Path planning [HW2]

Special Topics in Robotics

Jakub Tomasek

October 12, 2013

Contents

1 Problem specification

Implement a path planning algorithm to navigate robot in given environment specified by a given map (see Fig. 1.1). Make simulations. Robot starts at coordinates $\mathbf{x}_s = \begin{bmatrix} 220 \\ 140 \end{bmatrix}$; meanwhile the goal position is $\mathbf{x}_g = \begin{bmatrix} 80 \\ 40 \end{bmatrix}$.

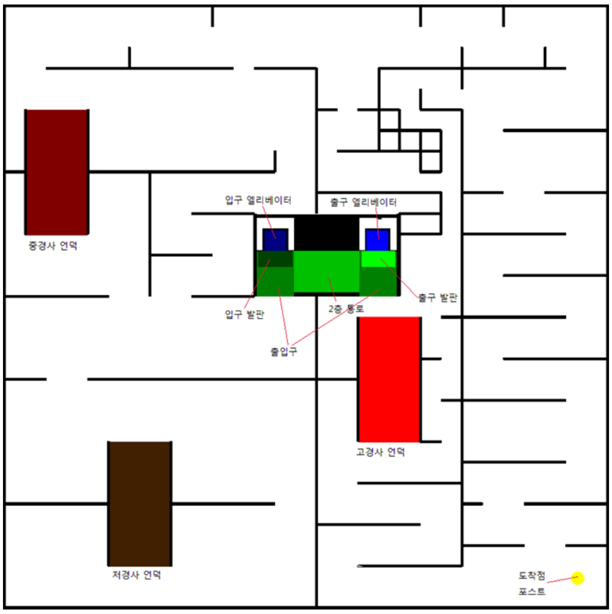


Figure 1.1: Map for path planning

2 Solution

I used the potential field method. In the potential field method, the robot is attracted towards the goal position and repelled from the obstacles he obseres. I chose parabolic potential for the attractive forces as it was suggested in lecture:

$$U_{attr} = \frac{1}{2}kd^2 \,,$$

where d is distance from the goal and k is arbitrary constant. Potential was pre-generated for each point in map. Repulsive forces were generated at each step of robot based on the distance of the obstacles from the robot:

$$\mathbf{F}_{rep} = \begin{cases} -k_r \left(\frac{1}{d_o^3} - \frac{1}{d_o^2 d_{crit}} \right) \mathbf{n} & d_o < d_{crit} \\ 0 & otherwise \end{cases} \,,$$

where d_o is distance to the obstacle, d_{crit} is a critical distance and $\mathbf{n}$ is the unit vector pointing from the robot towards the obstacle.

Obviously, for the chosen attractive potential and for the given map there are some local minima, e.g. around point $\begin{bmatrix} 90 \\ 78 \end{bmatrix}$ and the small corridor. To eliminate this problem which causes stacking the robot, I tilted the attractive potential in the critical area of the corridor.

I chose Matlab for the simulation. This piece of program pre-generates the attractive potential function.

```
%generate the attractive potential
attractive=zeros(size(map));
for x=1:size(map,1)
    for y=1:size(map,2)
    attractive(y,x)=1/2*distance([x;y],goal)^2;
    end
end

%tilt the potential in the corridor
InflexPoint=[90;78];
infl=zeros(size(map));
kinfl=1.2;
for x=(InflexPoint(1)-60):(InflexPoint(1)+300)
    for y=(InflexPoint(2)):(InflexPoint(2)+200)
    infl(y,x)=k2*x;
    end
end

attractive=attractive+infl;
%get the x and y forces
[FXa, FYa] = gradient(attractive);
```

At each step of robot, repulsive force is determined based on the distance from the obstacles:

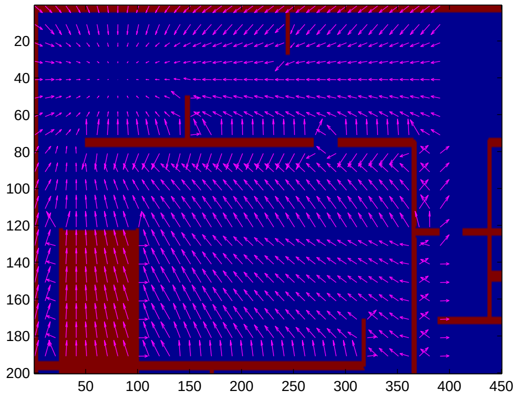
```
function force=getRepulsiveForce(surroundings,N,v,criticalDistance,krep)
    expanded=zeros(N,v);
    distances=zeros(N,1);

    %for each angle we rotate the image and get line at zero degree
    for k=0:N-1
```

```
        rotated=imrotate(surroundings,360/N*k);
        c=round(size(rotated)/2);
        expanded(k+1,:)=rotated(c(2),c(1)+1:c(1)+v);

    end

    %we determine distance at each angle
    for k=1:N
        line=expanded(k,:);
        l=1;
        while line(l)==1
            l=l+1;
            if(l==v)
                distances(k)=Inf;
                break;
            end
        end
        distances(k)=l;
    end

    %we sum forces from each angle
    F=[0;0];

    for k=1:N
        d=distances(k);
        angle=2*pi/N*(k-1);
       if(d<criticalDistance)
           F=F-krep*(1/d^3-1/(d^2*criticalDistance))*[cos(angle);sin(angle)];
       end
    end
    force=F;
end
```

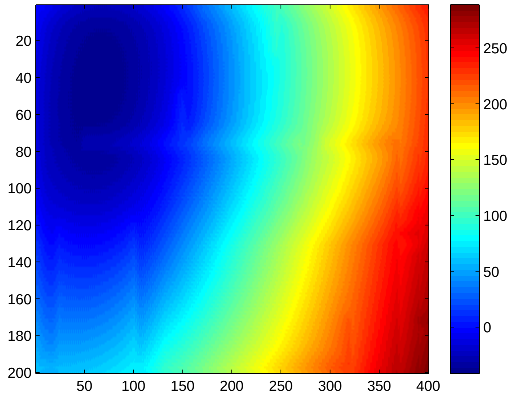
The robot is moved by the forces until it reaches the goal:

```
while ~isGoal(goal,position)

    %robot doesn't see the whole map
    surroundings=getSurroundings(map,visibility,rpos);

    %we determine distance from the obstacles and find the repulsive force
    Frep=getRepulsiveForce(surroundings,numberOfAngles,visibility,criticalDistance,krep)
    Fatt=-k*[FXa(rpos(2),rpos(1));FYa(rpos(2),rpos(1))];
    F=Frep+Fatt;
    %in case that the force is too large for our hypothetic motor :)
    F=limitSpeed(F,maxSpeed);

    position=position+2*F;
    positions=cat(2,positions,position);
    rpos=round(position);
    displayMap(map,position,positions);

    c=c+1;
    M(c)=getframe(gcf);
end
```

3 Conclusion



(a) Final force field



(b) Potential function

Figure 3.1: (a) Shape of the force field and (b) potential function in left top 400×200 area.

After tilting the potential function, the robot is able to navigate from the start coordinates to the goal as well as other initial coordinates without colliding with obstacles; see Fig. 3.2 which depicts the trajectories. Additionally, I generated animations (http://www.youtube.com/watch?v=935OUciGIkA). Important part of the task was also to identify right parameter values such as k, k_{rep} or the critical distance d_{crit}. Figures 3.1a and 3.1b show the force field and final potential. One can observe the slight tilt in the area of the hallway.

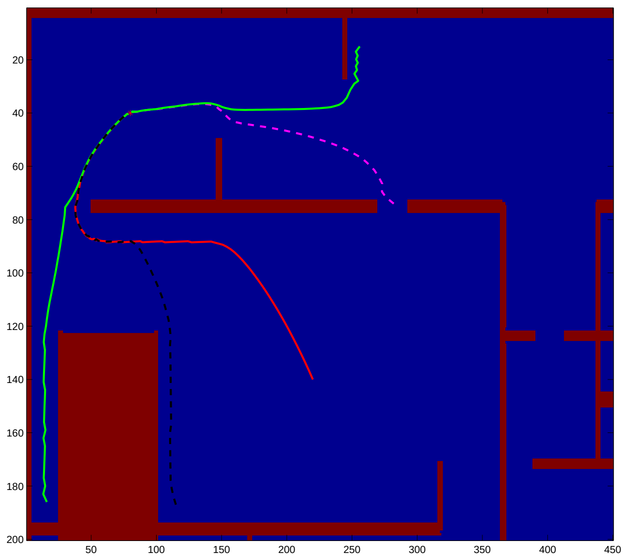


Figure 3.2: Sample trajectories for several different initial coordinates. The goal is highlighted by red start.