value represent an obstacle.

131.1	141.9	155.7	174.2	199.8	237.7	297.8	300	300	300
130.9	141.8	155.8	174.5	200.4	238.6	299.6	300	300	300
130.6	141.7	155.9	174.7	201.0	240.5	300	300	300	300
130.2	141.4	155.8	174.9	201.4	241.1	300	300	300	300
129.7	141.1	155.7	175.0	201.6	240.6	300	300	300	300
129.1	140.8	155.6	175.2	202.1	241.4	300	300	300	300
128.5	140.3	155.5	175.4	202.9	243.7	300	300	300	300
127.8	139.8	155.3	175.6	203.7	245.4	300	300	300	300
126.9	139.2	155.0	175.9	204.5	246.0	300	300	300	300
125.9	138.5	154.7	176.2	205.6	248.0	300	300	300	300

Fig 2.1.2

2.2 FINDING THE PATH

The final part now is to find the path and this step takes the all map elements that contains the entire obstacle and goal forces values and made some steps to get the path as follows:

1. Assume the size of the map is 100*100, the algorithm first step is to begin from the first element (1,1) or origin and form a 3*3 sub map that contain the first 9 elements.

2. After forming the 3*3 map from the corner of the largest map, it began to compare the 9 element with each other and determine the smallest one that represents the smallest potential force found.

3. Then make a shift of the sub map by the way that the smallest element found in step2 will be the center of the new map.

4. Comparing the smallest element coordinates with the target coordinates, if they are equal then stop else continue.

5. Repeat steps 2, 3, 4 until finishing the 100*100 map.

2.3 LOCAL MINIMUM PROBLEM

If sometimes the robot falls in a region that results from step 2 after forming the 3*3 sub map and finding that the new target is equal to the old one and it is not equal the final coordinates, this region called a local minimum region or problem.

The proposed solution of this problem is to assume that at the local minimum point it will found an obstacle with high potential and repeating step2 will get the robot out of this region.

The example below illustrates how the algorithm works. In the first table the first element is local minimum then we replace its value with 300. After this step it found its way to the next one that also found as a local minimum then repeat until it get out.

96.9	96.91	97.1	97.5	98.2	99.2	100.6	.102	105.1	108.6
97.3	97.4	97.7	98.2	99.0	100.2	101.7	103.9	106.7	110.6
97.7	97.9	98.2	98.9	99.8	101.1	102.8	105.2	108.3	112.4
98.1	98.4	98.8	99.57	100.6	102.0	103.9	106.4	109.8	114.2
98.5	98.8	99.4	100.2	101.3	102.9	104.9	107.6	111.2	115.8
99.0	99.3	99.9	100.8	102.1	103.7	105.9	108.8	112.5	117.3
99.4	99.8	100.5	101.5	102.8	104.6	106.9	109.9	113.7	118.8
99.85	100.3	101.0	102.1	103.5	105.4	107.8	110.9	114.9	120.1
100.2	100.8	101.6	102.7	104.2	106.1	108.7	111.9	116.0	121.4
100.6	101.2	102.1	103.3	104.8	106.9	109.5	112.8	117.1	122.6

Fig 2.3.1

300	96.91	97.1	97.5	98.2	99.2	100.6	.102	105.1	108.6
97.3	97.4	97.7	98.2	99.0	100.2	101.7	103.9	106.7	110.6
97.7	97.9	98.2	98.9	99.8	101.1	102.8	105.2	108.3	112.4
98.1	98.4	98.8	99.57	100.6	102.0	103.9	106.4	109.8	114.2
98.5	98.8	99.4	100.2	101.3	102.9	104.9	107.6	111.2	115.8
99.0	99.3	99.9	100.8	102.1	103.7	105.9	108.8	112.5	117.3
99.4	99.8	100.5	101.5	102.8	104.6	106.9	109.9	113.7	118.8
99.85	100.3	101.0	102.1	103.5	105.4	107.8	110.9	114.9	120.1
100.2	100.8	101.6	102.7	104.2	106.1	108.7	111.9	116.0	121.4
100.6	101.2	102.1	103.3	104.8	106.9	109.5	112.8	117.1	122.6

Fig 2.3.2

300	300	97.1	97.5	98.2	99.2	100.6	.102	105.1	108.6
97.3	97.4	97.7	98.2	99.0	100.2	101.7	103.9	106.7	110.6
97.7	97.9	98.2	98.9	99.8	101.1	102.8	105.2	108.3	112.4
98.1	98.4	98.8	99.57	100.6	102.0	103.9	106.4	109.8	114.2
98.5	98.8	99.4	100.2	101.3	102.9	104.9	107.6	111.2	115.8
99.0	99.3	99.9	100.8	102.1	103.7	105.9	108.8	112.5	117.3
99.4	99.8	100.5	101.5	102.8	104.6	106.9	109.9	113.7	118.8
99.85	100.3	101.0	102.1	103.5	105.4	107.8	110.9	114.9	120.1
100.2	100.8	101.6	102.7	104.2	106.1	108.7	111.9	116.0	121.4
100.6	101.2	102.1	103.3	104.8	106.9	109.5	112.8	117.1	122.6

Fig 2.3.3

300	300	300	97.5	98.2	99.2	100.6	.102	105.1	108.6
97.3	97.4	97.7	98.2	99.0	100.2	101.7	103.9	106.7	110.6
97.7	97.9	98.2	98.9	99.8	101.1	102.8	105.2	108.3	112.4
98.1	98.4	98.8	99.57	100.6	102.0	103.9	106.4	109.8	114.2
98.5	98.8	99.4	100.2	101.3	102.9	104.9	107.6	111.2	115.8
99.0	99.3	99.9	100.8	102.1	103.7	105.9	108.8	112.5	117.3
99.4	99.8	100.5	101.5	102.8	104.6	106.9	109.9	113.7	118.8
99.85	100.3	101.0	102.1	103.5	105.4	107.8	110.9	114.9	120.1
100.2	100.8	101.6	102.7	104.2	106.1	108.7	111.9	116.0	121.4
100.6	101.2	102.1	103.3	104.8	106.9	109.5	112.8	117.1	122.6

Fig 2.3.4

2.4 AVOIDING LOCAL MINIMUM PROBLEM

Several methods have been suggested to deal with the local minimum phenomenon in potential filed m. One idea is to avoid the local minimum by incorporating the potential field with the high-level planner, so that the robot can use the information derived from its sensor, but still plan globally.
Another set of approaches are to allow the robot to go into local minimum state, but then try to fix this situation by:

- Backtracking from the local Minimum and then using another strategy to avoid the local minimum.
- Doing some random movements, with the hope that these movements will help escaping the local minimum.
- Using a procedural planner, such as wall following, or using one of the bug algorithms to avoid the obstacle where the local Minimum is.
- Using more complex potential fields that are guaranteed to be local minimum free, like harmonic potential fields.
- Changing the potential field properties of the position of the local minimum. So that if the robot gets repelled from it gradually.

All these approaches rely on the fact that the robot can discover that it is trapped, which is also an ill-defined problem.

The method used in this work, is similar to the last point, where the properties of the potential fields are changed.

When the robot senses that it is trapped, a new force called virtual free space force, (or virtual force simply), will be applied to it.

The virtual force is proportional to the amount of free space around the robot, and it helps pulling the robot outside of the local minimum area. After the virtual force is applied, the robot will be dragged outside of the local minimum and it will begin moving again using the potential field planner. However, it is now unlikely that it will be trapped again to the same local minimum.

More formally, the virtual force Ff is proportional to the free space around the robot. It is calculated by:

$$F_f = F_{cf}\,(\cos(\theta)\, e_x + \sin(\theta) e_y) \quad [1]$$

where θ is robot to free space orientation and F_{cf} is force constant

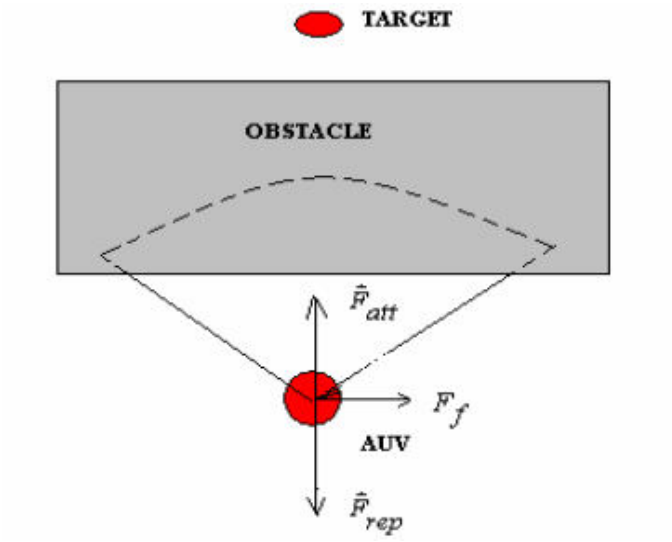

Fig 2.4.1

Unfortunately, the project does not discuss in detail what robot to free space. Orientation means, nor does it include any figure explaining ex, and ey. However, it can be inferred that will be toward the free space around the robot, in an opposite direction of the obstacle. Its two components on x and y can be described by cos(θ) ex and sin(θ)ey

The total force will be the sum of the attractive force (derived from the goal potential field), the repulsive force (derived from the obstacle fields), and the virtual force:

$$\mathbf{F} = \mathbf{F}_{att} + \mathbf{F}_{rep} + \mathbf{F}_{f}$$

Finally, to detect that the robot is trapped, we use a position estimator to estimate the current position of the robot (open-loop). If the current position does not change for a considerable amount of time (a predefined threshold), the virtual force is generated to pull out the robot from the local minimum.

Note that, the robot speed is decreased once it approaches an obstacle, (due to the repulsive force). Therefore, the robot cannot stay in a non-local minimum place for a long time, unless if this is planned by the high-level planner.

2.5 RESULTS

The Matlab GUI program was generated to perform this algorithm. The GUI has user friendly tools to make it easy to test all possible ideas.

Figure show the interface of GUI.

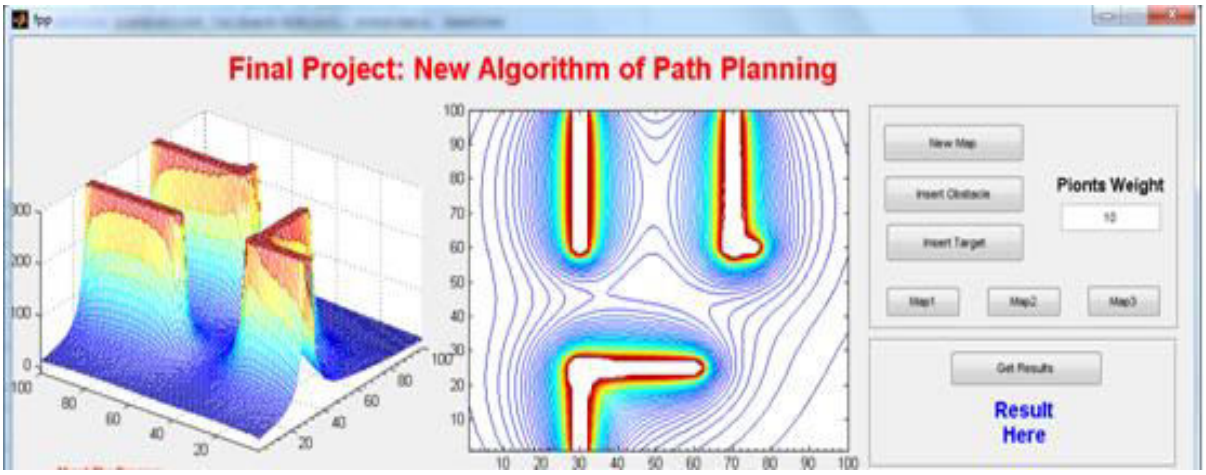

Fig 2.5.1

The manual of using this GUI is:

This Program produce a path planning for a mobile robot centered at the origin (1,1) to the target chosen according to you and avoiding any obstacle.

You can use predefined maps (map1, map2, map3) to define the obstacles in the map or use create new map to make your own.

Insert Obstacle: this button guides you to draw a wall and the variable "Weight" define the effects of the wall.

Insert Target: this button allows you to position the target.

Finally, by the button "Get Results" you can get if the path planning algorithm is success or failed to find a path.

The "Result" can be "Path has been found" or "No solution has been found".

Note: This algorithm handles the Local Minimum Problem and if the handling process failed to find a solution then No solution result appeared.

2.6 TESTING THE ALGORITHM

We have 3 predefined maps to test it or we can create our own to test it

Map1:

After choosing map1 and mark the target at approximately (9.4, 8.8) the right graph show the obstacles, target potential plus the path created by the algorithm. The left graph shows the potential in another way.

Note that there is an extra high potential was appeared in the resulted map. This means that there was a local minimum problem was solved.

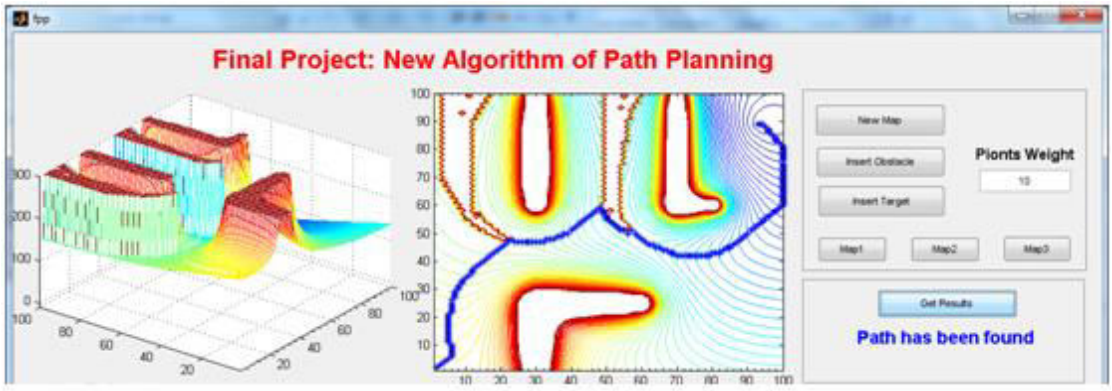

Fig 2.6.1

Map 2:

The same was found when testing map 2

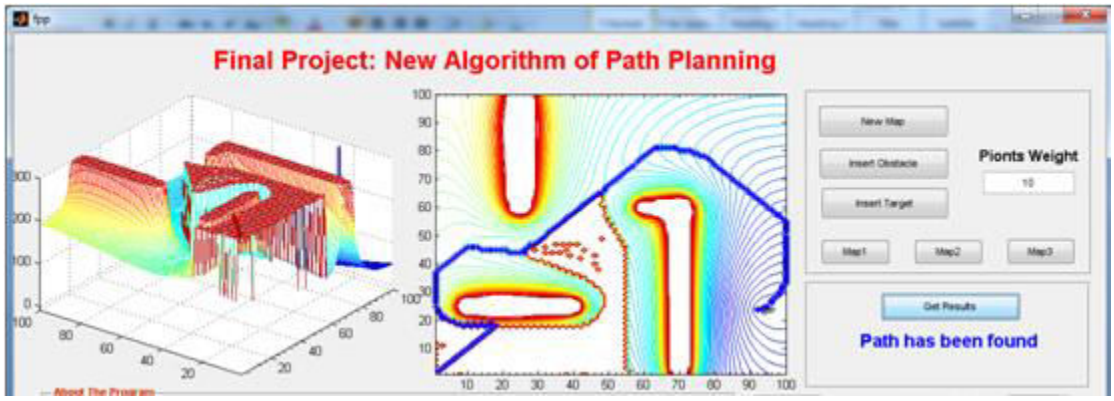

Fig 2.6.2

Map3:

The same was found when testing map 3

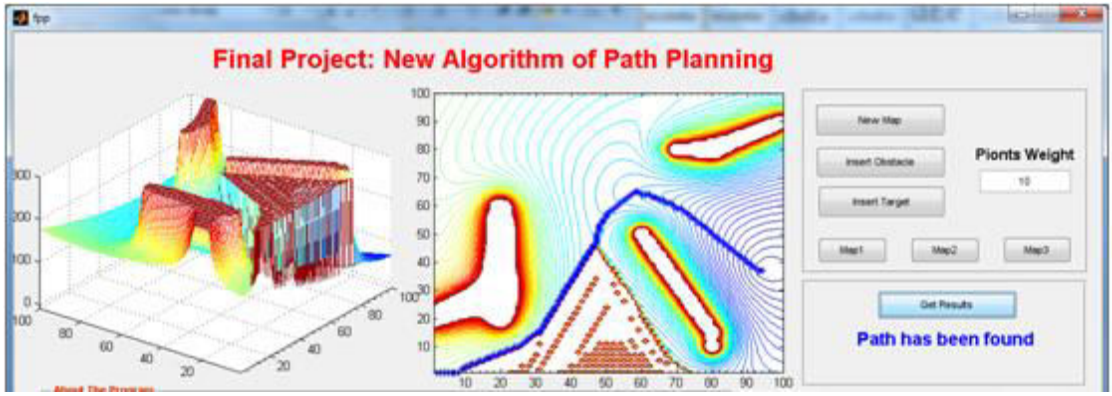

Fig 2.6.3

New Map:

We define a map and mark a target and the result were excellent

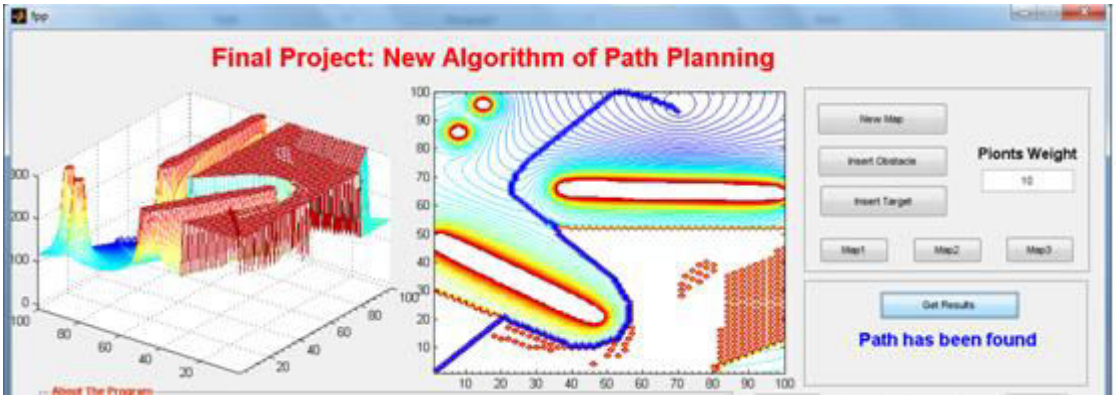

Fig 2.6.4

In all the previous results if we decrease the weight of the obstacles the robot will found a path easily but if increase the path will be more difficult.

New map:

The next map has obstacles as figure below and when the target has been chosen to be at (3,9.2) a "No solution" was the result and the resulting path was wrong because it cut the bottom obstacle.

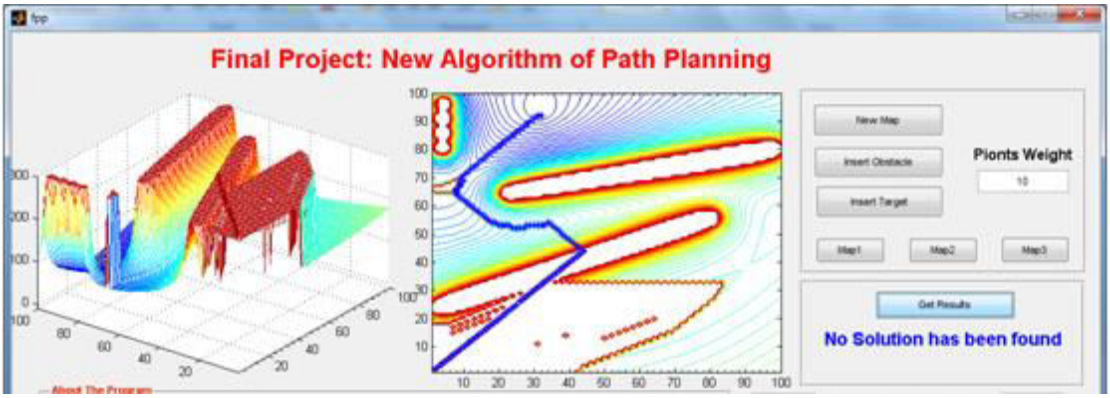

Fig 2.6.5

2.7 MATLAB CODES

A) TO GET PATH:

```
function [A,B,LMF,NoSol,xx,yy]=getpath(map)

x=map(:,2);
y=map(:,1);
w=map(:,3);
R=[];
A1=[];
GoalY=y(length(x));
GoalX=x(length(x));

 for I=1:100
  for J=1:100
      R=[];
      for r=1:length(x)-1
          R=[R w(r)/((J/10-y(r))^2+(I/10-x(r))^2)];
      end
         RG = sqrt((J/10-GoalY)^2+(I/10-GoalX)^2);
         A1(I,J) = sum(R)+ 15*RG;
            if (A1(I,J)>300)
                A1(I,J)=300;
            end
 end
 end

i=1:100;
j=1:100;
A=A1;
LMF=1;
while(LMF==1)
% Searching the path
a=1;
b=1;
LMF=0;
a2=0;
b2=0;
B=zeros(length(A));
```

```
value=(abs(a-GoalX*10)+abs(b-GoalY*10));
while value>0,    %target error
[a1,b1] = checking(A,a,b,100,100);
if((a1==a)&(b1==b))     %Local min or target
a=a1;
b=b1;
B(a,b)=w(length(w));
LMF=0;
break;
else
    if(a1==a2) & (b1==b2)    %Local min exist
        LMF=1;
        A(a1,b1)=300;
       % B(a1,b1)=0;
        break;
    end
a2=a;
b2=b;
a=a1;
b=b1;
B(a,b)=w(length(w));
end
value=(abs(a-GoalX*10)+abs(b-GoalY*10));
end

% if(LMF==1)
%     display('Local Minimum exists')
% end
end

xx=[];
yy=[];
for ii=1:100
    for jj=1:100
        if B(ii,jj)~=0
            xx=[xx ii];
            yy=[yy jj];
        end
    end
end

NoSol=0;
for ii=1:100
    for jj=1:100
        if (B(ii,jj)==w(length(w)))
            if (A1(ii,jj)==300)
                NoSol=1;
            end
        end
    end
end

return
```

B) CHECKING OF A SOLUTION:

```
function [x,y] = checking(A,a,b,X,Y);
% Usable to mobile robot navigation problem
% b
```

```
% ^
% | A(a,b)
%
% A13 A23 A33
% A12 X A32
% A11 A21 A31 -> a
A11 = 50000;
A12 = 50000;
A13 = 50000;
A21 = 50000;
A23 = 50000;
A31 = 50000;
A32 = 50000;
A33 = 50000;
if (a>=2)&(b>=2)&(a<X)&(b<Y)
A11 = A(a-1,b-1);
A21 = A(a,b-1);
A31 = A(a+1,b-1);
A12 = A(a-1,b);
A32 = A(a+1,b);
A13 = A(a-1,b+1);
A23 = A(a,b+1);
A33 = A(a+1,b+1);
elseif (a>=2)&(b>=2)&(a<X) % eliminates b+1
A11 = A(a-1,b-1);
A21 = A(a,b-1);
A31 = A(a+1,b-1);
A12 = A(a-1,b);
A32 = A(a+1,b);
elseif (a>=2)&(a<X)&(b<Y) % eliminates b-1
A12 = A(a-1,b);
A32 = A(a+1,b);
A13 = A(a-1,b+1);
A23 = A(a,b+1);
A33 = A(a+1,b+1);
elseif (a>=2)&(b>=2)&(b<Y) % eliminates a+1
A11 = A(a-1,b-1);
A21 = A(a,b-1);
A12 = A(a-1,b);
A13 = A(a-1,b+1);
A23 = A(a,b+1);
elseif (b>=2)&(a<X)&(b<Y) % eliminates a-1
A21 = A(a,b-1);
A31 = A(a+1,b-1);
A32 = A(a+1,b);
A23 = A(a,b+1);
A33 = A(a+1,b+1);
elseif (a==1)&(b==1)
A32 = A(a+1,b);
A23 = A(a,b+1);
A33 = A(a+1,b+1);
elseif (a==X)&(b==1)
A12 = A(a-1,b);
A13 = A(a-1,b+1);
A23 = A(a,b+1);
elseif (a==1)&(b==Y)
A21 = A(a,b-1);
A31 = A(a+1,b-1);
A32 = A(a+1,b);
elseif (a==X)&(b==Y)
A11 = A(a-1,b-1);
A21 = A(a,b-1);
```

```
A12 = A(a-1,b);
elseif (a==1)
A21 = A(a,b-1);
A23 = A(a,b+1);
A31 = A(a+1,b-1);
A32 = A(a+1,b);
A33 = A(a+1,b+1);
elseif (a==X)
A21 = A(a,b-1);
A23 = A(a,b+1);
A11 = A(a-1,b-1);
A12 = A(a-1,b);
A13 = A(a-1,b+1);
elseif (b==1)
A12 = A(a-1,b);
A13 = A(a-1,b+1);
A23 = A(a,b+1);
A32 = A(a+1,b);
A33 = A(a+1,b+1);
elseif (b==Y)
A11 = A(a-1,b-1);
A21 = A(a,b-1);
A31 = A(a+1,b-1);
A12 = A(a-1,b);
A32 = A(a+1,b);
end
% A13 A23 A33
% A12 X A32
% A11 A21 A31 =>a
if (A11<A21)&(A11<A31)&(A11<A12)&(A11<A32)&(A11<A13)&(A11<A23)&(A11<A33)
x=a-1;
y=b-1;
elseif (A21<A31)&(A21<A12)&(A21<A32)&(A21<A13)&(A21<A23)&(A21<A33)
x=a;
y=b-1;
elseif (A31<A12)&(A31<A32)&(A31<A13)&(A31<A23)&(A31<A33)
x=a+1;
y=b-1;
elseif (A12<A32)&(A12<A13)&(A12<A23)&(A12<A33)
x=a-1;
y=b;
elseif (A32<A13)&(A32<A23)&(A32<A33)
x=a+1;
y=b;
elseif (A13<A23)&(A13<A33)
x=a-1;
y=b+1;
elseif (A23<A33)
x=a;
y=b+1;
else
x=a+1;
y=b+1;
end
```

CHAPTER 3

AUTONOMOUS MOBILE ROBOT

3.1 INTRODUCTION TO AUTONOMOUS ROBOTS

A robot is autonomous if it has the computational resources - both in terms of hardware and software - other than real time interference from a human agent, to estimate how it is physically embedded in the environment to compute best possible actions bounded by some constraints to perceive and move if needed, to achieve a set of goals. According to this working definition, a robot's ability to estimate its current state (how it is physically embedded in the environment) is an essential component of autonomy. Then, it has to have adequate computational resources at its disposal to take an action within bounds, to perceive the environment more if needed, and move if needed, to achieve a given goal.

3.2 TYPES OF AUTONOMOUS ROBOTS

Out of three types of manipulation robotic system, the autonomous system is further classified into four types:

- Programmable
- Non-programmable
- Adaptive
- Intelligent

1. **Programmable Automatic Robot :**

Fig 3.2.1

A programmable robot is a first generation robot with an actuator facility on each joint. The robots can be reprogrammable based on the kind of application they are commissioned to. The function and application of the robots can be changed by reprogramming after the robot is programmed once to perform a function in the given pattern and fixed sequence.

Robot kits like Lego mind storms, Bioloid from programmable Robotics can help the students to learn about its programming and working. The advanced mobile robot, robotic arms and gadgeteer are some of the examples of these programmable robots.

The main drawback of this autonomous robot is that once programmed it persists operation even if there is a need to change its task (in the event of emergency). These robots can be used in different applications like mobile robotics, industrial controlling and space craft applications.

2. Non-Programmable Automatic Robot :

Fig 3.2.2

This robot is one of the basic types of robot, in fact, a non-programmable robot. This robot is not even considered as a robot, but is an exploiter lacking reprogrammable controlling device. The mechanical arms used in industries are some of the examples of these types of robots wherein the robots are generally attached to the programmable devices used in industries for mass production as shown in the figure.

These types of robots find applications in some of the devices including path guiders and medical products' carriers and also some line follower robots.

3. Adaptive Robot :

Fig 3.2.3

Adaptive robots are also industrial robots that can be adapted independently to various ranges in the process. However, these robots are more sophisticated than programmable robots. These can be adapted up to a certain extent, and after evaluation they can perform the action required in that adapted area. These robots are mostly equipped with sensors and control systems.

Sensors are used to sense environmental conditions, process variables and other parameters related to a particular task. Feedback control system accesses these signals from the sensors, and depending on the algorithm implemented, it controls the outputs.

Adaptive robots are mainly used in applications such as spraying and welding systems. Robotic gripper and 2- finger adaptive gripper are examples of this autonomous robot. These robots can be used in different applications like aerospace, medical, consumer goods, house-hold applications and manufacturing industrial areas.

4. Intelligent Robots :

Fig 3.2.4

Intelligent robots, as the name suggests, are the most intelligent of all the other types of robots with sensors and microprocessors for storing and processing the data. These robots performance is highly efficient due to their situation-based analysing and task performing abilities. Intelligent robots can sense the senses like pain, smell and taste and are also capable of vision and hearing, and – in accordance, perform the actions and expressions like emotions, thinking and learning.

These robots find their applications in the fields like medical, military applications and home appliance control systems, etc.

These are the four different types of autonomous robotic systems, which can be implemented over a wide range of applications. Furthermore, for any queries regarding the real-time robots and their implementation in robotics projects, you can comment in the comments section given below.

3.3 APPLICATIONS OF AUTONOMOUS ROBOTS

For real world applications the robot must be sold in numerous copies to customer, who will read a short set of recommendations, power the robot, and check from time to time that the work is being properly done. Customers are not willing to spend time in instructing their robot, letting it carefully explore the environment, and buy the risk of sub-optimal performance. Current limitations in energy autonomy naturally favour “white-collar” applications of autonomous robots, such as surveillance.

Autonomous robots find numerous applications in diverse fields such as defence, medical, surveillance, security, and space exploration. These applications are rapidly growing in scope and implementation, and will include environmental membrane filtration and medical treatment. Researchers are developing autonomous unmanned aerial vehicles (UAVs) that can fly and work together in groups. They would have invented functionally improved surveillance and rescue-bots, pets that double up as security guards, elderly care robots, robotic public transportation, and multipurpose home-cleaning robots. CSIRO autonomous robotic systems can assist or replace people in tasks that are repetitive, difficult, unpleasant, or performed in hazardous environments. They can be used across a wide range of industries. Some important fields of use are:

Aerial :

Autonomous robotic systems can be used to carry out hazardous or difficult missions that until now have been performed by people. CSIRO, collaborating with the Queensland University of Technology, Boeing Research and Technology Australia, and Insitu Pacific, are developing autonomous unmanned aircraft for use in a range of applications including invasive species surveying over tropical rainforests.

CSIRO developed technology for the Smart Skies Project, a multi-award winning international research project that developed an electro-optical mid-air collision avoidance system, a static obstacle avoidance system, a mobile ground-based aircraft

Fig 3.3.1

Mining :

Remote telerobotic systems increase efficiency, productivity and profitability, and remove people from hazardous and inhospitable working environments by allowing them to remotely control mining equipment traditionally, remote operation in the mining industry involved a human operator relying only on video streaming to make decisions. CSIRO has developed technologies that improve mining operations by using a variety of sensors to provide additional information to the human operator in real-time, thus allowing better control over the equipment while reducing human fatigue and errors

Fig 3.3.2

Underwater monitoring :

CSIRO Autonomous Underwater Vehicles (AUVs) use video cameras as one of the primary sensors for navigation, and are ideal for data collection, inspecting and cataloguing natural habitats. CSIRO is developing technologies that allow the underwater robot to immediately and autonomously recognise objects in the video stream and make decisions accordingly. This enables adaptive mission planning with the vehicle changing its survey plan based on data obtained in real-time

Fig 3.3.3

Education :

CSIRO, partnering with National Museum education experts and the Department of Broadband, Communications and the Digital Economy, has developed an autonomous robot that allows remote visitors, such as school students in rural Australia, to virtually visit the National Museum through a high-speed broadband connection. The robot navigates itself around the museum alongside an educator, and remote visitors can talk to the educator through a video chat session and see the museum gallery through the robot cameras

Fig 3.3.4

CHAPTER 4

CONSTRUCTION AND WORKING OF AN AUTONOMOUS ROBOT

4.1 INTRODUCTION TO COMPONENTS

4.1.a ARDUINO UNO HC

Arduino is a software company, project, and user community that designs and manufactures computer open-source hardware, open-source software, and microcontroller-based kits for building digital devices and interactive objects that can sense and control physical devices.

The project is based on microcontroller board designs, produced by several vendors, using various microcontrollers. These systems provide sets of digital and analog I/O pins that can interface to various expansion boards (termed shields) and other circuits. The boards feature serial communication interfaces, including Universal Serial Bus (USB) on some models, for loading programs from personal computers. For programming the microcontrollers, the Arduino project provides an integrated development environment (IDE) based on a programming language named Processing, which also supports the languages C and C++.

The Arduino project provides the Arduino integrated development environment (IDE), which is a cross-platform application written in the programming language Java. It originated from the IDE for the languages Processing and Wiring. It is designed to introduce programming to artists and other newcomers unfamiliar with software development. It includes a code editor with features such as syntax highlighting, brace matching, and automatic indentation, and provides simple one-click mechanism to compile and load programs to an Arduino board. A program written with the IDE for Arduino is called a "sketch".

The Arduino IDE supports the languages C and C++ using special rules to organize code. The Arduino IDE supplies a software library called Wiring from the Wiring project, which provides many common input and output procedures. A typical Arduino C/C++ sketch consists of two functions that are compiled and linked with a program stub main() into an executable cyclic executive program:

- setup(): a function that runs once at the start of a program and that can initialize settings.
- loop(): a function called repeatedly until the board powers off.

After compiling and linking with the GNU tool chain, also included with the IDE distribution, the Arduino IDE employs the program avrdude to convert the executable code into a text file in hexadecimal coding that is loaded into the Arduino board by a loader program in the board's firmware.

Most Arduino boards contain an LED and a load resistor connected between pin 13 and ground which is a convenient feature for many tests.

A typical program for a beginning Arduino programmer blinks a light-emitting diode (LED) on and off.

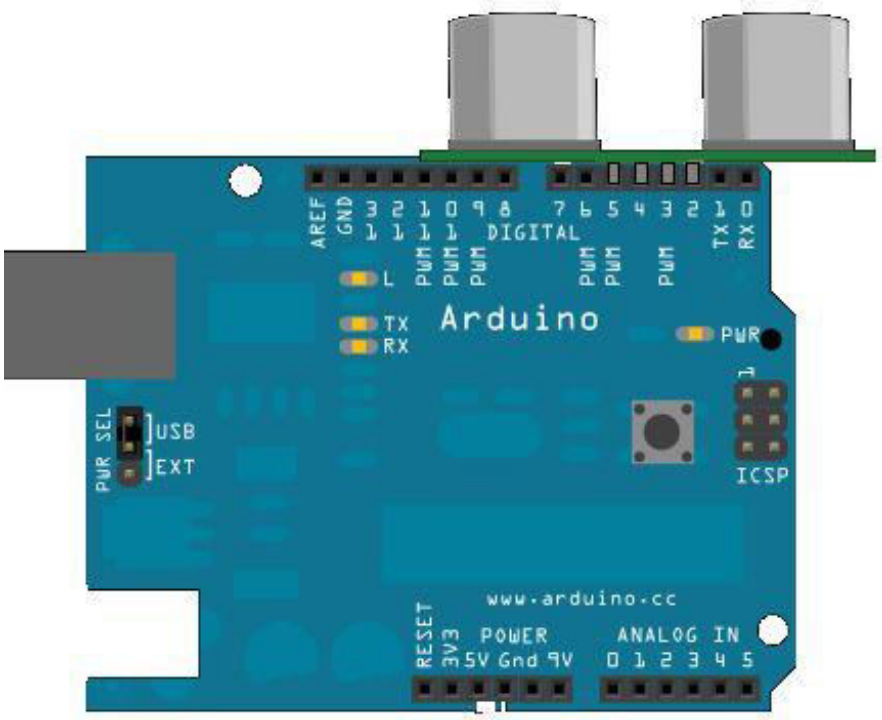

Fig 4.1.a.1

4.1.b SR-04 ULTRASONIC SENSOR

The HC-SR04 Ultrasonic Sensor is a very affordable proximity/distance sensor that has been used mainly for object avoidance in various robotics projects. It essentially gives your Arduino eyes / spatial awareness and can prevent your robot from crashing or falling off a table. It has also been used in turret applications, water level sensing, and even as a parking sensor. This simple project will use the HC-SR04 sensor with an Arduino and a Processing sketch to provide a neat little interactive display on your computer screen.

Fig 4.1.b.1

4.1.c L293D MOTOR DRIVER

L293D is a Motor driver integrated circuit which is used to drive DC motors rotating in either direction. It is a 16-pin IC which can control a set of two DC motors simultaneously. The L293D uses 5V for its own power and external power source is needed to drive the motors, which can be up to 36V and draw up to 600mA.

Working of L293D

There are two drive pins on L293D. Pin 1 (left H-bridge) and pin 9 (right H-bridge). To turn ON the corresponding motor, pin 1 or 9 need to be set to HIGH. If either pin 1 or pin 9 goes low, then the motor in the corresponding section will go OFF (high impedance). These inputs (1 and 9) are the ones that should be used to control motor START/STOP and motor speed under PWM, since there would be high impedance output during low semi period of PWM, it would not provoke overload of the L293D when the motor is turning. Thus, PWM or motor ON/OFF control should never be input to pins 2, 7, 15, 10, which should only be used to control direction (Clockwise – Counter Clockwise).

The direction-defining four Input pins for the L293D are pin 2 and 7 on the left and pin 15 and 10 on the right as shown on the pin diagram. Left input pins will determine the rotation of motor connected on the left side and right input for motor on the right hand side. The motors are rotated on the basis of the inputs provided at the input pins as LOGIC 1 or LOGIC 0.

L293d LOGIC TABLE

Assuming a motor connected on left side output pins (pin 3, 6).

- Pin 2 = Logic 1 and Pin 7 = Logic 0 | Clockwise Direction
- Pin 2 = Logic 0 and Pin 7 = Logic 1 | Anticlockwise Direction
- Pin 2 = Logic 0 and Pin 7 = Logic 0 | Brake (this is not high impedance) (force stop rotation using electric brake = same voltage both pins of the motor = overload while the motor is still running)
- Pin 2 = Logic 1 and Pin 7 = Logic 1 | Brake (this is not high impedance) (force stop rotation using electric brake = same voltage both pins of the motor = overload while the motor is still running)

In a similar way, the motor can be operated across input pins 15 and 10 to control the direction of the motor attached to the H-bridge's right side. Using pins 2 and 7 (15 and 10) to determine motor START/STOP or PWM duties it's dangerous, since there wouldn't be high impedance outputs: Current would flow back during the low semi period of PWM when the motor is turning. For on/off purposes or PWM speed control, pins 1 and 9 should be used.

VOLTAGE SPECIFICATIONS

VCC is the voltage that it needs for its own internal operation 5v; L293D will not use this voltage for driving the motor. For driving the motors, it has a separate provision to provide motor supply VSS (V supply). L293d will use this to drive the motor. It means if you want to operate a motor at 9V then you need to provide a Supply of 9V across VSS Motor supply.

The maximum voltage for VSS motor supply is 36V. It can supply a max current of 600mA per channel. Since it can drive motors Up to 36v hence you can drive pretty big motors with this l293d.VCC pin 16 is the voltage for its own internal Operation. The maximum voltage ranges from 5v and up to 36v.

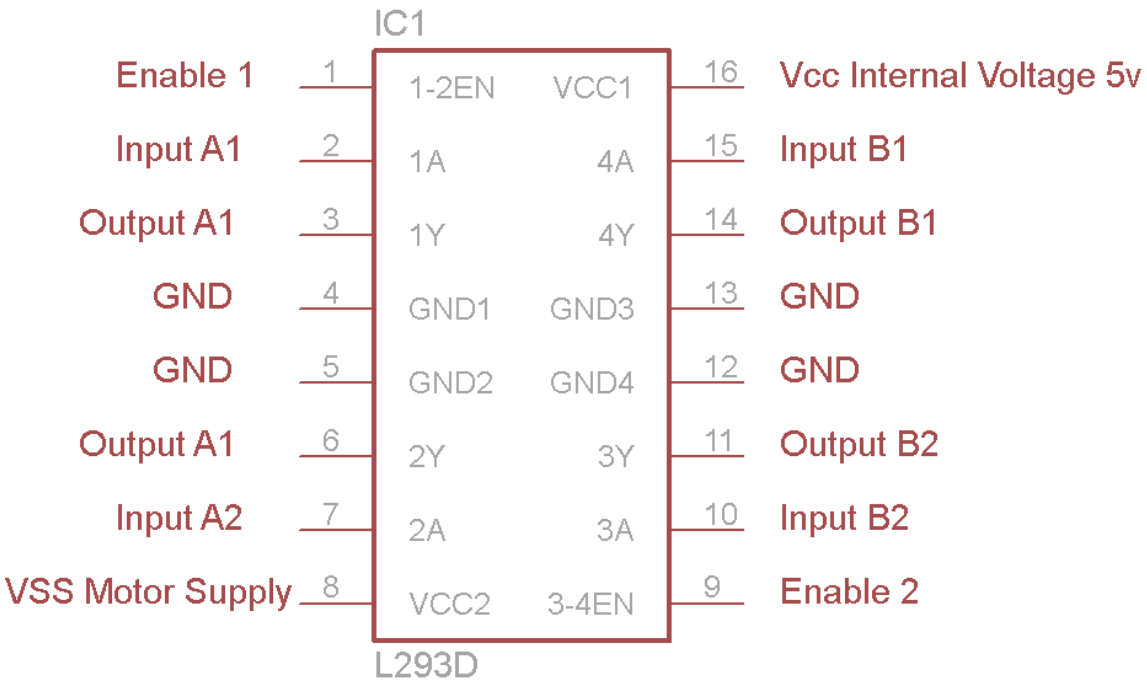

Fig 4.1.c.1

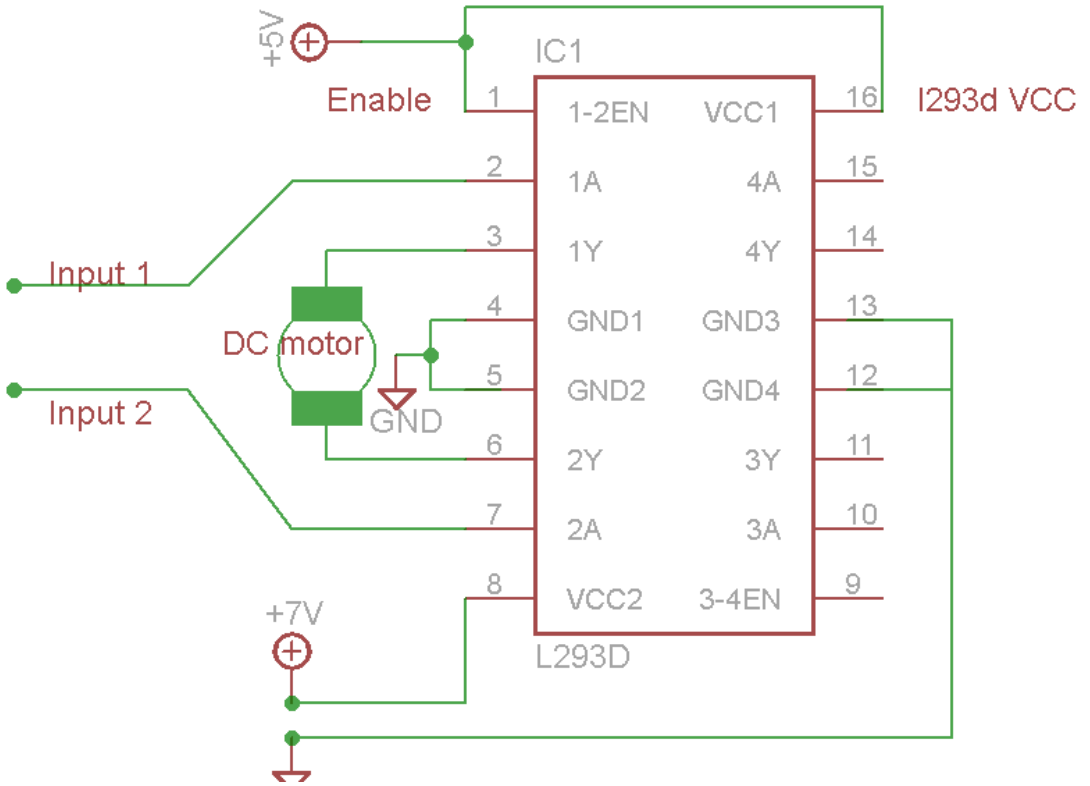

Fig 4.1.c.2

4.1. d OTHER COMPONENTS

- Metal chassis
- 2x BO motors
- 1x caster wheel
- l2x wheels
- 2x 9v battery
- 2x dc battery connectors
- connecting wires
- Ultrasonic sensor clamp (optional)

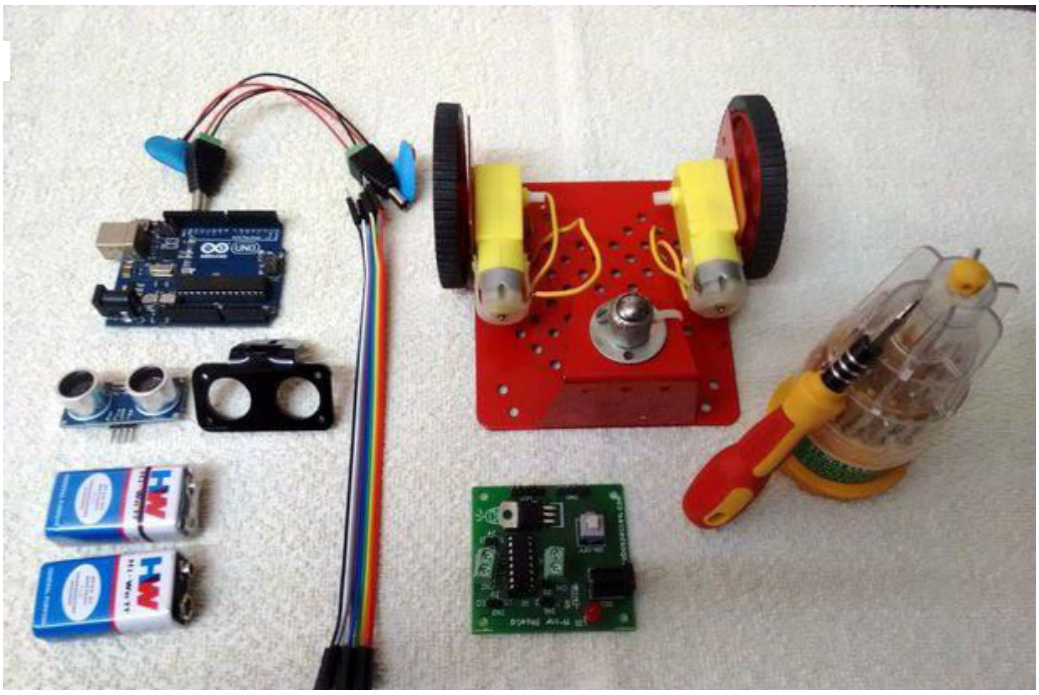

Fig 4.1.d.1

4.2 ULTRASONIC SENSORS AND THEIR WORKING PRINCIPLE

The ultrasonic sensor is used for obstacle detection. Ultrasonic sensor transmits the ultrasonic waves from its sensor head and again receives the ultrasonic waves reflected from an object. The ultrasonic sensor emits the short and high frequency signal. These propagate in the air at the velocity of sound. If they hit any object, then they reflect back echo signal to the sensor. The ultrasonic sensor consists of a multi vibrator, fixed to the base. The multi vibrator is combination of a resonator and vibrator. The resonator delivers ultrasonic wave generated by the vibration.

The ultrasonic sensor actually consists of two parts; the emitter which produces a 40 kHz sound wave and detector detects 40 kHz sound wave and sends electrical signal back to the Arduino.

The ultrasonic sensor enables the robot to virtually see and recognize object, avoid obstacles, measure distance. To use this sensor to measure distance, the robot's brain must measure the amount of time it takes for the ultrasonic sound to travel.

Sound travels at approximately 340 meters per second. This corresponds to about 29.412µs (microseconds) per centimetre. To measure the distance, the sound has travelled we use the formula: Distance = (Time x Speed of Sound) / 2. The "2" is in the formula because the sound has to travel back and forth. First the sound travels away from the sensor, and then it bounces off of a surface and returns back. The easy way to read the distance as centimetres is to use the formula: Centimetres = ((Microseconds / 58.2).

Working principle

The obstacle avoidance robotic vehicle uses ultrasonic sensors for its movements. A UNO of Arduino family is used to achieve the desired operation. The motors are connected through motor driver IC to Arduino Uno. The ultrasonic sensor is attached in front of the robot. Whenever the robot is going on the desired path the ultrasonic sensor transmits the ultrasonic waves continuously from its sensor head. Whenever an obstacle comes ahead of it the ultrasonic waves are reflected back from an object and that information is passed to the Arduino. The Arduino controls the motors based on ultrasonic signals.

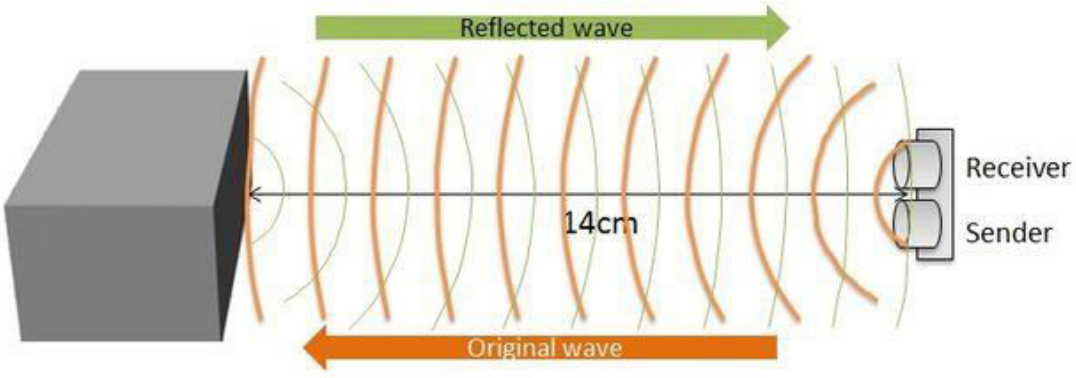

Fig 4.2.1

4.3 STEPS FOR CREATING A ROBOT

Step 1:

Fix the wheels on the metal chassis and mount the Dc BO motors on the back wheels and fix a caster wheel in the front.

Connect the L293D motor driver with the Arduino and dc motors.

CONNECTIONS:

3,6(l293d) to left motor(output)

11,14(l293d) to right motor(output)

2,7,10,15(l293d) to pins 2,3,4,5 of Arduino(inputs)

NOTE: you can use an l293d IC or a readymade module, each module differs so check L293D IC pin diagram to make correct connections.

Step 2:

CONNECTING WITH PING/ ultrasound SENSOR

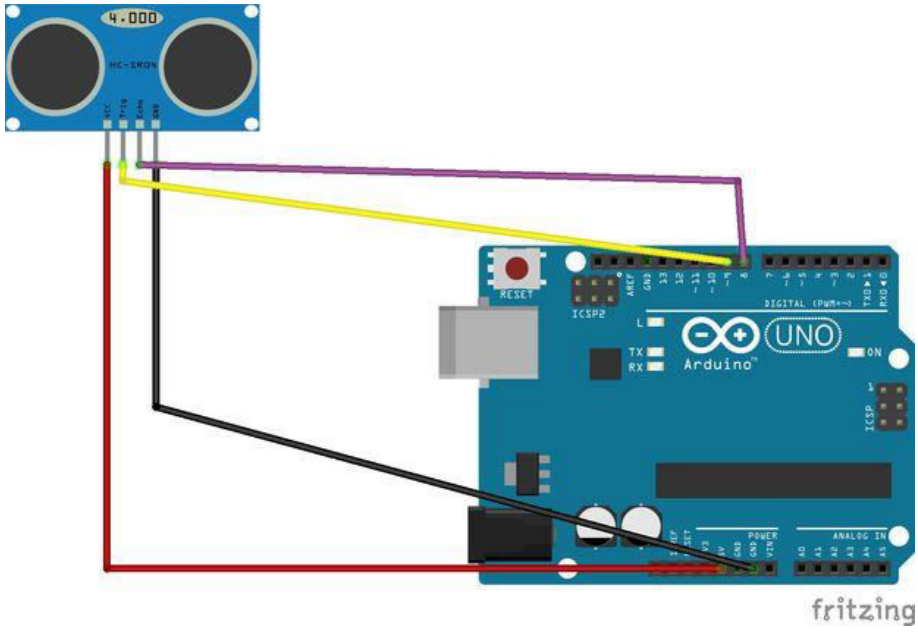

Fig 4.3.1

CONNECTION WITH SENSORS

Connect the ultra-sonic sensor with your Arduino,

Pin 8 of Arduino to echo pin

Pin 9 of Arduino to trigger pin

Vcc-5v

Gnd-gnd

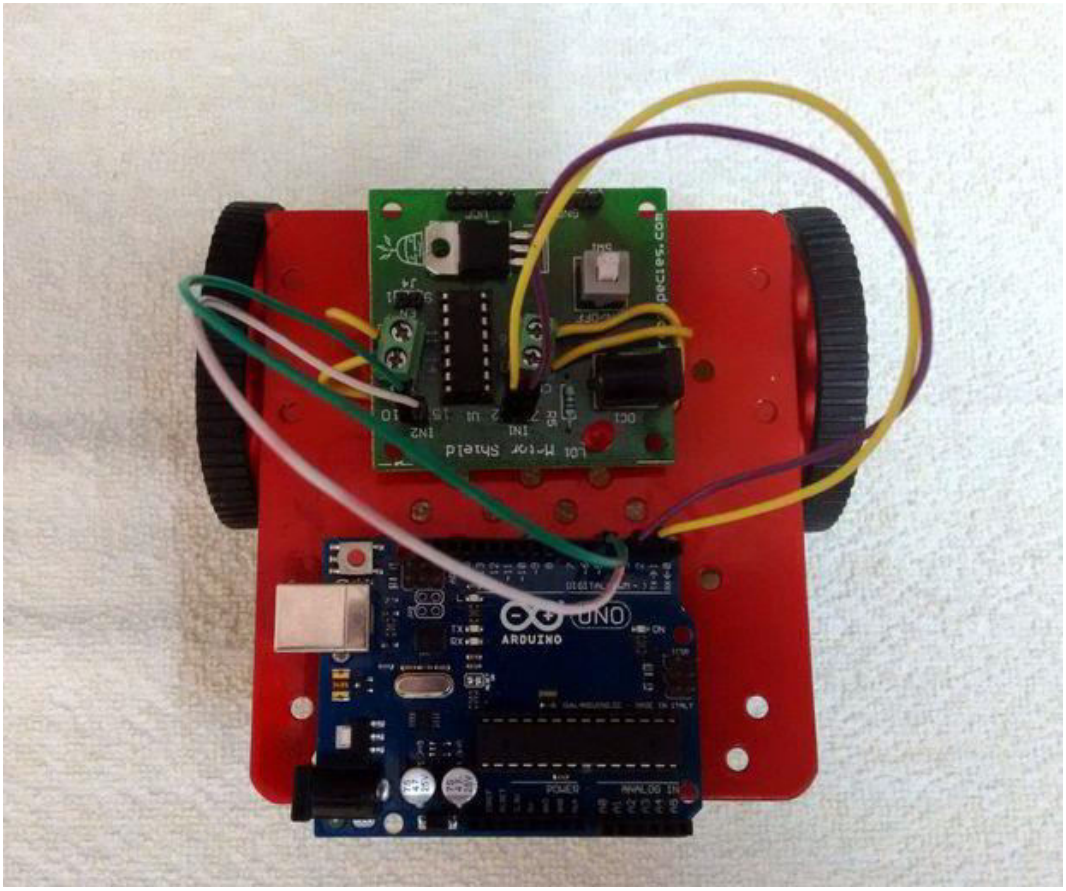

Fig 4.3.2

Step 3:

UPLOADING PROGRAM:

Interface your Arduino with Arduino IDE software and upload the below program.

```
#define echopin 8 // echo pin
#define trigpin 9 // Trigger pin
int maximumRange = 30;
long duration, distance;

void setup()
    {
          Serial.begin (9600);
         pinMode (trigpin, OUTPUT);
         pinMode (echopin, INPUT );
         pinMode (2, OUTPUT);
         pinMode (3, OUTPUT);
         pinMode (4, OUTPUT);
         pinMode (5, OUTPUT);
     }

void loop ()
{

    {
         digitalWrite(trigpin,LOW);
         delayMicroseconds(2);
         digitalWrite(trigpin,HIGH);
         delayMicroseconds(10);
         duration=pulseIn (echopin,HIGH);
         distance= duration/58.2;
         delay (50);
         Serial.println(distance);
     }

 if (distance >= 30 )
    {
         digitalWrite(2,HIGH);
         digitalWrite(3,LOW);
         digitalWrite(4,HIGH);
         digitalWrite(5,LOW);
         delay (200);
     }

 else if (distance >=15 && distance <= 25)
    {
          digitalWrite (2,HIGH);
         digitalWrite (3,LOW);
         digitalWrite (4,LOW);
         digitalWrite (5,LOW);
         delay (1000);
```

```
    }

else if (distance < 15)
   {
        digitalWrite (2, LOW);
        digitalWrite (3, HIGH);
        digitalWrite (4, LOW);
        digitalWrite (5, HIGH);
        delay (1000);
        digitalWrite (2,LOW);
        digitalWrite (3,LOW);
        digitalWrite (4,HIGH);
        digitalWrite (5,LOW);
        delay (1000);
    }
```

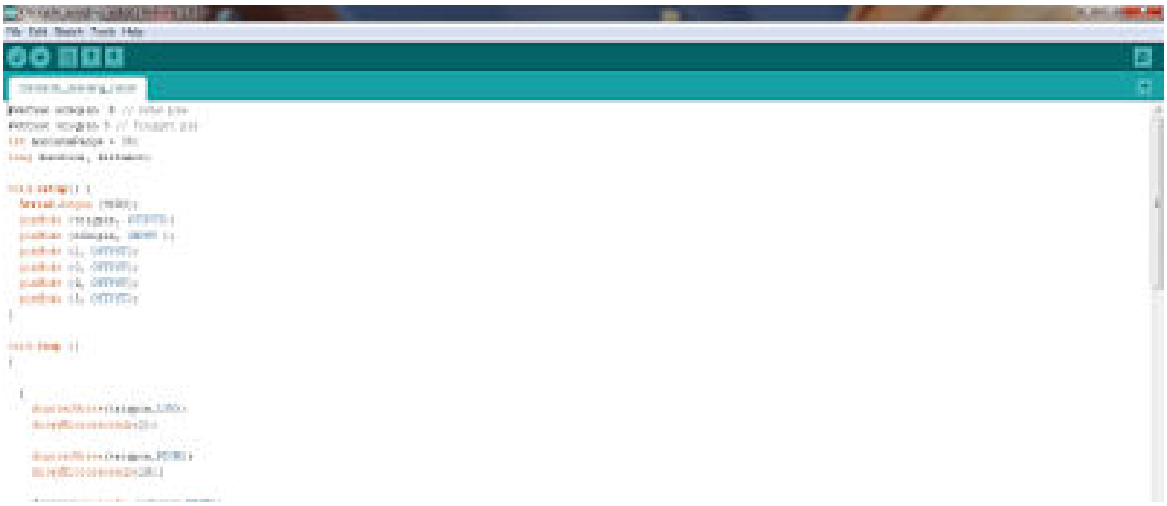

Fig 4.3.3

Step 4:

Now give the power supply to your Arduino and motor driver. Finally your robot is ready to function autonomously, avoiding obstacles and being able to find its own path.

SUMMARY

In this project we begin by proposing an algorithm to find a path of a robot to the target avoiding any obstacles and solving the local minimum problems that may be faced.

The idea of the algorithm depends on summing the potential of the obstacles and the target then finding the path at the points where the potential is minimum.

The results of the algorithm were very successful.

The disadvantage of this algorithm is the robot must know its location and the target location beside the obstacles to get the path.

In the latter half of our project, we have successfully developed an autonomous robot which is capable of finding its own path and avoiding collision with obstacles with the help of ultrasonic sensors.

The ultrasonic sensor enables the robot to virtually see and recognize object, avoid obstacles, measure distance. A UNO of Arduino family is used to achieve the desired operation.

REFERENCES

1. www.wseas.us/journals/saed/saed-45.pdf
2. Matlab Help Center
3. http://www.rakeshmondal.info/L293D-Motor-Driver
4. http://electronictechwizards.blogspot.in/2016/03/obstacle-avoiding-robot-using-arduino.html
5. https://www.elprocus.com/different-types-of-autonomous-robots-and-real-time-applications/
6. https://www.researchgate.net/post/What_is_an_autonomous_robot