PATH PLANNING FOR MOBILE ROBOTICS

Project report

Name	Ahtisham Ahmed , Furqan Kakar
Registration Number	201108 ,201153
Class	BEMTS-V(B)
Instructor's Name	Ma'am Anum Maqbool
Session	F20

Executive summary:

We made a robot which follow the shortest optimal path to reach its destination. We use random particle optimization technique for this purpose. Random particle optimization algorithm for local path planning problem of mobile robots in dynamic and unknown environments is proposed. The new algorithm inspired from bacterial foraging technique is based on particles which are randomly distributed around a robot. These particles search the optimal path toward the target position while avoiding the moving obstacles by getting help from the robot's sensors. The criterion of optimal path selection relies on the particles distance to target and Gaussian cost function assign to detected obstacles. Then, a high level decision making strategy will decide to select best mobile robot path among the proceeded particles, and finally a low level decision control provides a control signal for control of considered holonomic mobile robot.

Introduction:

Application of mobile robots in industries, military activities, and indoor usages has been remarkably growing recently. In addition, in some missions such as hazardous or hardly

accessible territorial, where humans could be in danger, mobile robots can carry out tasks such as nuclear plants, chemical handling, and rescue operations. One of the most challenging subjects in mobile robot's research is path planning. The path planning problem is usually defined as follows. Given a robot and a description of an environment, plan a path between two specific locations. The path must be collision-free (feasible) and satisfy

certain optimization criteria minimum traveling distance

Methodology:

First of all we discuss different type of scenario to get optimal solution. At the end we decided to solve problem by rpo .To deal with this we discuss at paper how to deal with different scenario. What difficulties we can face. Then we do code at MATLAB to implement knowledge of RPO in coding.

MATLAB:

Following is the code we done at MATLAB

```
clear all
clc

%Step-1 ( The X,Y position of Robot(R), Obstacle(O) and
Target(T)
Xr=4                                    %x-Cordinate of Robot
Yr=3                                    %y-Cordinate of Robot
Xo=4.5                                  %x-Cordinate of
Obstacle
Yo=4                                    %y-Cordinate of
Obstacle
```

```
Xt=5                                        %x-Cordinate of Target
Yt=5                                        %x-Cordinate of Target
Ao=1                                        %Alpha knot shows
height of Obstacle
At=1                                        %Alphat T shows depth
of Target
Mo=4                                        %Meu knot shows lenght
of Obstacle
Mt=4                                        %Meu T shows lenght of
Target

Robot=[Xr,Yr]
plot([Xr Xt],[Yr Yt],'r')                       %Plot of Robot
hold on

Obstacle=[Xo,Yo]                        %Plot of Obstacle
hold on

Target=[Xt,Yt]                          %Plot of Target

%Step-2 ( Stepsize or Radius )
NPTS=100                                  %Number of Artificial
points
DRO=norm(Robot-Obstacle)                   %Distance between Robot
and Obstacle OR DRO=sqrt((Xo-Xr)^2+(Yo-Yr)^2)
Ct=0.4*DRO                                  %Radius
viscircles([Xr,Yr],Ct)


%Step-3 ( No of Artificial Points and StepDegree )
StpDeg=360/NPTS                               %StepDegree

%step-4 ( Jobstacle , JTarget and Jtotal of Original Points )
Jobs=Ao*exp(-Mo*((Xr-Xo)^2+(Yr-Yo)^2))    %Value of J obstacle
Jtar=-Ao*exp(-Mo*((Xr-Xt)^2+(Yr-Yt)^2))   %Value of J target
Jtotal=Jobs+Jtar                            %Value of Jtotal

%Step-5 ( Distance from Robot to Target )
DRT=sqrt((Xt-Xr)^2+(Yt-Yr)^2)             %Distance btw Robot
and Target

index=0
file=fopen('LAB-NO-4_6 PathPlanning.txt','w')
theta(1)=0
fprintf(file,'AP\t\t\tx\t\t\t\ty\t\t\tJobsctc\t\t\tjgoalc\t\t\
tjtotalc\t\t\terrorj\t\t\terrorD')
for AP=1:NPTS

%Step-7 ( X , Y positions for Artificial Points Generated )
X(AP)=Xr+Ct*cos(pi*theta(AP)/180)
```

```
Y(AP)=Yr+Ct*sin(pi*theta(AP)/180)
theta(AP+1)=theta(AP)+StpDeg

%plot([X(AP) Xt],[Y(AP) Yt],'g ')

%cost function for AP
Jobs(AP)=Ao*exp(-Mo*((X(AP)-Xo(1))^2+(Y(AP)-Yo(1))^2))
Jtar(AP)=-Ao*exp(-Mo*((X(AP)-Xt)^2+(Y(AP)-Yt)^2))
Jtot(AP)=Jobs(AP)+Jtar(AP)


%Step-8 ( Error of cost function {errJ} )
errJ(AP)=Jtot(AP)-Jtotal


%Step-9 ( distance of Artificial Points to Goal )
DRTt(AP)=sqrt((X(AP)-Xt)^2+(Y(AP)-Yt)^2)

%Step-10 ( Error in Distance to Goal {errD} )
errD(AP)=DRTt(AP)-DRT
fprintf(file,'\n%d\t\t%f\t\t%f\t\t%f\t\t%d\t\t%f\t\t%f\t\t%f\t
\t',AP,X(AP),Y(AP),Jobs(AP),Jtar(AP),Jtot(AP),errJ(AP),errD(AP
))

    if(errJ(AP)<0 && errD(AP)<0)
        fprintf(file,'number found')

fprintf(file,'\t%d\t\t%f\t%f\t%f',(AP+1),theta(AP),X(AP),Y(AP)
)


     if (index==0)
           index=AP
     end

     elseif (index~=0)

     if errD(index) < errD(AP)
           index=index

     elseif errD(AP) < errD(index)
               index=AP
                plot([X(AP) Xt],[Y(AP) Yt],'g ')
     end


    else
        disp(AP)

   end
```

```
end
```

Simulation:

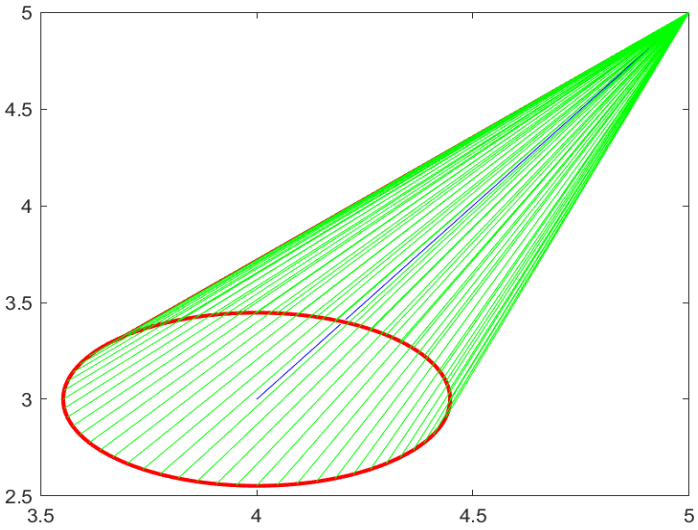


figure 1

we use 60 number of artificial points.that is the distance of target to all artificial points . All the points have their feasible way to reach destination but we need best and optimal path that should also be feasible.

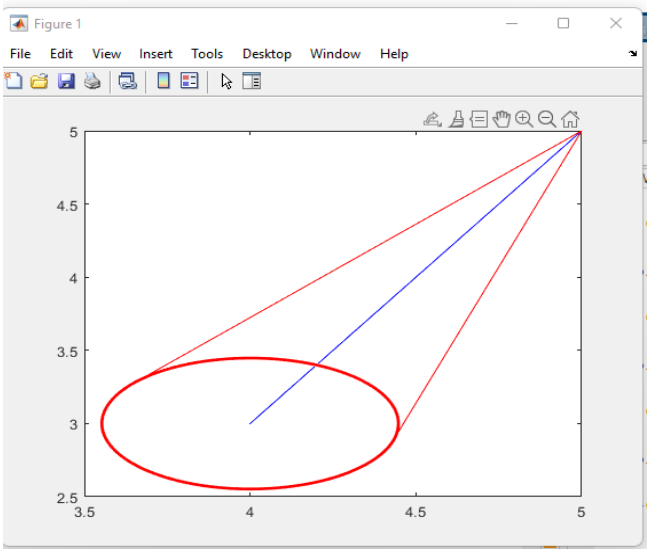


figure 2

At the next step we get 2 feasible and optimal path. Let first discuss why only these two path selected not others. The reason is the following criteria

the Robot will decide to move to the Point at which both

errD < 0 (-ve)

errJ < 0 (-ve)

And here as both errD and errJ are less then 0 the robot will move to the point at which errD is at a greater distance from 0 to the Negative side.

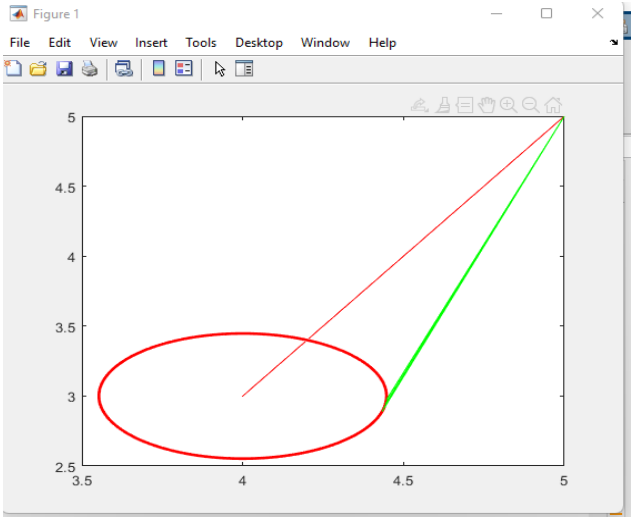


figure 3

Now among both this green one is most optimal path to get optimal and feasible path for mobile robot. Which have greater error should be selected.

Arduino IDE:

To implement this RPO algorithm in hardware we convert matlab code in c++ .for Arduino ide.Deploy code in Arduino uno microcontroller to travel robot in short optimal feasible path.

Following is the code of RPO in Arduino IDE:

```
double Xr,Yr,Xo,Yo,Xt,Yt,X_ib,Y_ib ; // coordinates of robot and obstacle and
target
double Jobs,Jtar,Jtotal,Jobs_AP,Jtar_AP,Jtot_AP ; // cost function parameters
double DRT,DRT_AP, DRO;                           //distances
double errJ[10],errD[10],F_errD[10],F_errj[10];   //errors
double Ao,At,Mo,Mt;
double theta_r,Theta[10];
double Ct,NPTS,StpDeg;
int x=1;
int index=0;

void setup() {
  // put your setup code here, to run once:
Serial.begin(9600);

}

void loop() {
```

```
  // put your main code here, to run repeatedly:
RPO_CODE();
}

double RPO_CODE()
{
  Xr = 4;
  Yr = 3;
  Xo = 4.5;
  Yo = 4;
  Xt = 5;
  Yt = 5;
  Ao = 1;
  Mo = 4;
  At = 1;
  Mt = 4;
  NPTS = 6;
  StpDeg=360/NPTS ;
  Jobs=Ao*exp(-Mo*(sqrt(pow((Xr-Xo),2)+pow((Yr-Yo),2)))) ;
  Jtar=-Ao*exp(-Mo*(sqrt(pow((Xr-Xt),2)+pow((Yr-Yt),2))));
  Jtotal=Jobs+Jtar ;
  DRT=sqrt(pow((Xt-Xr),2+pow((Yt-Yr),2))) ;
  DRO=sqrt(pow((Xo-Xr),2+pow((Yo-Yr),2)));
  Ct=0.4*DRO;
    for(int i=0;i<NPTS ;i++)
        {
          Theta[0] = 0;
           X_ib=Xr+Ct*cos((3.14*theta_r)/180);
           Y_ib=Yr+Ct*sin((3.14*theta_r)/180);

           Jobs_AP=Ao*exp(-Mo*(sqrt(pow((X_ib-Xo),2)+pow((Y_ib-Yo),2)))) ;
           Jtar_AP=-Ao*exp(-Mo*(sqrt(pow((X_ib-Xt),2)+pow((Y_ib-Yt),2))));
           Jtot_AP=Jobs_AP+Jtar_AP;
           DRT_AP=sqrt(pow((X_ib-Xt),2+pow((Y_ib-Yt),2))) ;
           errD[i]=DRT_AP-DRT ;
           errJ[i]=Jtot_AP-Jtotal ;
           Theta[1+i] += StpDeg;
           if(errJ[i]&&errD[i]<0)
           {
             F_errD[x]=errD[i];
             F_errD[x]=errJ[i];
             x+=1;

           }

        if (index==0)
           {index=i;}
```

```
        else (index!=0);
            {
                if (F_errD[index] < F_errD[x])
                  index=index;

                else (F_errD[x] < F_errD[index]);
                index=x;

            }
        }
}
```

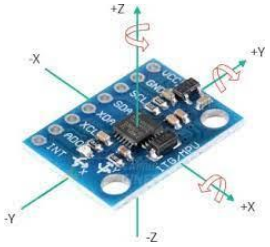
Discussion:

We are trying to implement RPO algorithm in hardware.let first discuss material required for hardware

- MPU6050
- Sonar(ultrasonic sensor)
- Arduino uno
- Dc motors
- Robot body

MPU6050:

MPU6050 is a Micro Electro-mechanical system (MEMS), it consists of three-axis accelerometer and three-axis gyroscope. It helps us to measure velocity, orientation, acceleration, displacement, and other motion like features.



MPU6050 is helping us to define coordinates for robot.

Sonar(ultrasonic sensor)

This sensor is an electronic device that will measure the distance of a target by transmitting ultrasonic sound waves, and then will convert the reflected sound into an electrical signal.

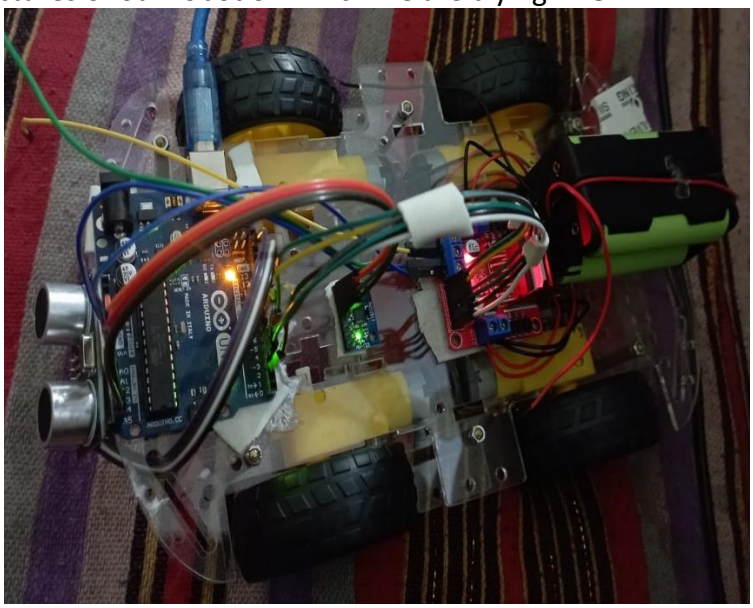
Arduino uno:

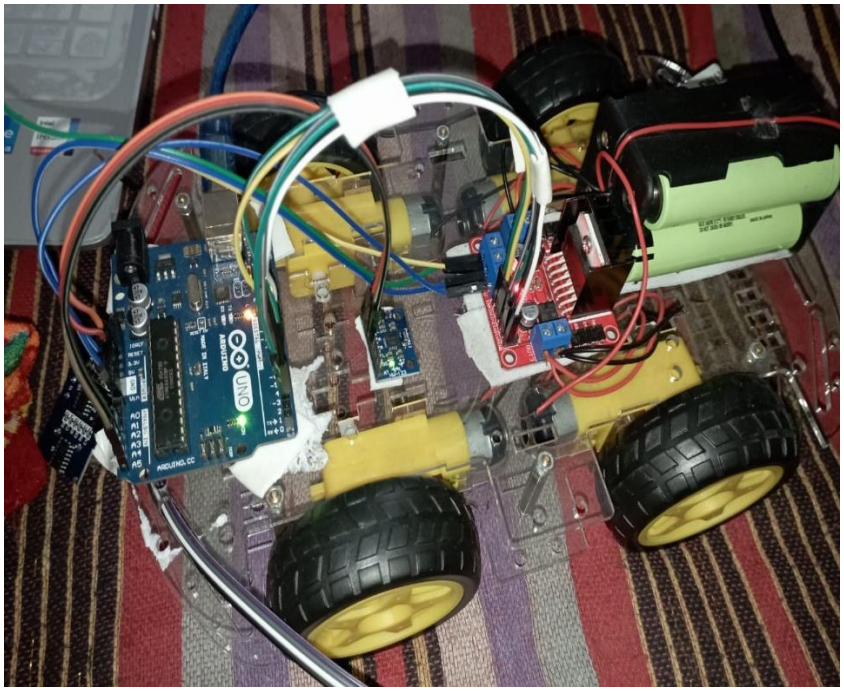
Arduino UNO is a low-cost, flexible, and easy-to-use programmable open-source microcontroller board that can be integrated into a variety of electronic projects.

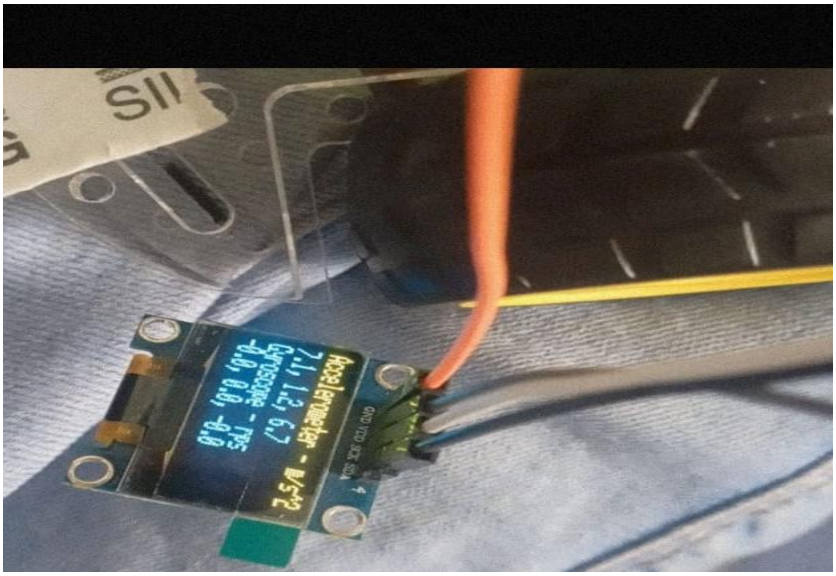
Brief explanation:

Ultrasonic sensor will detect the object .After detecting robot will go in manuver phase. Path coordinates is provided by mpu6050. After using RPO algorithm robot will reach destination by using optimal path .

Here are some pictures of our robot on which we are trying RPO.







Conclusion:

the RPO gives much optimal solution in shorter time. Furthermore, it is free from some drawbacks such as trapped states or gradient of potential functions which could provide

impulsive force to the robot.

we apply the proposed RPO algorithm on a mobile robot to find an optimal path for the robot to reach its target in the presence of obstacles. This is an online algorithm. In fact, we did not need any information and prior computation from the environment in which the robot performs its task. The only parameters that are obtained from the environment are the target position, robot position, and data from obstacle detection sensors which practically can be obtained from the sensors installed on the robot. All of

the other parameters are selected by the operator Then, the proposed algorithm is computed in the main CPU of the mobile robot, and it helps the robot to find optimal path. The number of obstacles and particles can effect on the real-time performances, but with high speed CPUs there is no problem in real-time performance. To evaluate our results, in this section three different scenarios have been set up. In the first scenario, obstacles

and target are assumed to be fixed. In the second one, the algorithm has been run on a dynamic environment with fixed target and unknown moving obstacles, and in the third one, both moving target and obstacles are evaluated.