IMPLEMENTATION OF PURE PURSUIT AND VECTOR FIELD HISTOGRAM IN MATLAB

V. ADARSH (21307R001)
PRADEEP CHUDASAMA (213070111)

1 INTRODUCTION

The purpose of this experiment was to make the robot follow a path predefined by way-points, in an environment filled with obstacles. The robot must follow the path while detecting and avoiding any obstacles in it's way and reach the goal point. To make the robot track the predefined path Pure Pursuit algorithm was implemented and to help it navigate through obstacles by avoiding collision with them, Vector field histogram algorithm was implemented.

2 PURE PURSUIT

Pure Pursuit is basically a path tracking/following algorithm that works by finding a target point on path "one lookahead distance" away and the curvature needed by robot to reach that point. Pure pursuit takes robot's current position, orientation and waypoints defining the path as input and outputs angular velocity that is needed to go along a curve to reach the target point.

To find the required radius to move along the arc, consider figure-1. Here target point TP is shown in robot's frame. The path or curvature needed for robot to go to TP is defined by the arc between O and TP. This arc is a part of a circle with radius "r" centered at "C" as shown. The chord of this arc is of length "l" the lookahead distance. From the figure-1,
$x^2 + y^2 = l^2$
$x + d = r$
$d^2 + y^2 = r^2$
From the above 3 equations we can simply find :
$r = \frac{l^2}{2x}$

To implement this in matlab, following steps were followed:

1. Closest waypoint from the robot's current location was found.

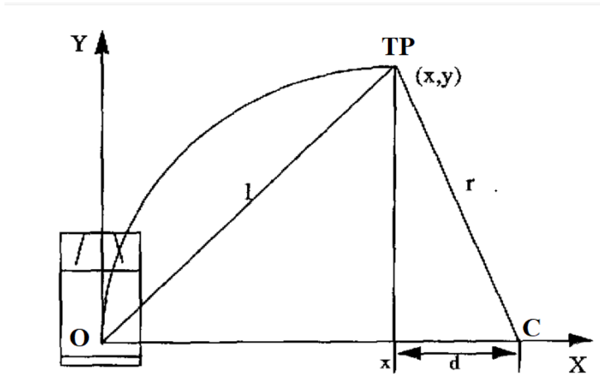

Figure 1: calculation of radius of curvature

2. The waypoint that comes after the closest waypoint along the path *next_waypoint* was then taken and the line segment between these two was considered.

```
wpx=waypoints(:,1);
    wpy=waypoints(:,2);
    dist_from_wp=sqrt((pose(1,1)-wpx).^2 +(pose(2,1)-wpy).^2);
    closest_wp_index=find(dist_from_wp==min(dist_from_wp));
    closest_wp=waypoints(closest_wp_index(1,1),:);
    s=size(waypoints);
    next_wp=waypoints(closest_wp_index(1,1),:);
    if(closest_wp_index(1,1)~=s(1))

         next_wp=waypoints(closest_wp_index(1,1)+1,:);
    end
```

3. Next, a circle centered at robot's current location with a radius equal to "LOOKAHEAD DISTANCE" was considered and intersection between this circle and the line segment defined in previous step was found.

4. In case of a single intersection point, that was chosen as target point. If there were 2 intersection points then the one closer to the *next_waypoint* was taken as target point.

5. There can be cases where no valid intersection points exist. In such a case a line segment was considered between the robot's location and it's closest waypoint or the last target point which was lying along the path, whichever is closer. In this case there will always be one intersection point, which will then be taken as target point

```
E=transpose(closest_wp);
 TP=E
 L=transpose(next_wp);
 last_TP_on_path=E;
 C=pose(1:2,1);
 d=L-E;
 f=E-C;
 l=0.35; %lookahead distance
 a=norm(d)^2;
 b=2*dot(f,d);
 c=norm(f)^2-l^2;
 D=b^2-(4*a*c);
 if (D<0)
      flag=1;
 end
   if (D>=0)
      D=sqrt(D);
      t1=(-b-D)/(2*a);
      t2=(-b+D)/(2*a);
      if( (t1<0 || t1>1) && (t2<0 || t2>1))
           flag=1;
           end
 end
 D=t2;
 if(flag==0)
      if(D==0)
           TP=E+t1*d;
           end
      if (D>0)
           if( (t1<0 || t1>1) && (t2>=0 && t2<=1))
                TP=E+t2*d;
                end
           if((t1>=0 && t1<=1) && (t2<0 || t2>1))
                TP=E+t1*d;
           end
           if((t1>=0 && t1<=1) && (t2>=0 && t2<=1))
                TP1=E+t1*d;
                TP2=E+t2*d;
                TPs=[transpose(TP1);transpose(TP2)];
                TPx=TPs(:,1);
                TPy=TPs(:,2);
                dist_from_L=sqrt((L(1,1)-TPx).^2 +(L(2,1)-TPy).^2);
      closest_TP_index=find(dist_from_L==min(dist_from_L));
                TP=transpose(TPs(closest_TP_index
```

```
                    (1,1),:));
                end
            end
            last_TP_on_path=TP
        end
        if(flag==1)
             norm1=norm(pose(1:2,1)-last_TP_on_path)
            norm2=norm(pose(1:2,1)-E)
            if(norm1<=norm2)
                L=last_TP_on_path;
            end
            if(norm1>norm2)
                L=E
                end
            E=pose(1:2,1);
            d=L-E;
            TP=E+l*(d)/norm(d);
        end
```

6. After finding target point, which will be in global coordinates, it was carefully converetd to robot's body frame, and then using then using pure pursuit the required radius of curvature (r) and hence the angular velocity(w) was calculated as $w = \frac{v}{r}$, where 'v' is fixed linear velocity.

```
vv=TP-pose(1:2,1);
        phi=atan2(vv(2,1),vv(1,1));
        if(phi<0)
            phi=2*pi+phi;
        end
        ang=phi-(sign(pose(3,1)))*pose(3,1);
        if(ang<0)
            ang=2*pi+ang;
            end
        if(ang>pi)
            TP_bot_x=-l*cos((pi/2)+(ang-2*pi));
        end
        if(ang<=pi)
            TP_bot_x=l*cos((pi/2)-ang);
            end
          vRef=0.75
        R=(l^2)/(2*(TP_bot_x));
        wRef=(vRef)/(R);
```

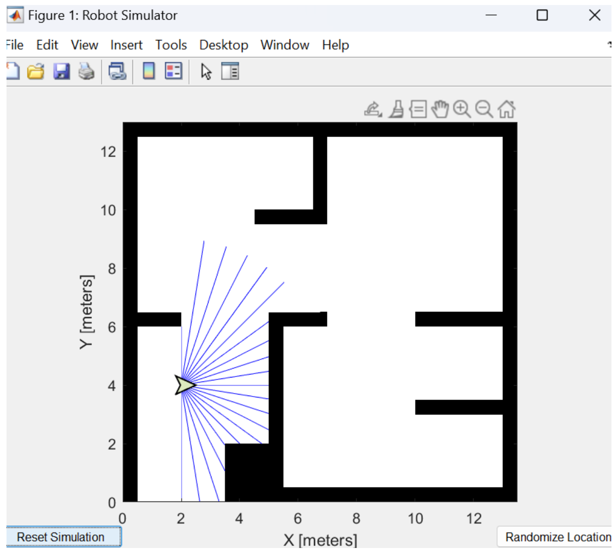

Figure 2: The robot and it's workspace

3 VECTOR FIELD HISTOGRAM

Vector field histogram or VHF is an algorithm used for obstacle avoidance in an unknown environment filled with obstacles. VFH tries to steer the robot towards the goal point while steering away from obstacle.

3.1 Certainity Grid

The first step to implement this algorithm is a certainty grid. The robot's workspace is discretized into a 2D array of square grid cells, where each grid cell holds a certainty value CV indicating the presence of an obstacle near that cell.
In our experiment, the workspace is 12.5 x 12.5 units. This was converted into a 126 x 126 grid, where each cell is 0.1x 0.1 units. Each grid cell is initially assigned a probability or CV of 0.5 indicating an equal chance of being occupied by an obstacle or free of obstacle. As the robot moves, the CVs are repeatedly updated by robot's laser scan reading. Hence this certainty grid keeps on updating with the robot's movement through the workspace. Initially at start of simulation, every grid cell has a CV of 0.5.

As shown in figure-2 the robot has 21 laser beams spanning from an angle of -pi/2 to pi/2 with 0 degrees along robot's heading. Each of these beam takes a range reading at every time step of simulation. For a given beam, using it's angle and orientation of robot and it's range reading the (x,y) position of that

object is determined, which can be then used to determine which grid cell that point belongs to. Once the grid cell is determined, it's confidence value CV is increased since we know there is an obstacle present near that cell. Further, the CV of all the cells lying along that beam up-to the cell that contains the object is decreased since the fact that beam passes through those cells without interruption implies those cells are not occupied by any object.
The confidence value or certainty value CV of each cell is updated using the procedure described in [1]. To briefly describe the procedure, denote 'm' as a binary random variable for a given grid cell such that it takes value 1 if obstacle present and 0 otherwise. Denote $x_{1:t}$ as the position of robot from till t time steps and $z_{1:t}$ as the laser measurements till t time steps. We are interested in the probability that m=1 given $x_{1:t}$ and $z_{1:t}$, i.e how does the probability that m=1 changes after every new measurement at next time step. Following the derivation described in [1] the final update rule is as follows:
$logodds(m = 1|z_{1:t}) = logodds(m = 1|z_t) + logodds(m = 1|z_{1:t-1})$
Here,
$logodds(x) = log(P(x)/(1 - P(x)))$
And $P(m = 1|z_t)$ is the probability that an obstacle is present given the measurement at time step 't'. If the grid cell lies at the end of laser beam from where the beam reflects off, this term takes an value of 0.8 and if the cell lies along the beam it takes the value of 0.2. The ceratinity value CV for each cell can then be obtained as:
$P(m = 1|z_{1:t}) = \frac{exp(logodds(m=1|z_{1:t}))}{(1+exp(logodds(m=1|z_{1:t})))}$
Figure-3 shows the visualization of occupancy grid obtained during one of the simulations, where robot moves in a predefined path and uses it's laser readings to find and repeatedly update the CVs of grid cells using the procedure described above. The grey cells are the ones that robot's laser has not reached hence a CV of 0.5, whereas lighter cells denote very less CV of presence of obstacle and darker cells indicate a strong presence of obstacle nearby. One can compare it with the actual map and see how it has successfully detected the edges of obstacles as well as free space in the map.

3.2 Polar Histogram

Here, every grid cell is mapped to a sector around the robot to construct a polar histogram which represents obstacle density in different directions around the robot. Once the certainty grid is created we have the CVs of the grid cells for a given time step. We consider an active window of $w_s x w_s$ cells centered around robot's center. We consider all the cells that fall in this active window and map them to different sectors around robot. For a cell (i,j), we construct an obstacle vector whose direction and magnitude are given by:
$\beta_{ij} = tan^{-1}(\frac{y_j - y}{x_i - x})$

$m_{ij} = c_{ij}^2(a - bd_{ij})$

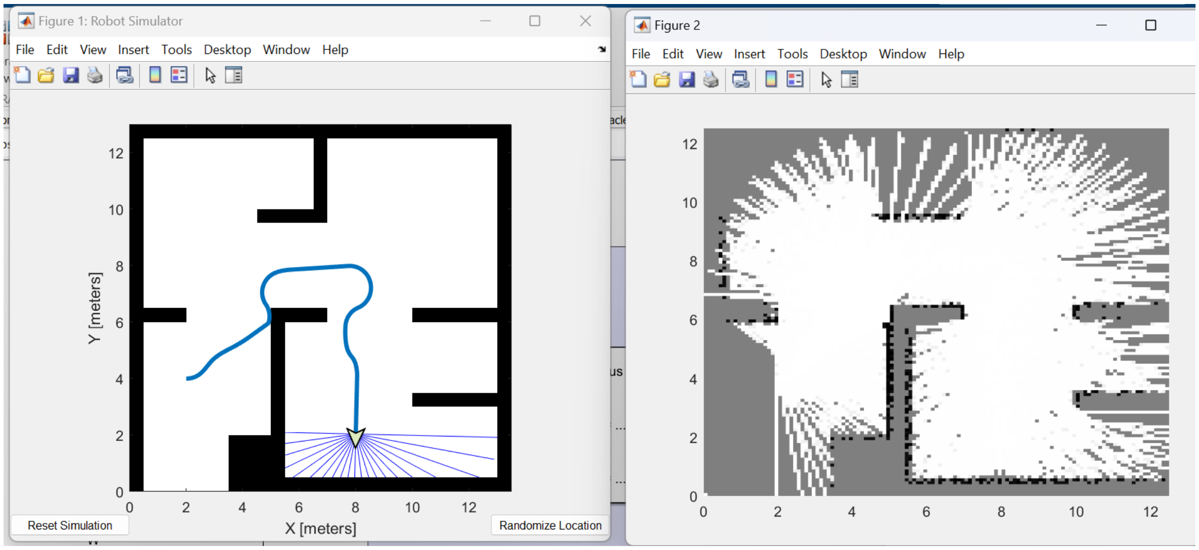

Figure 3: Occupancy map visulaization

where
x_i, y_j = center coordinates of (i,j) cell
x, y = center of robot
c_{ij} = CV of the cell
d_{ij} = distance between robot and cell
a, b = positive constants defined by user
Now we consider an angular resolution α which gives us $360/alpha$ sectors around the robot. Each grid cell is mapped to it's corresponding kth sector using :
$k = floor(\beta_{ij}/\alpha) + `1$
Once every cell in active window is mapped to it's corresponding sector, the next step is to calculate the polar obstacle density h_k for each sector which is simply given by summation of the magnitude of obstacle vector of all cells in that sector:
$h_k = \sum m_{ij}$

A further smoothing is applied as discussed in [2]
$h_k = \frac{h_{k-l} + 2h_{k-l+1} + ... + (l+1)h_k + ...h_{k+l}}{2l+1}$
It's just a weighted average over a range of sectors determined by 'l'.

3.3 Steering angle

The next step is to find which direction the robot has to steer based on the polar obstacle density around it. For this we identify groups of consecutive sectors around the robot such the each sector in a group has polar obstacle density below a predefined histogram threshold. Such groups are defined as candidate valleys in [2]. Usually more than one such candidate valley exists around the robot where each of them indicates that obstacle density is less in that direction, where we can identify the direction by the sectors included in that candidate valley.

Pure pursuit algorithm calculates the target point and angle between the robot's heading and target point every time step and this target direction is used as an input to VFH algorithm. Using this target direction we can determine the sector this belongs to by the same logic as we determined every grid cell's sector. Denote this sector as k_{targ}. For every candidate valley,

1. Find the sector in it closest to k_{targ} and denote it as $k_n. This$k$_n = k_{targ}$ when k_{targ} is included in the candidate valley. Else, it is either of the two extremes of candidate valley.
2. define $k_f = k_n + s_{max}$ if valley width is more than s_{max} else $k_f =$ other exterme of valley.
3. $\theta_{steer} = (k_n + k_f)/2$
4. find how far away this θ_{steer} is from target direction.

The final steering angle is the one that is closest to the target direction. This ensures that robot steers away from obstacle while trying to be as aligned to target direction as possible. This trade-off often determines the performance of robot when an obstacle lies directly in front of robot and it needs to sharply deviate from path.

3.4 Matlab Code

```
function [SteerDir,op,op1]= fcn(Ranges,Angles,pose,rmax,
    TargetDir)
global log_odds;
op1=zeros(1,1);
op=zeros(1,1);
global theta_steer_prev;
res=0.1;
w_s=25;
coder.varsize('Start');
```

```
coder.varsize('End');
coder.varsize('Start');
coder.varsize('temp4');
xx=1;
p=0;
yy=1;
xend=1;
yend=1;
TargetDir=robotics.internal.wrapToPi(TargetDir);
safety=2.0;
for k=1:numel(Ranges)
    dist=Ranges(k);
     if isnan(Ranges(k))
         dist=double(rmax);
     end
     theta=Angles(k);
     if(pose(3,1)>=0)
     if(theta<=0)
         theta=pose(3,1)-abs(theta);
     end
     if(theta>0)
         theta=pose(3,1)+theta;
     end
     end
     if(pose(3,1)<0)
     if(theta<=0)
         theta=pose(3,1)+(theta);
     end
     if(theta>0)
         theta=-abs(pose(3,1))+theta;
     end
     end
     ang=theta;
     if(theta>=0 && theta<pi/2)
             xx=pose(1,1):(res/2)*cos(ang):min(pose(1,1)+(dist-2*res)*cos(ang),12.5);
             yy=pose(2,1):(res/2)*sin(ang):min(pose(2,1)+(dist-2*res)*sin(ang),12.5);

             if(ang==0)
                 yy=ones(1,numel(xx))*pose(2,1);
             end
             xx=floor(xx/res)+1.0;
             yy=floor(yy/res)+1.0;
             xend=min(pose(1,1)+(dist)*cos(ang),12.5);
             yend=min(pose(2,1)+(dist)*sin(ang),12.5);
```

```
                xend=floor(xend/res)+1.0;
                yend=floor(yend/res)+1.0;

        end
        if(theta<=pi && theta>=pi/2)
                 ang=pi-theta;
                xx=pose(1,1):-(res/2)*cos(ang):max(0,pose(1,1)-(dist-2*res)*cos(ang));
                yy=pose(2,1):(res/2)*sin(ang):min(pose(2,1)+(dist-2*res)*sin(ang),12.5);
                if(ang==0)
                    yy=ones(1,numel(xx))*pose(2,1);
                end
                if(ang==pi/2)
                    xx=ones(1,numel(yy))*pose(1,1);
                end
                xx=floor(xx/res)+1.0;
                yy=floor(yy/res)+1.0;
                xend=max(pose(1,1)-(dist)*cos(ang),0);
                yend=min(pose(2,1)+(dist)*sin(ang),12.5);
                xend=floor(xend/res)+1.0;
                yend=floor(yend/res)+1.0;
        end
             if( theta>=-pi && theta<=-pi/2)
                 ang=pi-abs(theta);
                xx=pose(1,1):-(res/2)*cos(ang):max(0,pose(1,1)-(dist-2*res)*cos(ang));
                yy=pose(2,1):-(res/2)*sin(ang):max(0,pose(2,1)-(dist-2*res)*sin(ang));
                if(ang==0)
                    yy=ones(1,numel(xx))*pose(2,1);
                end
                if(ang==pi/2)
                    xx=ones(1,numel(yy))*pose(1,1);
                end
                xx=floor(xx/res)+1.0;
                yy=floor(yy/res)+1.0;
                xend=max(pose(1,1)-(dist)*cos(ang),0);
                yend=max(pose(2,1)-(dist)*sin(ang),0);
                xend=floor(xend/res)+1.0;
                yend=floor(yend/res)+1.0;
             end
             if( theta<0 && theta>-pi/2)
                 ang=abs(theta);
                xx=pose(1,1):(res/2)*cos(ang):min(pose(1,1)+(dist-2*res)*cos(ang),12.5);
```

```
                yy=pose(2,1):-(res/2)*sin(ang):max(0,pose(2,1)-(dist-2*res)*sin(ang));
                xx=floor(xx/res)+1.0;
                yy=floor(yy/res)+1.0;
                xend=min(pose(1,1)+(dist)*cos(ang),12.5);
                yend=max(pose(2,1)-(dist)*sin(ang),0);
                xend=floor(xend/res)+1.0;
                yend=floor(yend/res)+1.0;
            end


        for i=1:min(numel(xx),numel(yy))
            ux=xx;
            uy=yy;
            if(yy(i)>size(log_odds,1))
                uy(i)=size(log_odds,1);
                yy=uy;
            end
            if(xx(i)>size(log_odds,1))
                ux(i)=size(log_odds,1);
                xx=ux;
            end
            log_odds(yy(i),xx(i))=log_odds(yy(i),xx(i))+log(0.2/0.8);
        end
        if(~isnan(Ranges(k)))
            if(yend>size(log_odds,1))
                yend=size(log_odds,1);
            end
            if(xend>size(log_odds,1))
                xend=size(log_odds,1);
            end
            log_odds(yend,xend)=log_odds(yend,xend)+log(0.8/0.2);
        end
end
prob=exp(log_odds)./(1+exp(log_odds));
i1_grid=min(floor(pose(2,1)/res)+1+(w_s-1)/2,(12.5/res)+1);
j1_grid=max(floor(pose(1,1)/res)+1-(w_s-1)/2,1);
i2_grid=max(floor(pose(2,1)/res)+1-(w_s-1)/2,1);
j2_grid=min(floor(pose(1,1)/res)+1+(w_s-1)/2,(12.5/res)+1);
i1=i2_grid;
i2=i1_grid;
j1=j1_grid;
```

```
j2=j2_grid;
alpha=5*pi/180;
b1=5;
a1=b1*(2^0.5)*(w_s-1)/2;
%C_active_prob=prob(i1:i2,j1:j2);
sector_POD=zeros(1,(2*pi/alpha));
for i=i1:i2
    for j =j1:j2
        xi=(j*res)+res/2;
        yj=(i*res)+res/2;
        cell_pos=[xi;yj];
        beta_ij=atan2(cell_pos(2,1)-pose(2,1),cell_pos(1,1)-pose(1,1));
        if beta_ij<0
            beta_ij=2*pi+beta_ij;
        end
        sector=floor(beta_ij/alpha)+1;
        d=sqrt((cell_pos(1,1)-pose(1,1))^2 + (cell_pos(2,1)-pose(2,1))^2);
        if((prob(i,j)>=0.5))
        d=max(sqrt((cell_pos(1,1)-pose(1,1))^2 + (cell_pos(2,1)-pose(2,1))^2)-safety,0);
        end
        sector_POD(1,sector)=sector_POD(1,sector)+((prob(i,j)^2)*(a1-b1*d));
    end
end
l=5;
smoothened_sector_POD=zeros(1,(2*pi/alpha));
temp4=[0];
for k =1:(2*pi/alpha)
    k1=k-l;
    if(k1<1)
        k1=(2*pi/alpha)+k1;
    end
    k2=k+l;
        if(k2>(2*pi/alpha))
            k2=k2-(2*pi/alpha);
        end
     temp1=1:l+1;
     temp2=fliplr(temp1);
     temp3=cat(2,temp1,temp2(1,2:end));
     disp(k1)
     disp(k2)
     if(k1<k2)
        temp4=sector_POD(1,k1:k2).*temp3;
```

```
        end
        if(k1>k2)

            temp4=cat(2,sector_POD(1,k1:end),sector_POD(1,1:k2)).*temp3;

        end
        disp(temp4)
        smoothened_sector_POD(1,k)=sum(temp4)/(2*l+1);
end

Start=[0];
End=[0];
flag=1;

opp=[min(smoothened_sector_POD);max(smoothened_sector_POD)];
h_t=1; %Histogram Threshold
 for i=1:size(smoothened_sector_POD,2)
        if(smoothened_sector_POD(i)<h_t && flag==1)
            Start=cat(2,Start,[i]);
            flag=0;
        end
        if(smoothened_sector_POD(i)>=h_t && i>1 && flag==0)
            End=cat(2,End,[i-1]);
            flag=1;
        end
        if(i==size(smoothened_sector_POD,2) && smoothened_sector_POD(i)<h_t)
            if(size(Start,2)==2 && size(End,2)==1)
                End=cat(2,End,[i]);
                continue;
            end
            if(smoothened_sector_POD(1)<h_t)
            Start(1)=Start(end);
            Start(end)=[];
            end
            if(smoothened_sector_POD(1)>=h_t)

           End=cat(2,End,[i]);
            end
        end
 end
Start=Start(1,2:end);
End=End(1,2:end);
op1=size(Start,2);
```

```
op=size(End,2);
theta_targ=pose(3,1)+TargetDir;
if(theta_targ<0)
    theta_targ=2*pi+theta_targ;
end
k_targ=floor(theta_targ/alpha)+1;
Cost=inf;
kn=1;
kf=1;
smax=15;
a=3;
b=0.02;
c=0;
theta_steer_opt=TargetDir;
for m=1:size(Start,2)
    v_start=0;
    v_end=0;
    if(Start(m)<End(m))
        if(k_targ>=Start(m) && k_targ<=End(m))
            kn=k_targ;
            kf=kn+smax
            if(kf>2*pi/alpha)
            kf=mod(kf,2*pi/alpha);
            end
             end
        if(~(k_targ>=Start(m) && k_targ<=End(m)))
            if(Start(m)-k_targ<0)
                v_start=2*pi/alpha;
            end
            if(k_targ-End(m)<0)
                v_end=2*pi/alpha;
            end
            if(abs(Start(m)-k_targ+v_start)<=abs(k_targ-End(m)+v_end))
                kn=Start(m);
                kf=kn+smax
            if(kf>2*pi/alpha)
            kf=mod(kf,2*pi/alpha);
            end
            end
            if(abs(Start(m)-k_targ+v_start)>abs(k_targ-End(m)+v_end))
                kn=End(m);
                kf=kn-smax
            if(kf<0)
            kf=mod(kf,2*pi/alpha);
```

```
                end
                    %kf=mod(kn-smax,2*pi/alpha); %mod(-12,72)=60 as per this function!! so works
                    end

            end
        end
        if(Start(m)==End(m))
            kn=Start(m);
            kf=kn+smax
                if(kf>2*pi/alpha)
                kf=mod(kf,2*pi/alpha);
                end
            %kf=mod(kn+smax,2*pi/alpha);
            end
        if(Start(m)>End(m))
            if((k_targ>=Start(m) && k_targ<=2*pi/alpha) || (k_targ>=1 && k_targ<=End(m)))
                kn=k_targ;
                kf=kn+smax
                if(kf>2*pi/alpha)
                kf=mod(kf,2*pi/alpha);
                end
                %kf=mod(kn+smax,2*pi/alpha);
               end
            if(~((k_targ>=Start(m) && k_targ<=2*pi/alpha) || (k_targ>=1 && k_targ<=End(m))))
                 if(abs(Start(m)-k_targ)<=abs(k_targ-End(m)))
                    kn=Start(m);
                    kf=kn+smax
                if(kf>2*pi/alpha)
                kf=mod(kf,2*pi/alpha);
                end
                    %kf=mod(kn+smax,2*pi/alpha);

                end
                if(abs(Start(m)-k_targ)>abs(k_targ-End(m)))
                    kn=End(m);
                    kf=kn-smax
                if(kf<0)
                kf=mod(kf,2*pi/alpha);
                end
                    %kf=mod(kn-smax,2*pi/alpha); %mod(-12,72)=60 as per this function!! so works

                end
```

```
            end
        end
        theta_steer=(kn+kf)*alpha/2;
        if(abs(kn-kf)>smax)
            theta_steer=(((kn+kf)/2)+(pi/alpha))*alpha;
        end

        %###################################%
        p=pose(3,1);
        if(pose(3,1)<0)
            p=2*pi+pose(3,1);
        end
        theta1=abs(robotics.internal.wrapToPi(theta_steer-p))
            ;

        theta2=abs(robotics.internal.wrapToPi(theta_targ-
            theta_steer));

        theta3=abs(robotics.internal.wrapToPi(
            theta_steer_prev-theta_steer));

        if((a*theta2 +b*theta1 +c*theta3)<=Cost)
            Cost=(a*theta2 +b*theta1 +c*theta3);
            theta_steer_opt=robotics.internal.wrapToPi(
                theta_steer-p);
        end
    end
    SteerDir=theta_steer_opt;
    theta_steer_prev=theta_steer_opt;
    if(size(Start,2)==1 && size(End,2)==1)
        if(Start(1,1)==1 && End(1,1)==2*pi/alpha)
            SteerDir=TargetDir;
            theta_steer_prev=SteerDir;
        end
    end
end
```

4 SIMULATION AND RESULTS

A Simulink model was used to simulate the results. The model was connecetd to a ROS network. ROS allowed exchange of information about robot's different components such as it's current pose, laser scan readings which are essential inputs for pure pursuit and VFH. The calculated angular velocity was published to velocity topic of ROS which enabled the robot to move.

Figure-4 shows simulation results when the path is defined by [(2,4); (2,10);(10,4)].

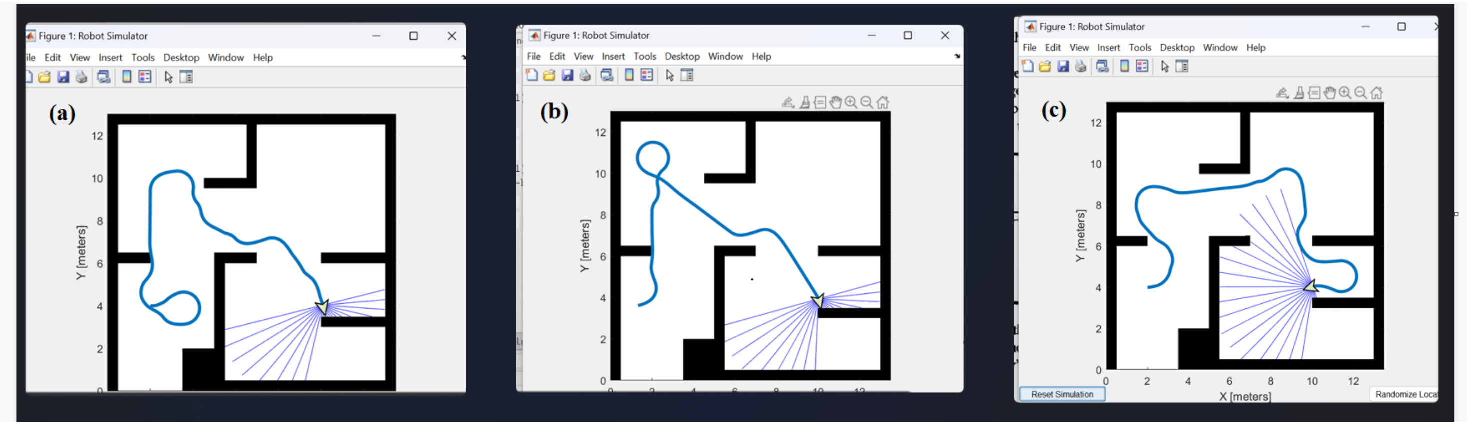

Figure 4: (a)and (b) small lookahead,l=0.35; (c) l=1.85

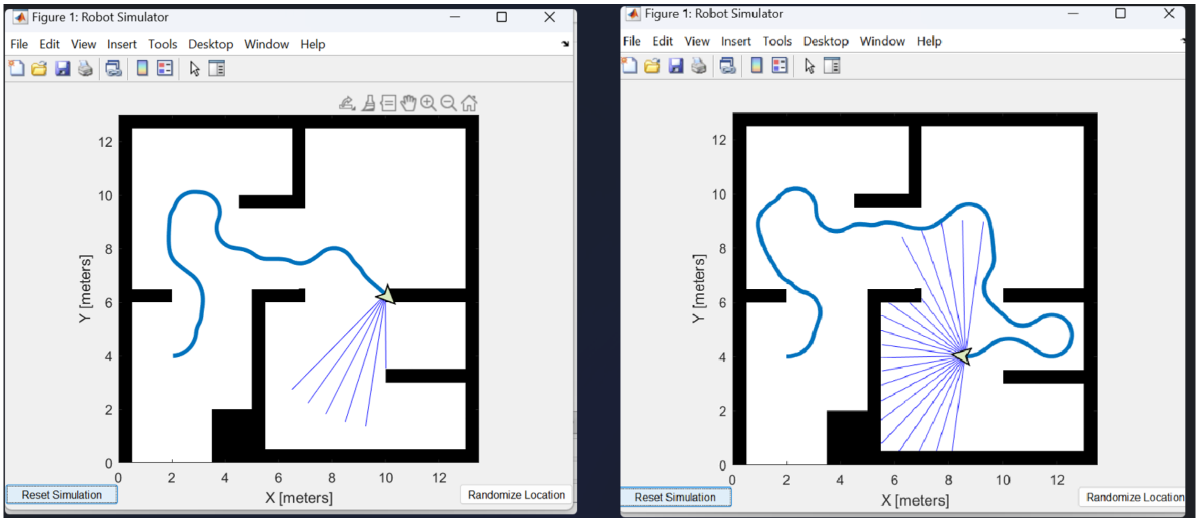

Figure 5: waypoints: [(2,4); (2,10);(10,4)]

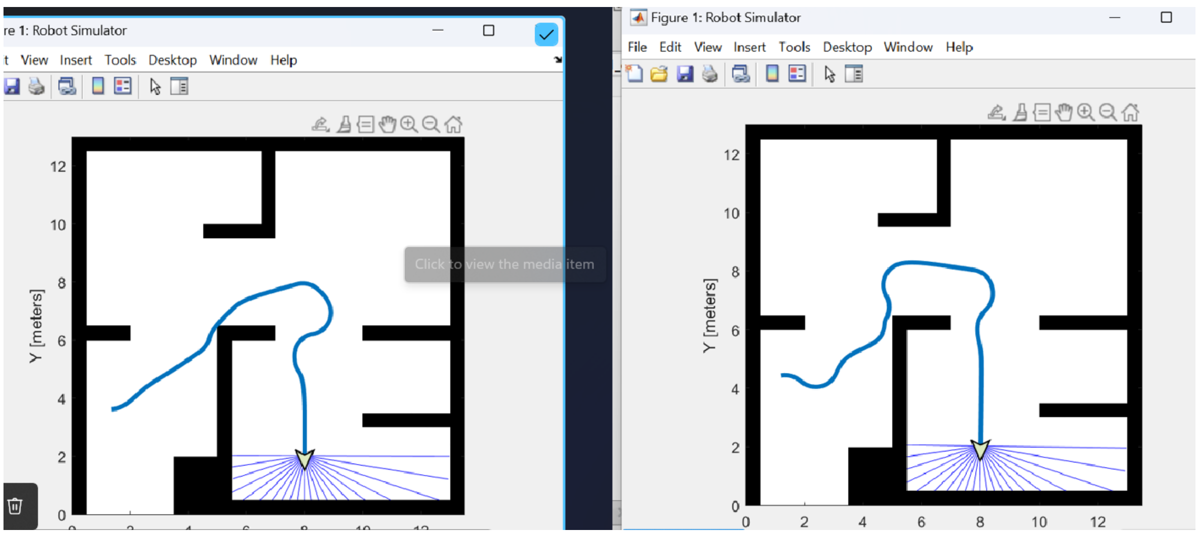

Figure 6: waypoints:[(2,4); (8,8);(8,2]

It can be seen that when lookahead distance is small like in case of (a),(b) the robot follows the path more closely as compared to the (c) where lookahead is large.
Figure 4 and 5 shows some more simulations obtained by varying active window size, s_{max}, and histogram threshold. It was observed that when robot is very closely surrounded by obstacles it shows unpredicatble behaviour and more often than not it collides with obstacle. Further active window size also plays a significant role in robot's performance. For example, a small window will make the robot closely follow the path until it comes very close to the obstacle, in which case it does not have enough space to steer away from obstacle. A large window leads to the robot trying to avoid obstacles even when it is not that near to one, which leads to oscillatory behaviour and and also sometimes robot wanders away from path. In most cases VFH is a tradeoff between not deviating much from target direction and steering clear of obstacles. If one puts more bias towards the former, then robot may collide with an obstacle whereas putting more bias towards the later leads the robot wandering off from the path. In both cases performance is undesirable, hence one should properly tune various parameters to balance this tradeoff.

5 REFERENCES

1. Drew Bagnell. Occupancy Maps, Statistical Techniques in Robotics. https://www.cs.cmu.edu/ 16831-f14/notes/F14/16831_lecture06_agiri_dmcconac_kumarsha_nbhakta.pdf

2. Borenstein, Johann, and Yoram Koren. "The vector field histogram-fast obstacle avoidance for mobile robots." IEEE transactions on robotics and automation 7.3 (1991): 278-288.

3. Salem, Marwan. "Building an efficient occupancy grid map based on lidar data fusion for autonomous driving applications." (2019).

4. Siegwart, Roland, Illah Reza Nourbakhsh, and Davide Scaramuzza. Introduction to autonomous mobile robots. MIT press, 2011.

5. Coulter, R. Craig. Implementation of the pure pursuit path tracking algorithm. Carnegie-Mellon UNIV Pittsburgh PA Robotics INST, 1992.