Control Systems Design for Autonomous Vehicles: SPC 418

Dr. Mohamed Elshalakani

Differential Robot Control

University of Science and Technology, Zewail City, Spring 2023

Table of Contents

Introduction

Differential robots are a class of mobile robots that are widely used in various applications, ranging from industrial automation to autonomous vehicles. These robots are characterized by their ability to move and manoeuvre by independently controlling the speeds of their two separate wheels or tracks. The differential drive mechanism allows the robot to achieve both translation and rotation motions, enabling it to navigate through complex environments efficiently.

To understand the dynamics of differential robots, it is essential to delve into the mathematical foundations that govern their motion. The dynamics of a differential robot can be described using a mathematical model that relates the inputs to the resulting motion. This model involves several key parameters and variables, including the robot's geometric properties, wheel radius, mass distribution, and the forces and torques acting on the robot.

Mathematically, the dynamics of a differential robot can be represented using the kinematic equations, which relate the robot's velocity and angular velocity to the velocities of its individual wheels. These equations are derived from the basic principles of rigid body motion and can be expressed as:

$$V = \left(\frac{R}{2}\right)(\omega_l + \omega_r)$$

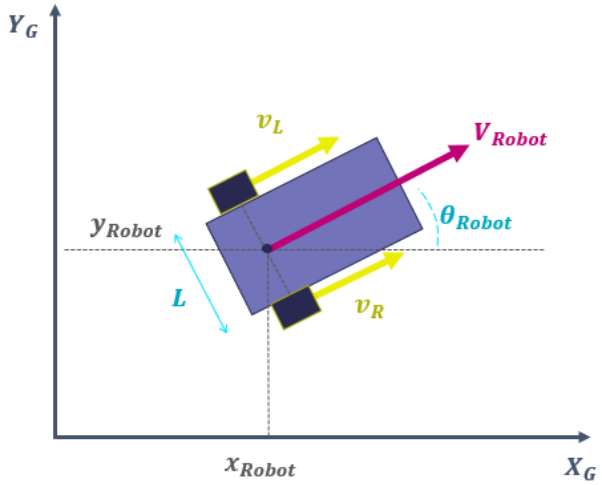
$$\omega = \left(\frac{R}{L}\right)(\omega_r - \omega_l)$$

where V represents the robot's linear velocity, ω denotes its angular velocity, R signifies the wheel radius, and L represents the distance between the robot's two wheels. The subscripts l and r denote the left and right wheels, respectively, while ω_L and ω_R represent their respective angular velocities.

These equations allow us to determine the robot's motion based on the inputs provided to the wheels. By controlling the individual wheel velocities, we can achieve various types of movements, such as forward motion, rotation in place, or curved trajectories. This mathematical representation forms the foundation for designing control algorithms that enable differential robots to navigate their environment effectively.

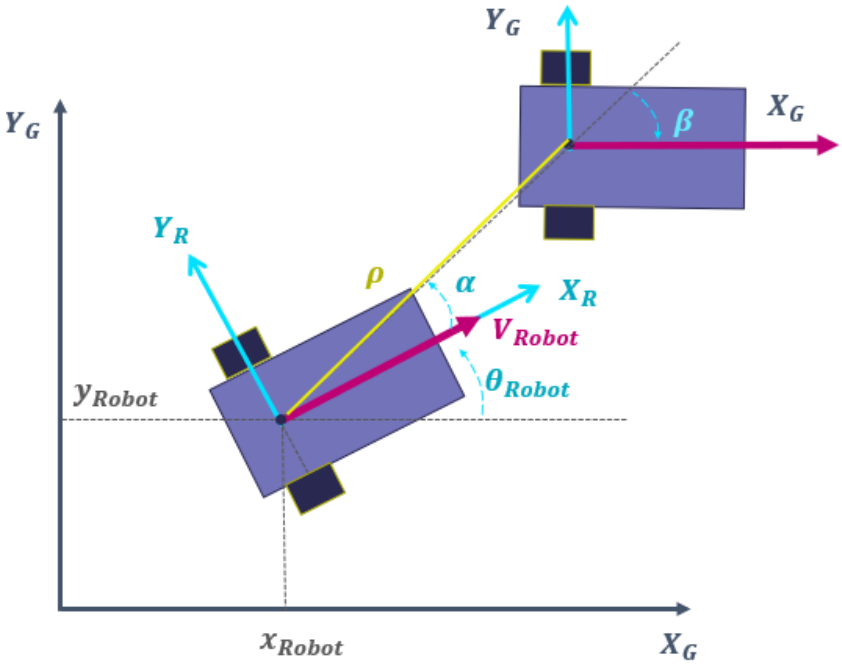
The system model can be written in terms of $x, y, and\ \theta$.

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}\frac{rcos\theta}{2} & \frac{rcos\theta}{2}\\ \frac{rsin\theta}{2} & \frac{rsin\theta}{2}\\ \frac{r}{L} & -\frac{r}{L}\end{bmatrix}\begin{bmatrix}\omega_R\\ \omega_L\end{bmatrix}$$



Control Algorithms

Linear



Defining $\rho, \alpha, and\ \beta$ as follows

$$\rho = \sqrt{\Delta x^2 + \Delta y^2}$$

$$\alpha = zatan2(\Delta y, \Delta x) - \theta$$

$$\beta = -\alpha - \theta$$

Yields this relation.

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -cos\ \alpha & 0 \\ sin\ \alpha/\rho & -1 \\ -sin\ \alpha/\rho & 0 \end{bmatrix} \begin{bmatrix} V \\ \omega \end{bmatrix}$$

This relation will be used to control the response of the robot. The best gains are $k_\rho = 0.1,\ k_\alpha = 0.2,\ k_\beta = -0.1$

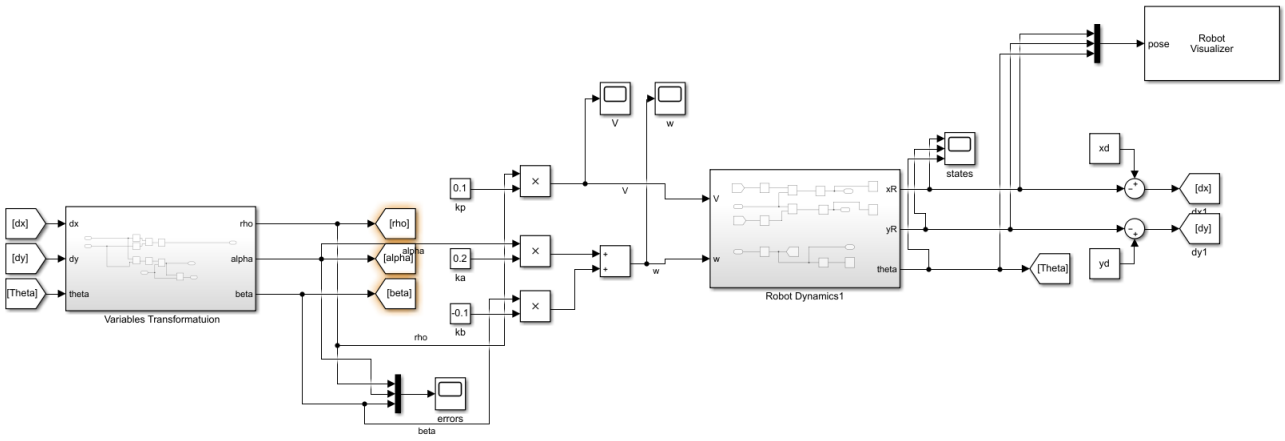


Figure 3 Simulink Model of Linear Control

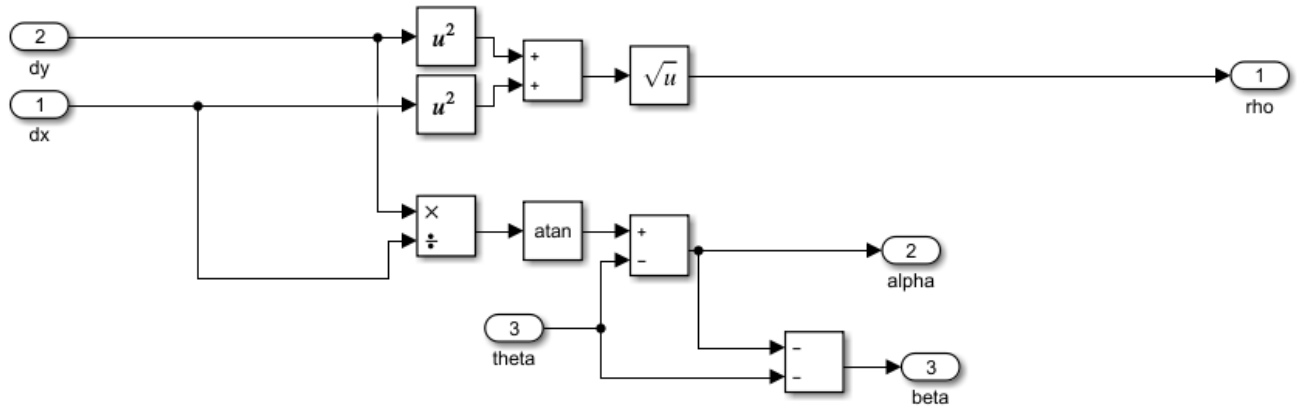


Figure 2 Inside the Variable Transformation Block

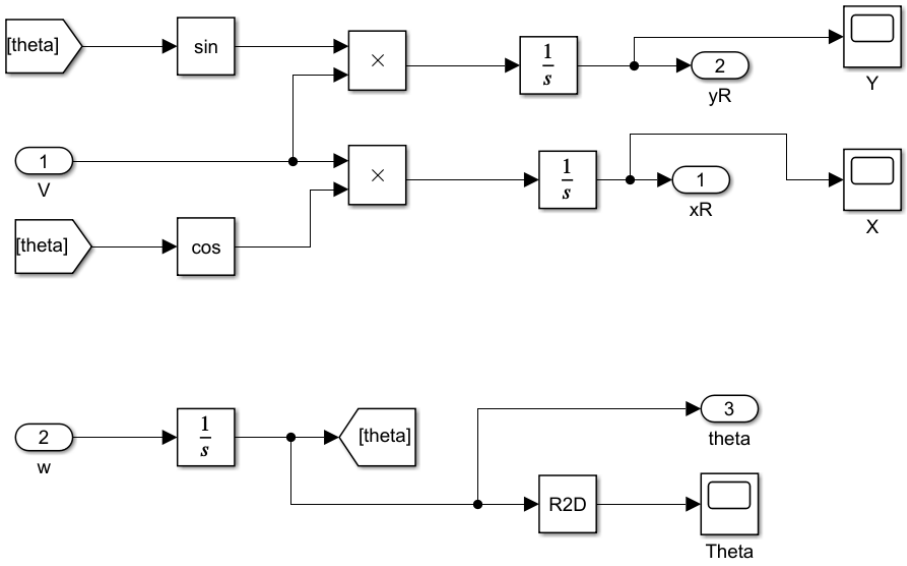


Figure 1 Inside the Robot Dynamics Block

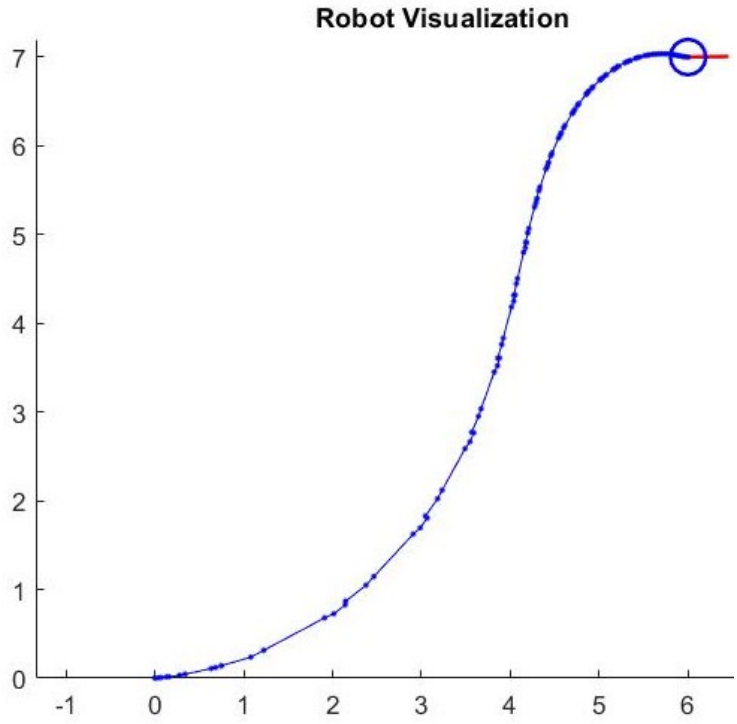


Figure 4 Robot response for gains $k_\rho = 0.1,\ k_\alpha = 0.2,\ k_\beta = -0.1$

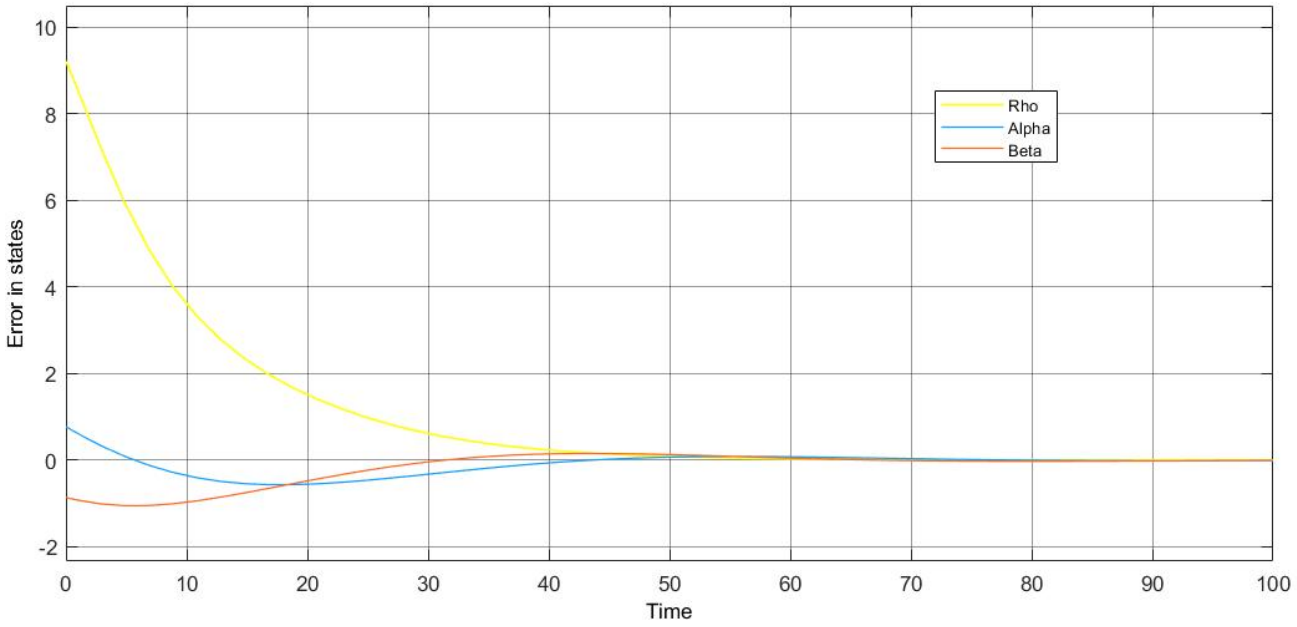


Figure 5 States error for gains $k_\rho = 0.1,\ k_\alpha = 0.2,\ k_\beta = -0.1$

After testing with different sets of gain values, $k_\rho = 0.1,\ k_\alpha = 0.2,\ k_\beta = -0.1$ is the best as it gives a good settling time, path, and a reasonable acceleration. The MATLAB file "eigen" was used to get stable gain samples. General conclusions are drawn as follows:

1- $k_\alpha > k_\rho$
2- $k_\alpha,\ k_\rho$>0
3- $k_\beta < 0$.

In cases where the target is behind the starting point ($x_d < x_0$ *or* $y_d < y_0$) even by 0.1, the path diverges, which supports our conclusion as before.

Lyupanov

The best set of gains is k_x=500 and k_θ=100.

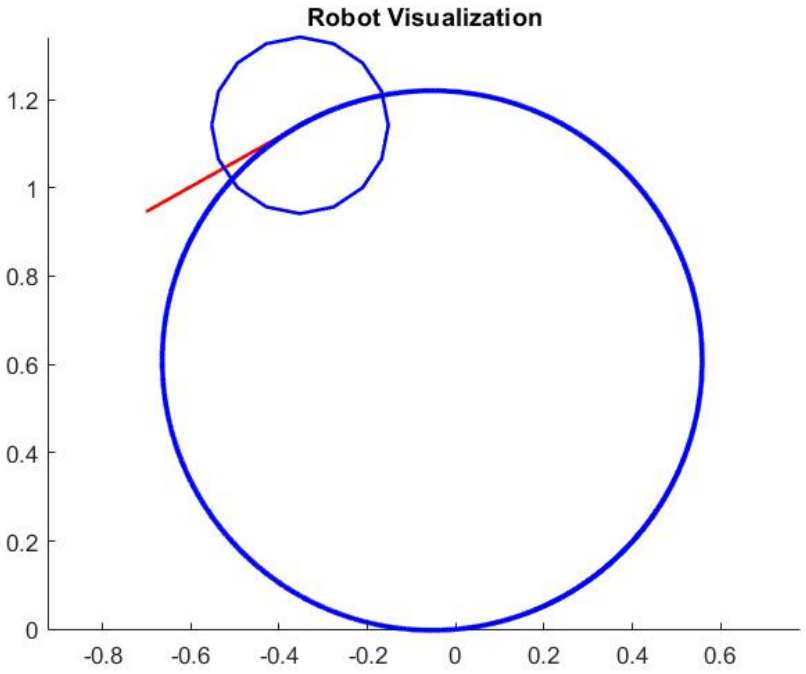


Figure 6 Robot Path after Lyapunov control, for k_x*=500 and* k_θ*=100,* ω_R*=10 and* ω_L*=1*

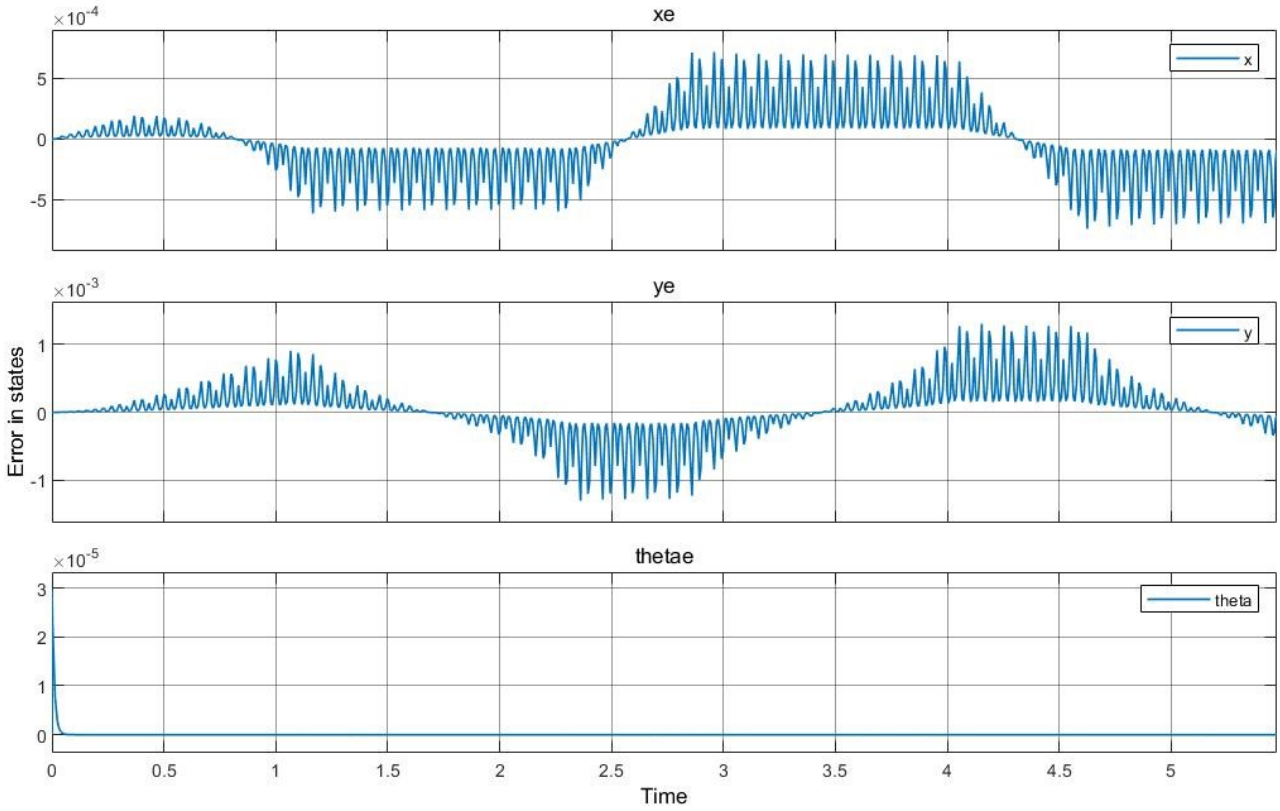


Figure 8 States Error after Lyapunov control, for k_x*=500 and* k_θ*=100,* ω_R*=10 and* ω_L*=1*

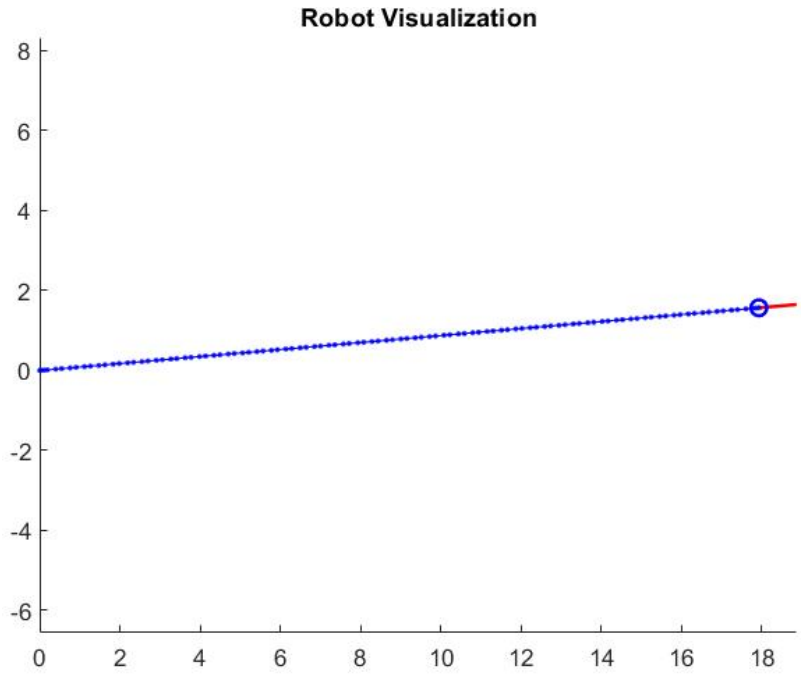


Figure 7 Robot path after Lyapunov control, for k_x*=500 and* k_θ*=100,* ω_R*=3 and* ω_L*=3*

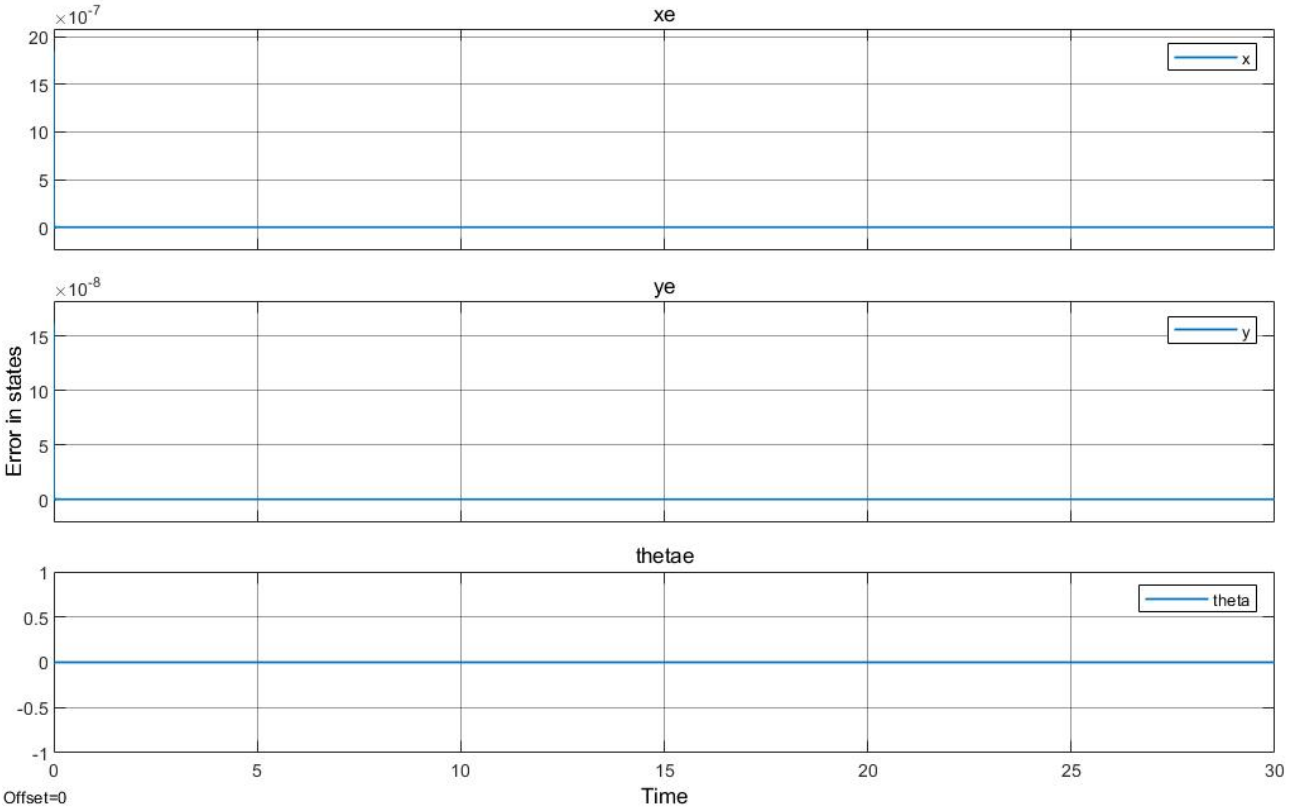


Figure 9 States Error after Lyapunov control, for k_x*=500 and* k_θ*=100,* ω_R*=3 and* ω_L*=3*

The errors are much less when the path is linear. In addition, Lyupanov based control is much better than linear control. And it allows going backwards.

Path Planning

Algorithm

By following the artificial potential field algorithm an attractive field and repulsive field has been developed.

$$U_{att} = k_{att}(q - q_{goal})^2$$

k_{att} is the gain for attraction field, and $q_{goal}$is the position of the goal.

$$q_{goal} = [y_{goal} \quad x_{goal}]$$

around any obstacle, there is a repulsive fiedl withen a certaon raneg. This range is ρ_lim is specified as L (the length of the robot) to allow for robot rotation around the obstacle.

$$U_{rep} = \begin{cases} k_{rep}\left(\frac{1}{\rho(q)} - \frac{1}{\rho_{lim}}\right)^2, & if\ \rho(q) <= \rho_{lim} \\ 0 \quad , & otherwise \end{cases}$$

k_{rep} is the gain for the repulsive field.

By adding the two fields, a gradient is specified in the form of a vector field with components g_y and g_x. This resembles the Laplace of the augmented function at point i.

$$G_i = [g_{y_i}, g_{x_i}]$$

The Laplace is normalized by its Euclidean norm. The new step is chosen from the next method.

$$q_{i+1} = q_i - \alpha * G_i$$

A negative sign is used to get the steepest decent direction of the point.

Optimization techniques can be used to get the best value of α, but for now it is determined iteratively.

Conditions to determine the step size α:

1- Does not exceed boundaries: no negative new points and no points that exceed the goal.
2- Does not get to a point with a higher potential field.

Algorithm to determine the step size α:

1- α is chosen to be 5 times the smallest grid size (x_t, or y_t) at the start of each point in path.
2- If α violates one of its conditions, it is reduced by 10%, and the iteration begins again.

Further speeding condition is put:

1- If the new point is close to the goal by 0.2, the new point is the goal point.

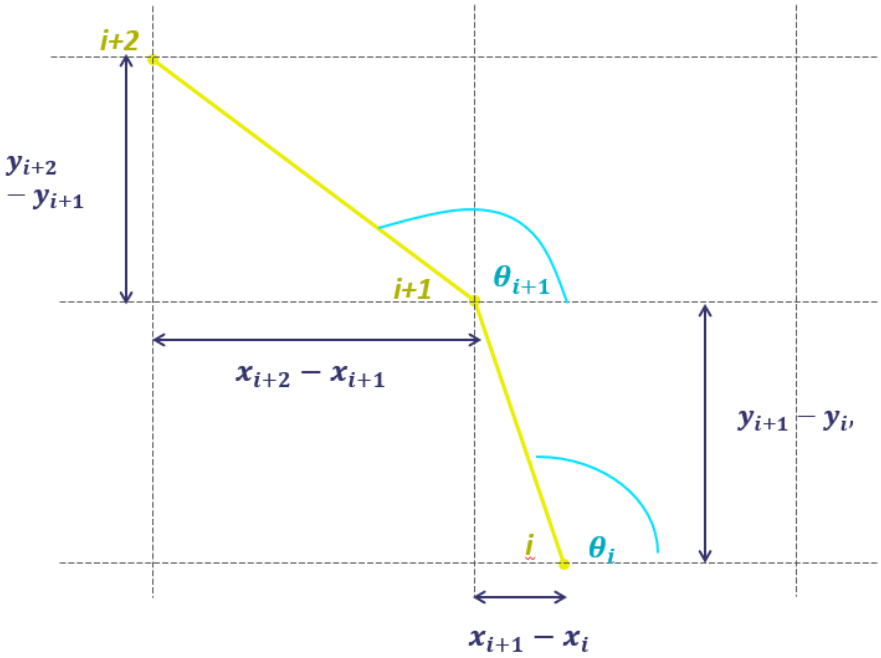
2- If progress near the goal point is slow, do not create a virtual obstacle.

For virtual obstacles:

1- If at least 10 steps in the path have been taken, compare the current point with the one before it by 10 steps.
2- If the difference is less than 0.2 and the new point is close to the goal by 1, create a virtual obstacle at the new point.
3- Create an obstacle by going u points up, down, left, and right and change the original map position to obstacle.
4- Get the new augmented field.
5- Set i=1 and overwrite original path.

At the end, to get the heading angle θ

$$\theta_i = atan2(y_{i+1} - y_i, x_{i+1} - x_i)$$



Results

Map 1

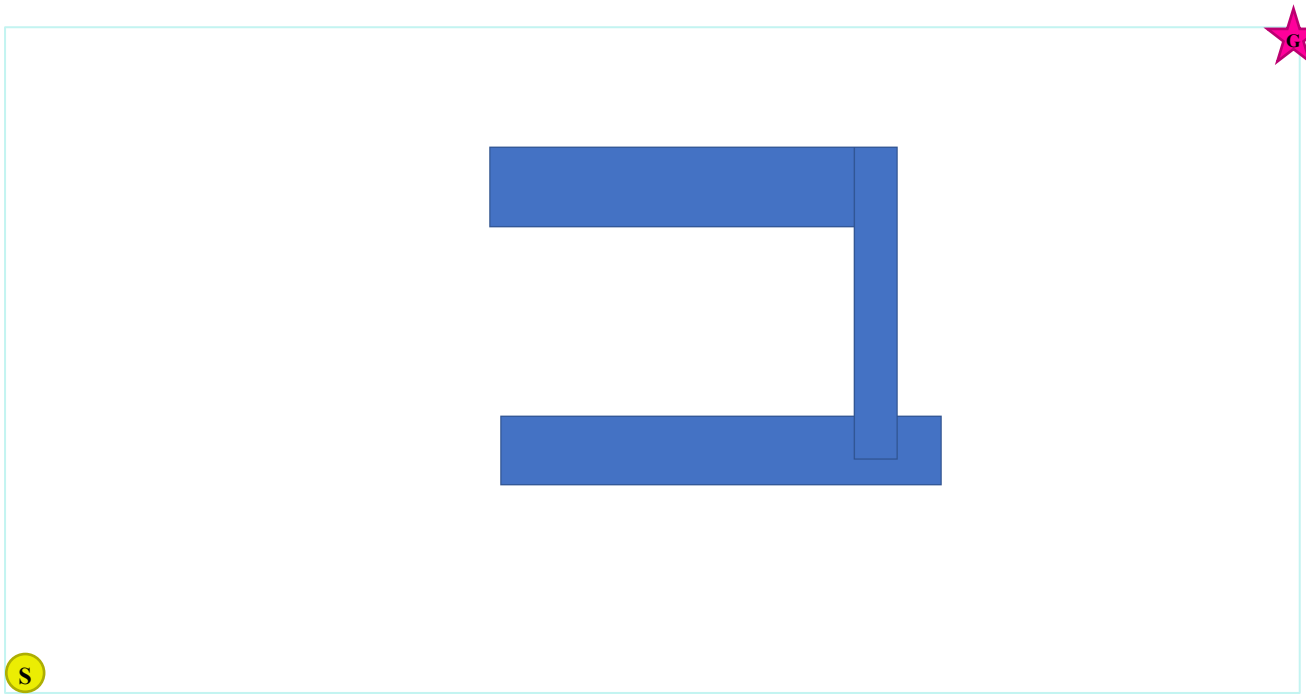


Figure 10 map 1 with start point at bottom left corner (0,0) and goal at upper right corner (6,7).

MATLAB

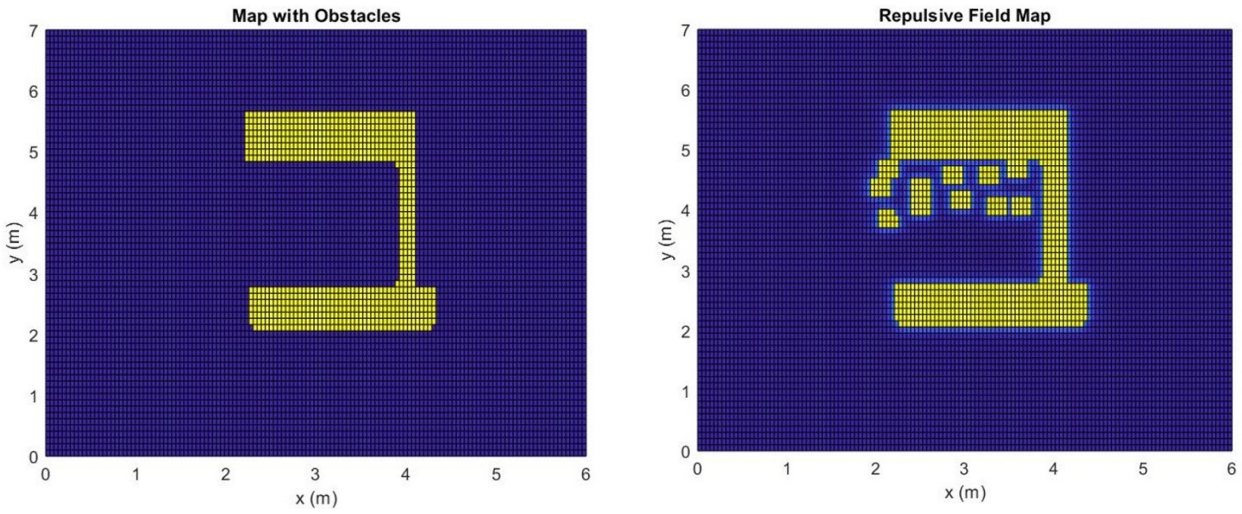


Figure 11 On the left is the map retrieved from the picture. On the right is a top view of the repulsive field. The blue hallow represents a fading repulsive force around the obstacle.

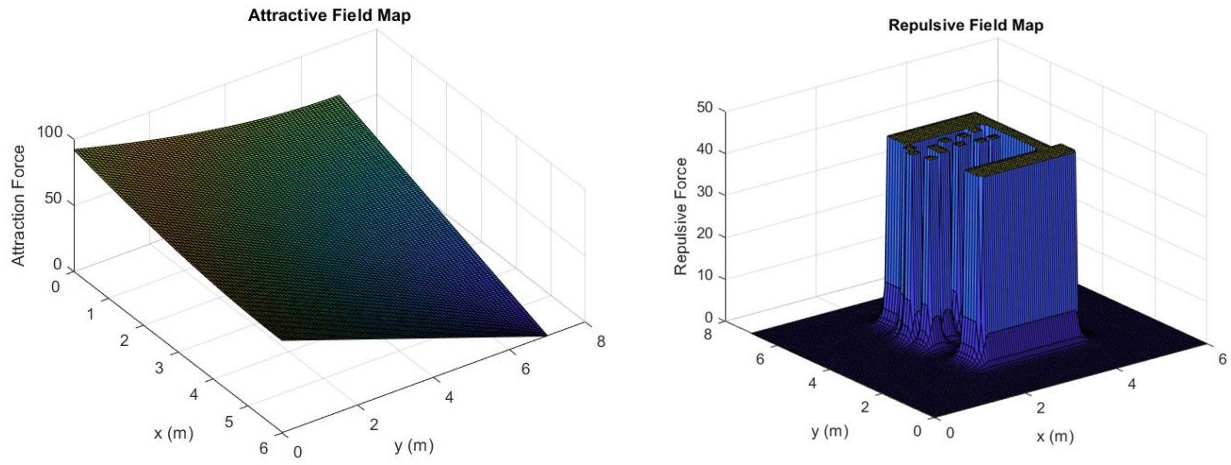


Figure 12 On the left is the attraction field. On the right is the repulsion field.

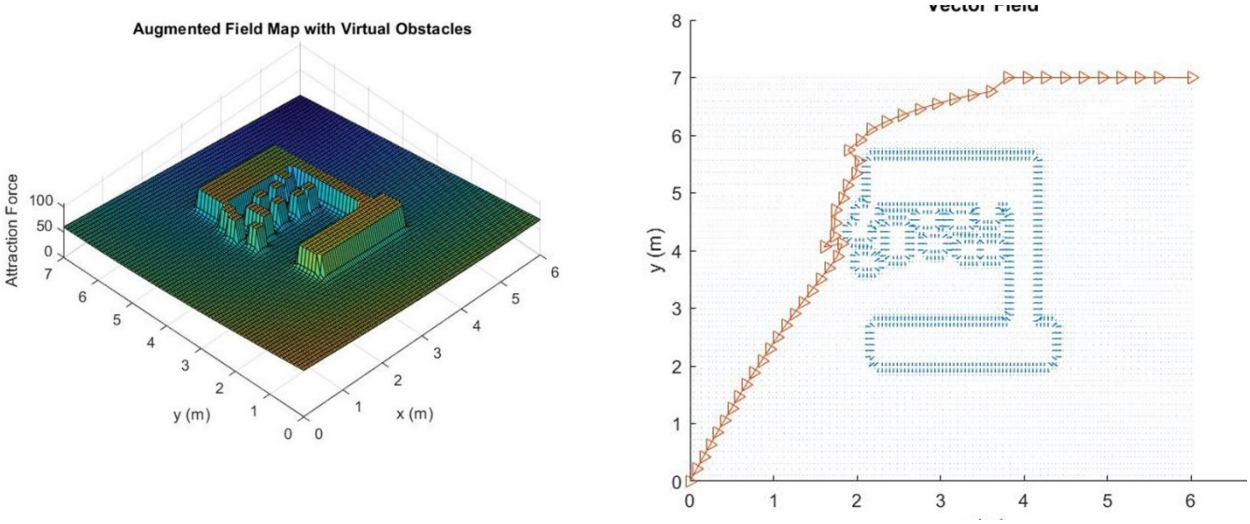


Figure 13 On the left is the augmented field with the virtual obstacles' repulsion add. On the right is the path of the robot on a vector field of map

Simulink: Lyupanov

There were a lot of factors affecting the accuracy of the simulation and the convergence of the solution. I have drawn some conclusions about how to set those parameters.

1- Import x, y, and θ value from workspace.
2- Choose to keep the final value after the input has ended before the simulation.
3- Choose a variable step-size in solver settings because it automatically adapts to change in sampling time.
4- If you choose a fixed step size in the solver, opt for a sampling time T_s an order lower than the minimum sampling time in the simulation.
5- Choose an automatic solver. If not, do not choose stiff solvers because they consume much longer running time.
6- After some order of k_x, and k_θ, the relative ratio between them does not matter and the solution is optimum. This order is found to be 100.
7- Choose the sampling time 2 orders smaller than the time step used in the path points.
8- An interpolation potation can be used in importing path points from workspace. It gives a slightly different path than a curvilinear one.
9- The interpolation option can be disregarded, but only giving a variable step size in the solver. It gives a more rigid, straight steps, but most precise to the path imported.
10- Not interpolating data increases error in states.
11- Finally, best gains for this map are k_x=1000 and k_θ=500, with interpolated data inputs.

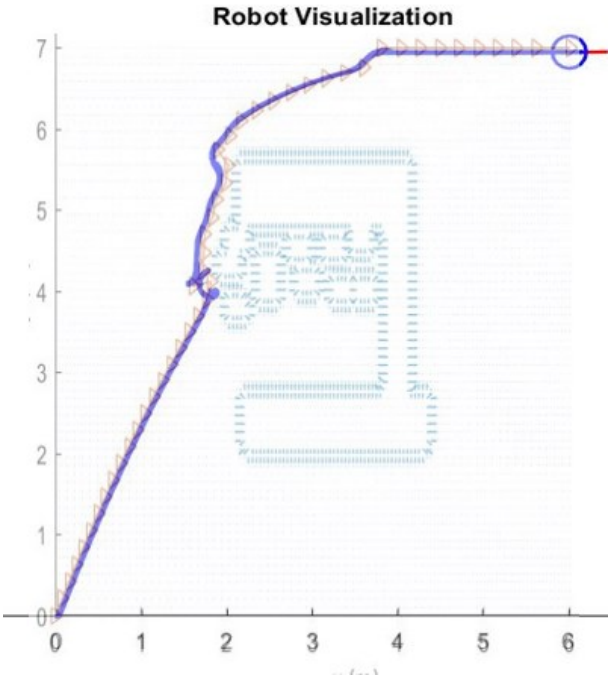


Figure 15 Simulink: Robot path over the map with virtual obstacles defined by vector field. Red line with arrows is the original path developed. k_x*=500 and* k_θ*=100,* $T_s = 0.001$*, way points interpolated*

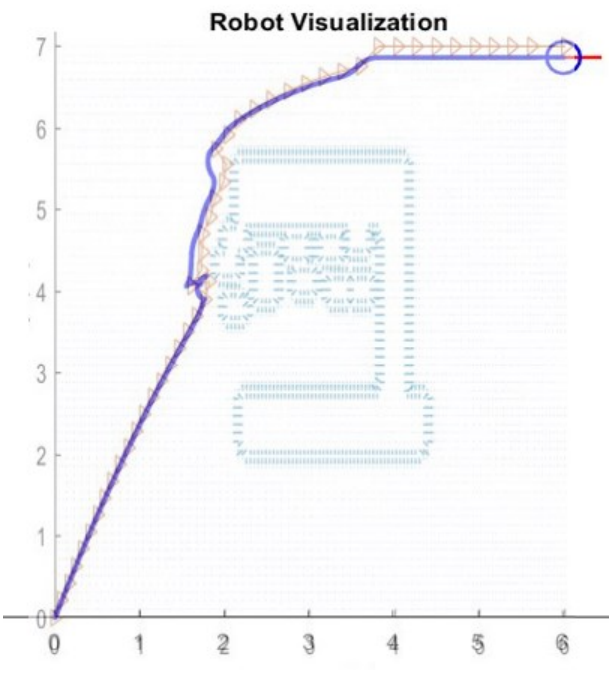


Figure 15 Simulink: Robot path over the map with virtual obstacles defined by vector field. Red line with arrows is the original path developed. k_x*=500 and* k_θ*=500,* $T_s = 0.001$*, way points interpolated*

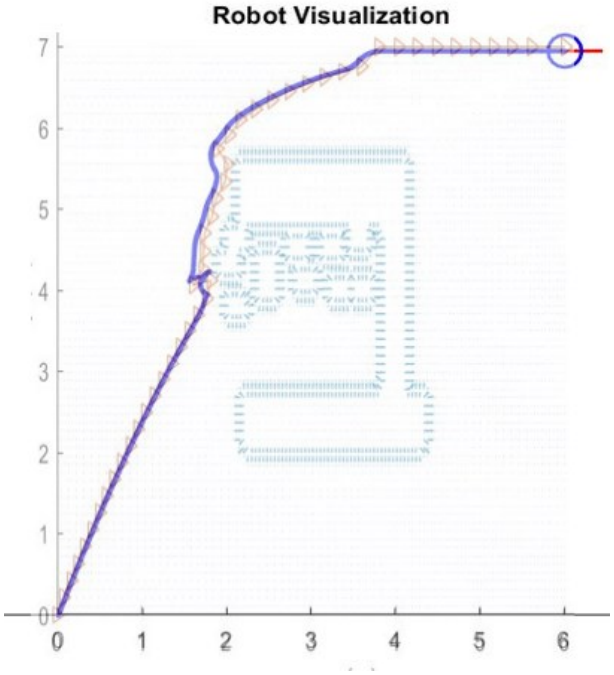


Figure 18 Simulink: Robot path over the map with virtual obstacles defined by vector field. Red line with arrows is the original path developed. k_x*=500 and* k_θ*=1000,* $T_s = 0.001$*, way points interpolated*

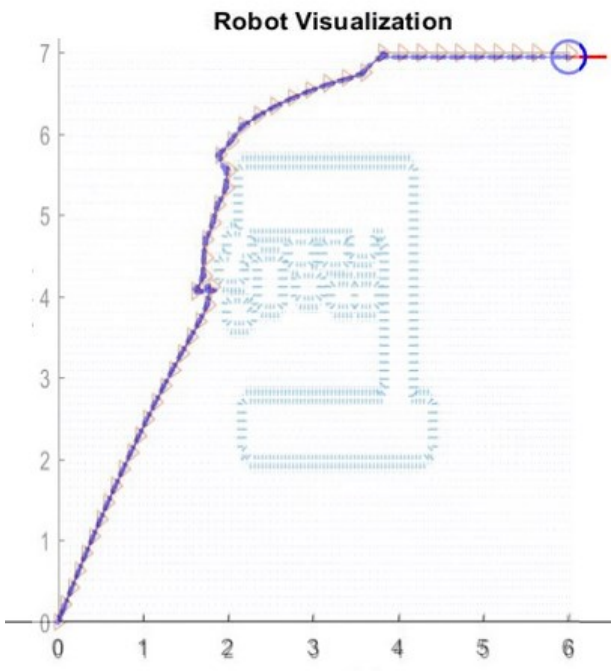


Figure 18 Simulink: Robot path over the map with virtual obstacles defined by vector field. Red line with arrows is the original path developed. k_x*=1000 and* k_θ*=500,* $T_s = 0.001$*, way points not interpolated.*

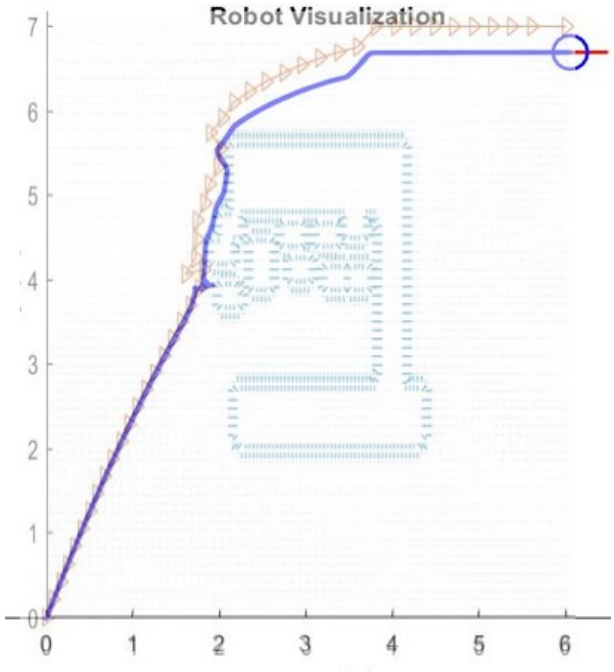


Figure 16 Simulink: Robot path over the map with virtual obstacles defined by vector field. Red line with arrows is the original path developed. k_x*=100 and* k_θ*=500,* $T_s = 0.001$*, way points not interpolated.*

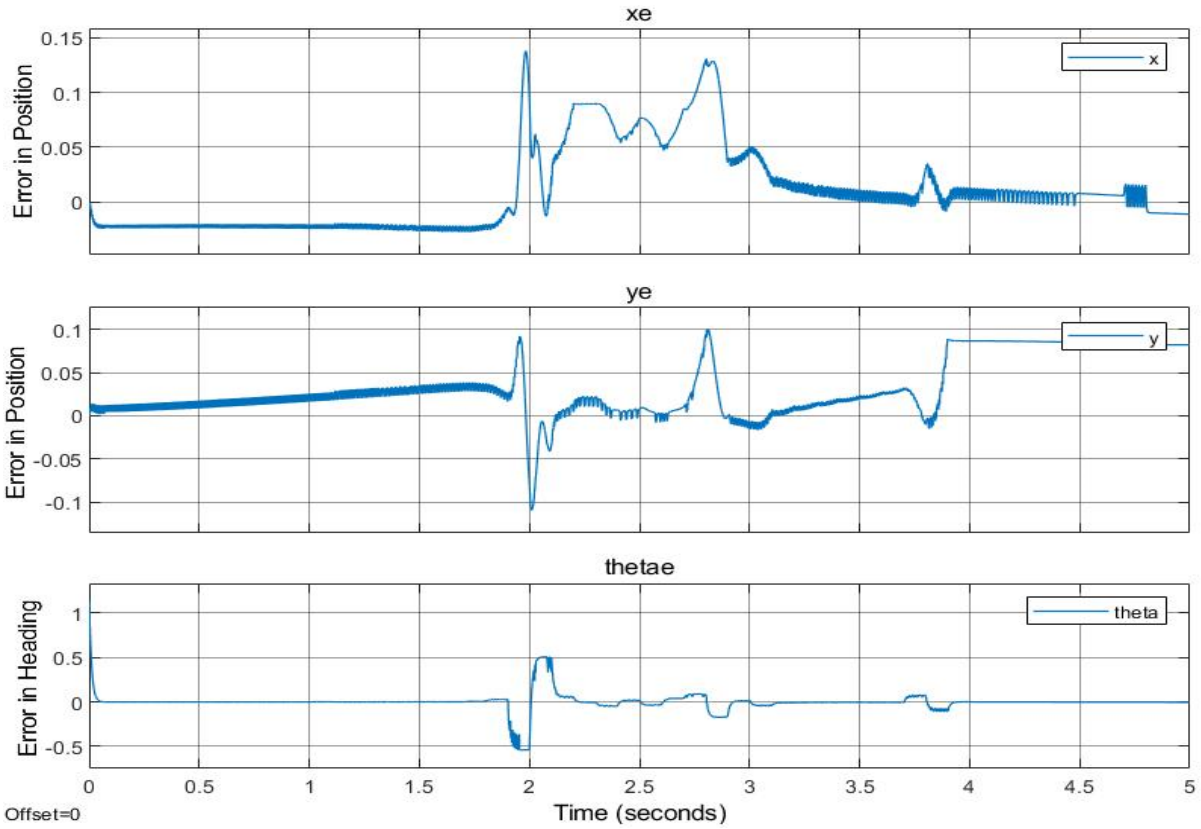


Figure 20 Error in states. k_x*=500 and* k_θ*=100,* $T_s = 0.001$*, way points interpolated.* $x_{e\_max} \approx 0.14$*,* $y_{e\_max} \approx$

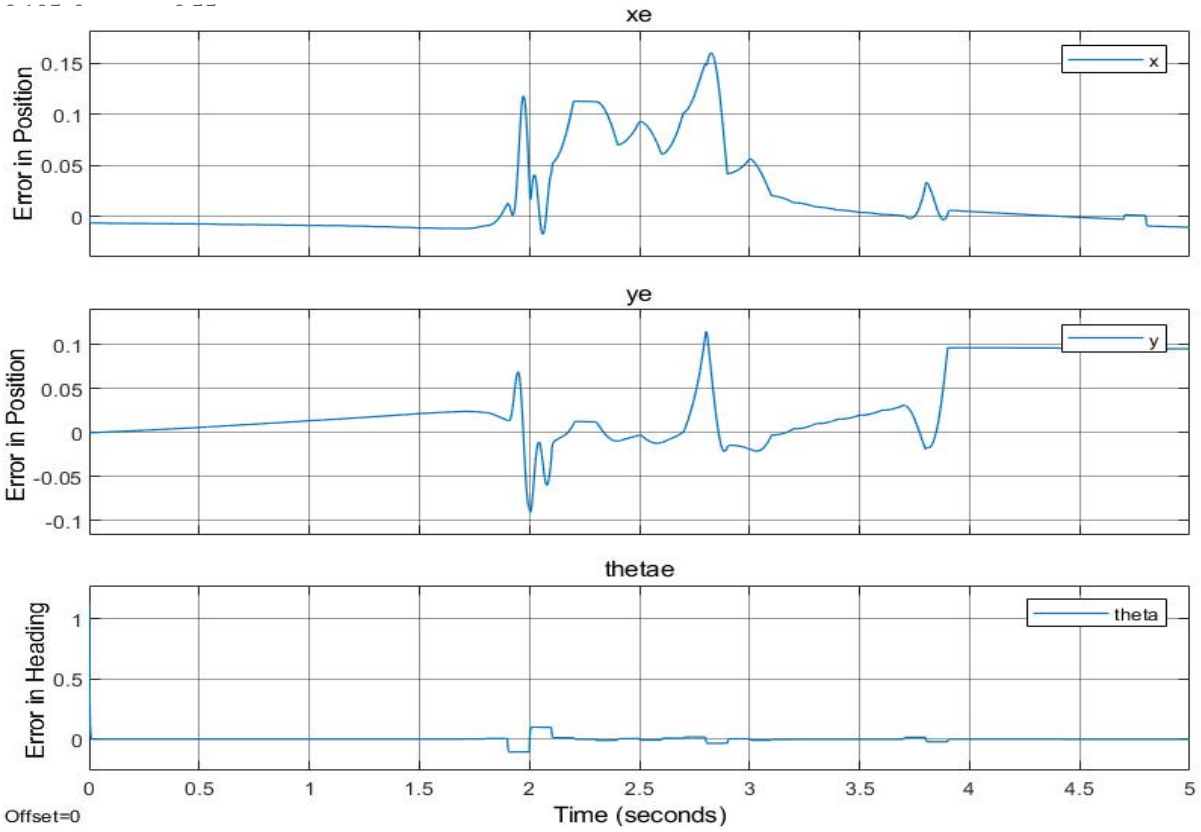


Figure 19 Error in states. k_x*=500 and* k_θ*=500,* $T_s = 0.001$*, way points interpolated.* $x_{e\_max} \approx 0.16$*,* $y_{e\_max} \approx 0.12$*,* $\theta_{e\_max} \approx 0.1$

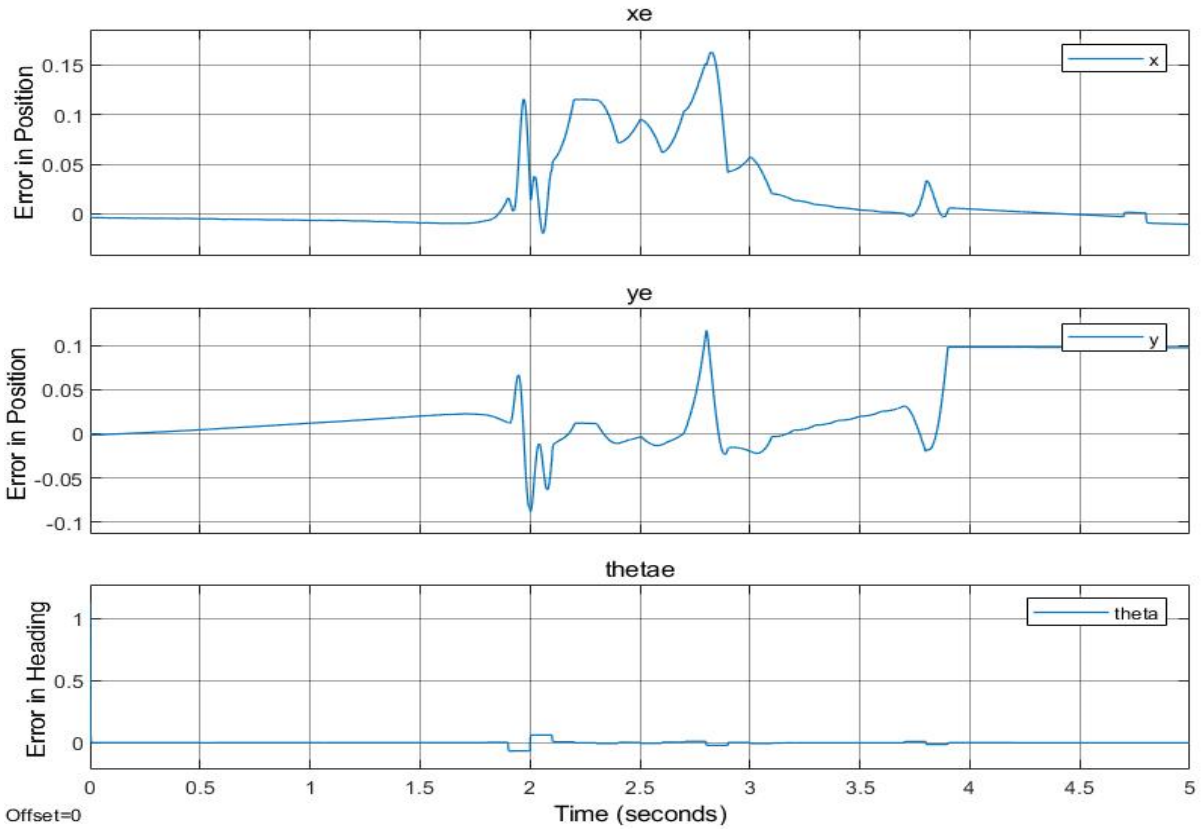


Figure 22 Error in states. k_x*=500 and* k_θ*=1000,* $T_s = 0.001$*, way points interpolated.* $x_{e\_max} \approx 0.16$, $y_{e\_max} \approx 0$

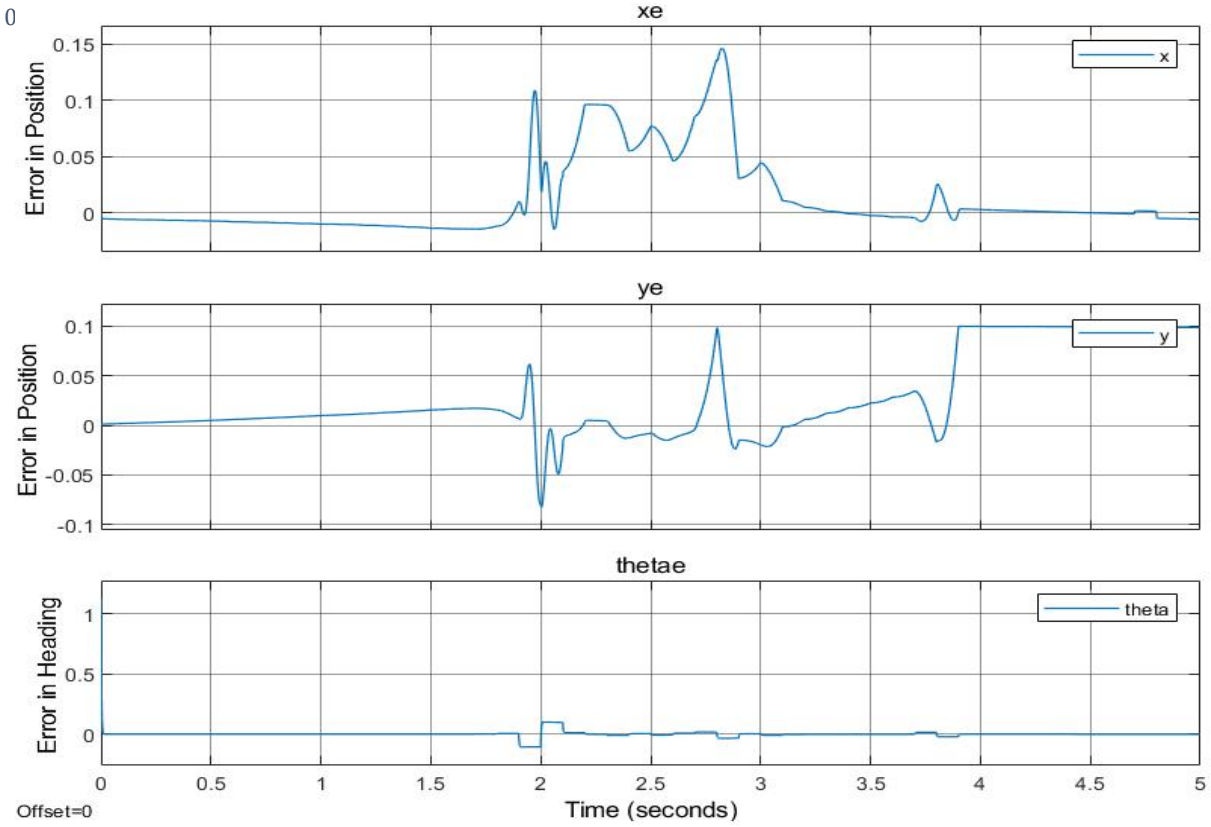


Figure 21 Error in states. k_x*=1000 and* k_θ*=500,* $T_s = 0.001$*, way points interpolated.* $x_{e\_max} \approx 0.145$, $y_{e\_max} \approx 0.1, \theta_{e_m} \approx 0.1$

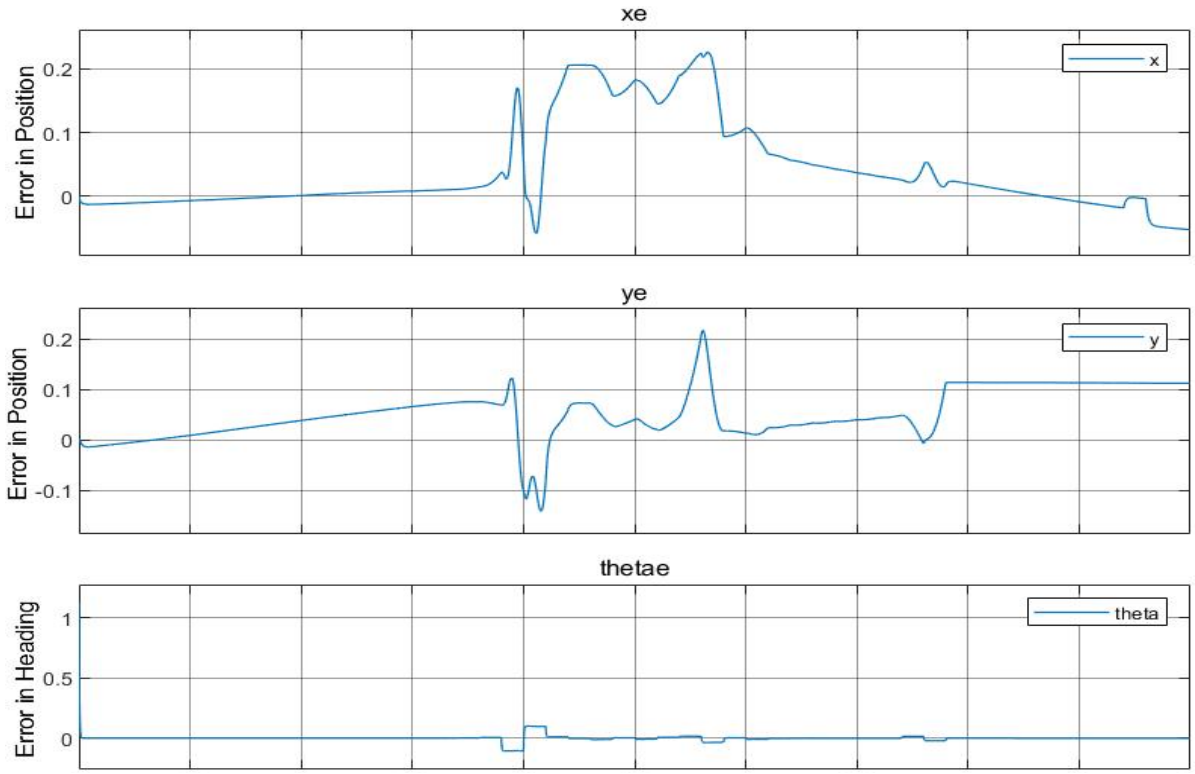


Figure 24 Error in states. k_x*=100 and* k_θ*=500,* $T_s = 0.001$*, way points interpolated.* $x_{e\_max} \approx 0.22$, $y_{e\_max} \approx 0.22, \theta_{e_m} \approx 0.1$

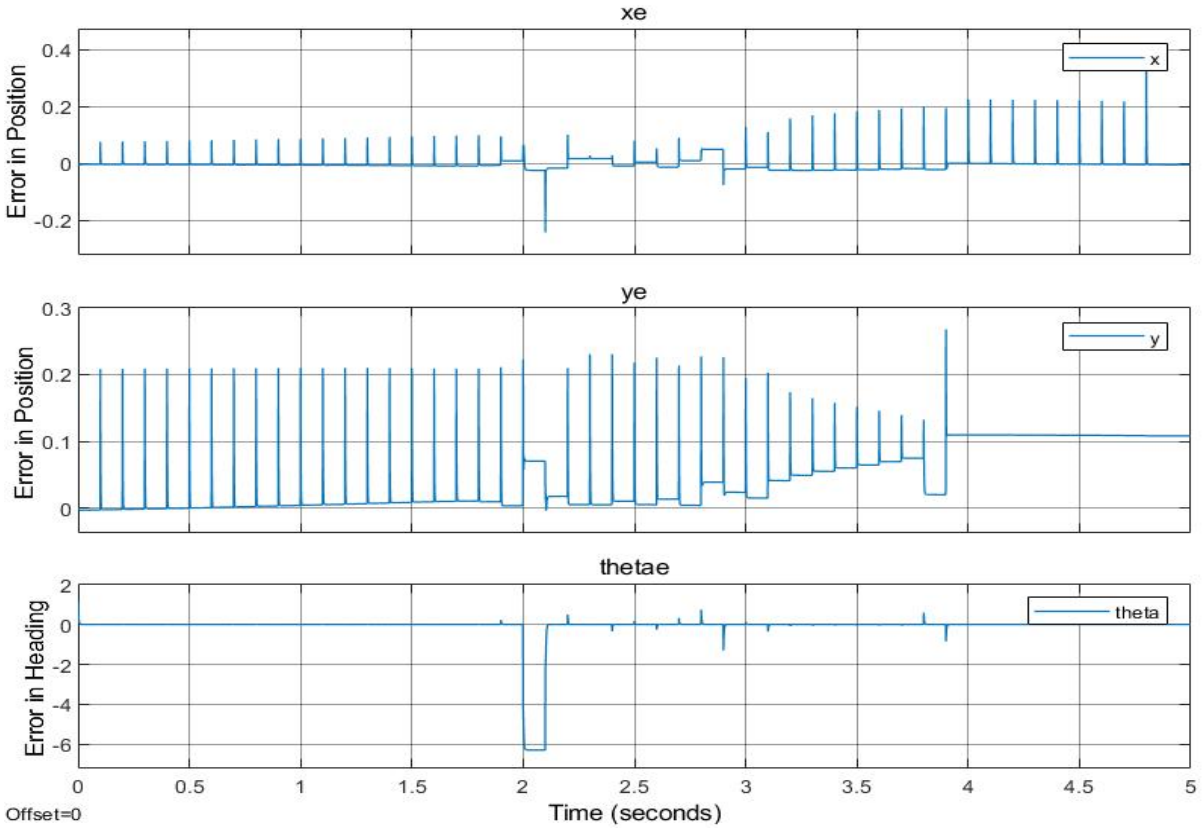


Figure 23 D Error in states. k_x*=1000 and* k_θ*=500,* $T_s = 0.001$*, way points not interpolated.* $x_{e\_max} \approx 0.4$, $y_{e\_max} \approx 0.25, \theta_{e\_max} \approx 6$

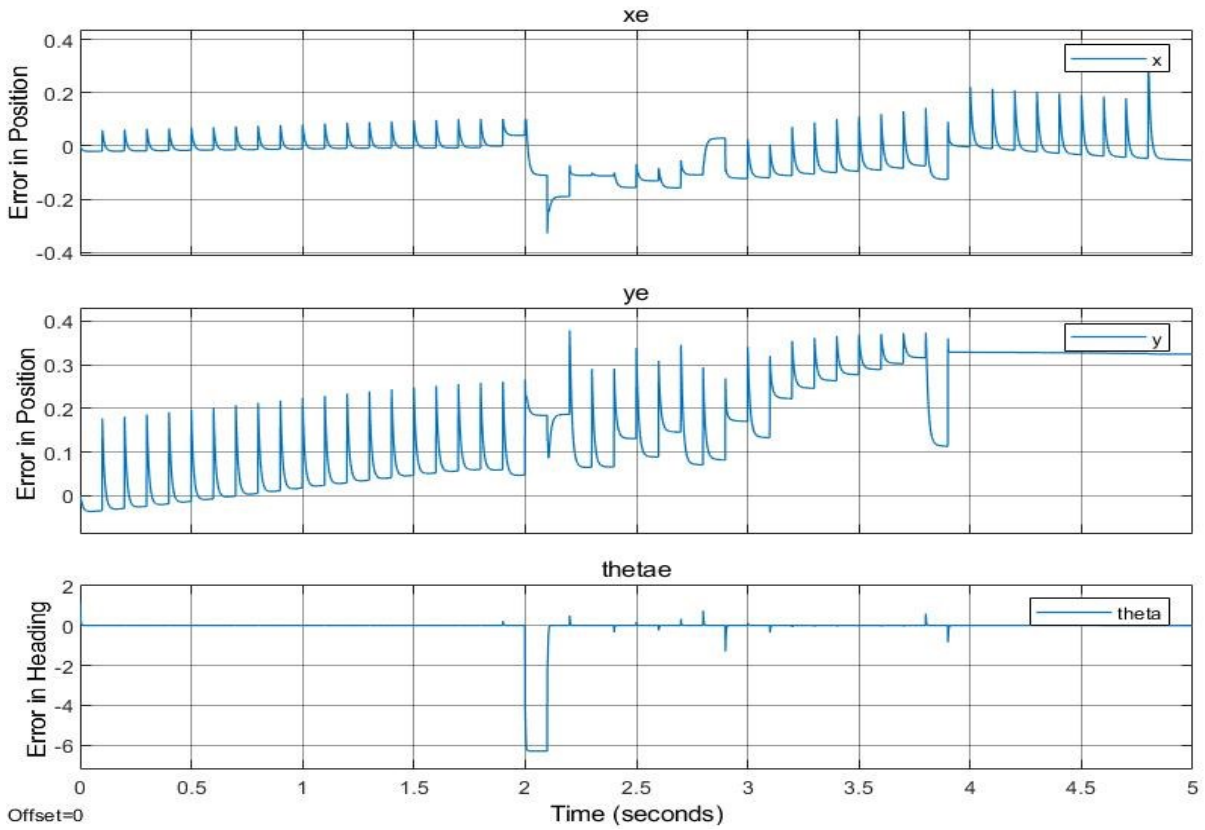


Figure 25 Error in states. k_x*=100 and* k_θ*=500,* $T_s = 0.001$*, way points not interpolated.* $x_{e\_max} \approx 0.3$, $y_{e\_max} \approx 0.32, \theta_{e\_max} \approx 6$

The following tables summarize the errors with different gains and with or without interpolation option.

Table 1 Error comparison for different set of gains, and with data interpolation

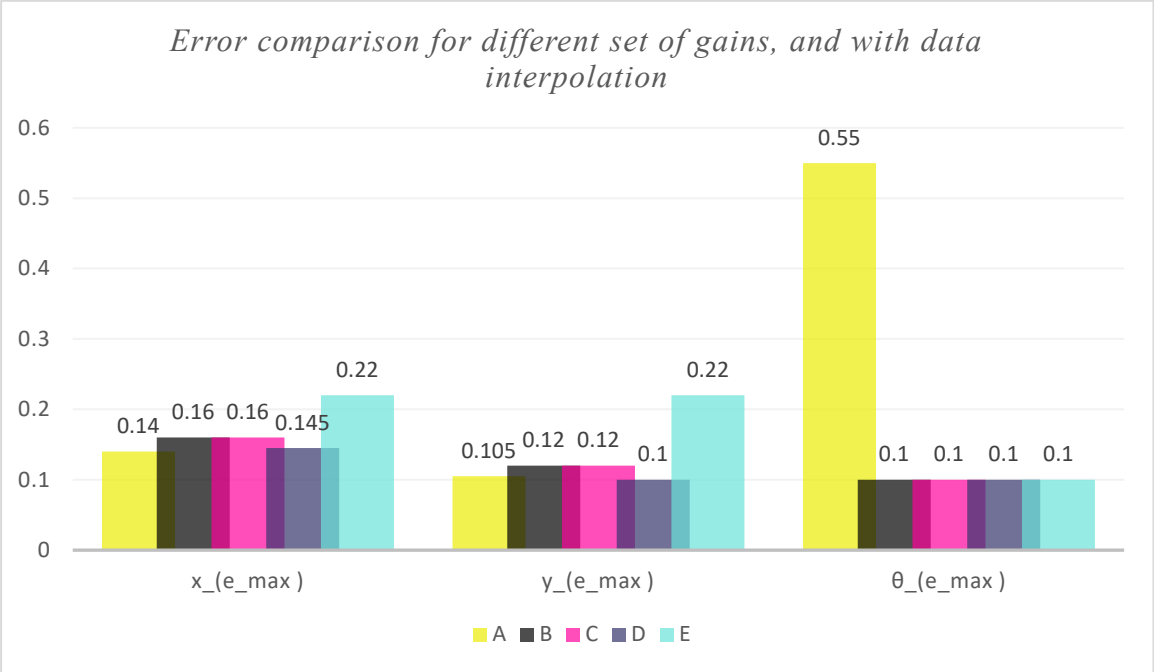
	k_x	k_θ	$x_{e_{max}}$	$y_{e_{max}}$	$\theta_{e_{max}}$
A	500	100	0.14	0.105	0.55
B	500	500	0.16	0.12	0.1
C	500	1000	0.16	0.12	0.1
D	1000	500	0.145	0.1	0.1
E	100	500	0.22	0.22	0.1

Table 2 Error comparison with continuous vs discrete path input for case D

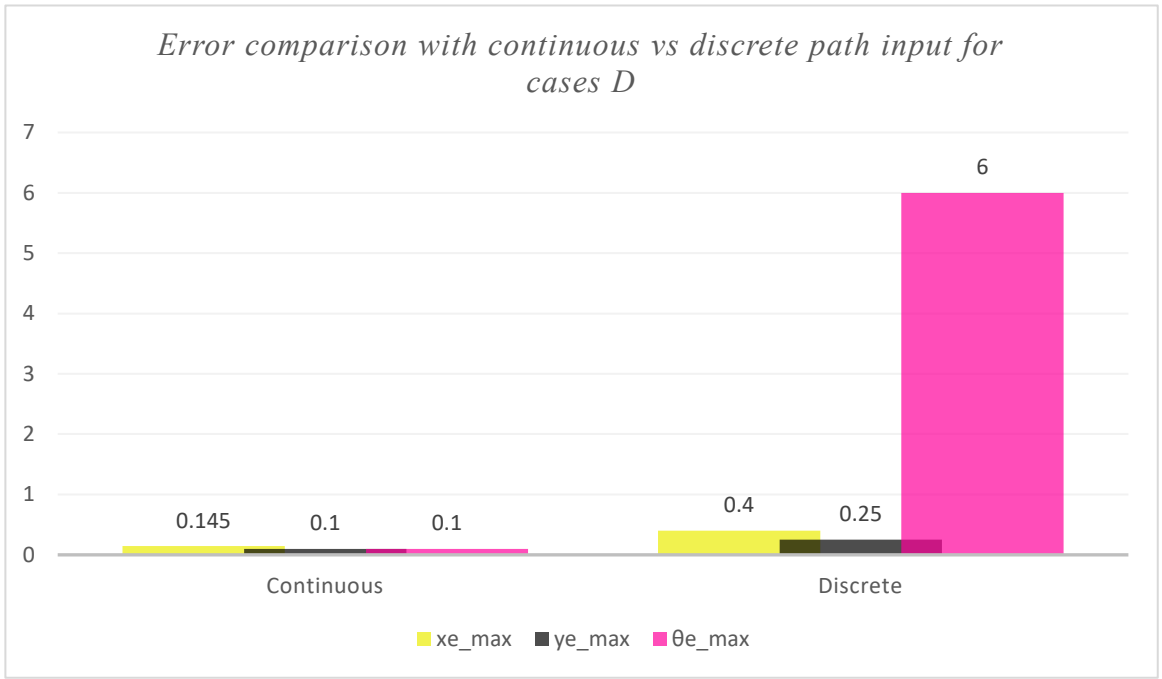
	k_x	k_θ	$x_{e_{max}}$	$y_{e_{max}}$	$\theta_{e_{max}}$
D, continuous	1000	500	0.145	0.1	0.1
D, discrete	1000	500	0.4	0.25	6

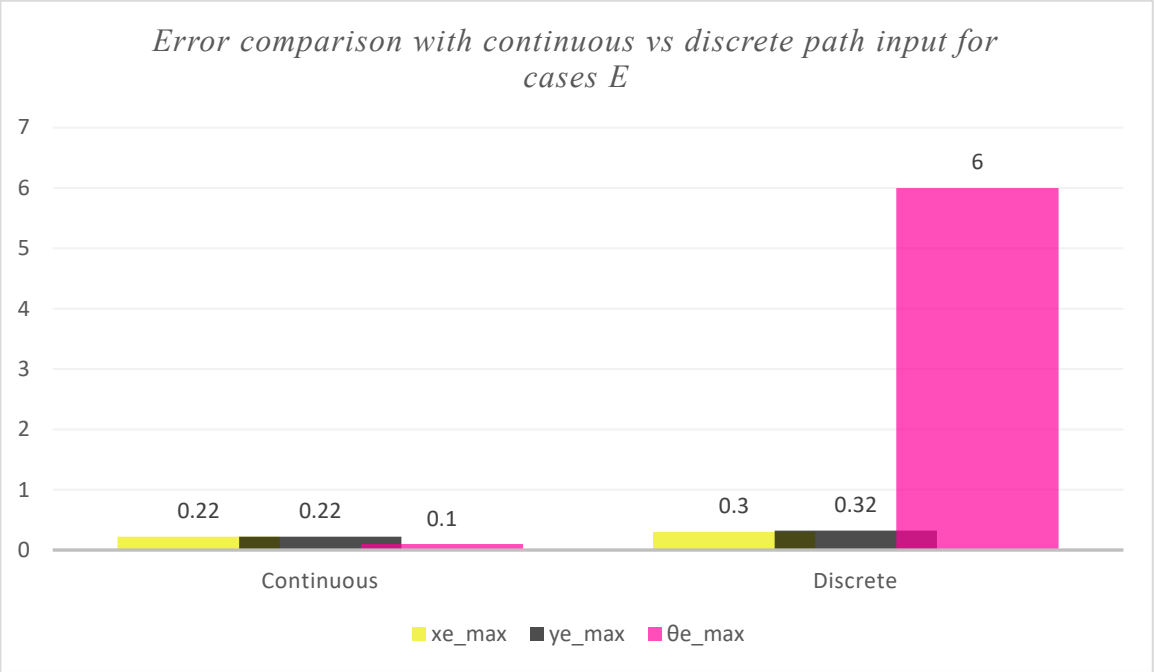
Table 3 Error comparison with continuous vs discrete path input for case E

	k_x	k_θ	$x_{e_{max}}$	$y_{e_{max}}$	$\theta_{e_{max}}$
E, continuous	1000	500	0.22	0.22	0.1
E, discrete	1000	500	0.3	0.32	6



It is visible from the above graph; the minimum error is generated for set of gains D: k_x=1000 and k_θ=500.





Using a discrete set of data input increases states error by almost double, except for theta, which is increased by a factor of 6.

Map 2

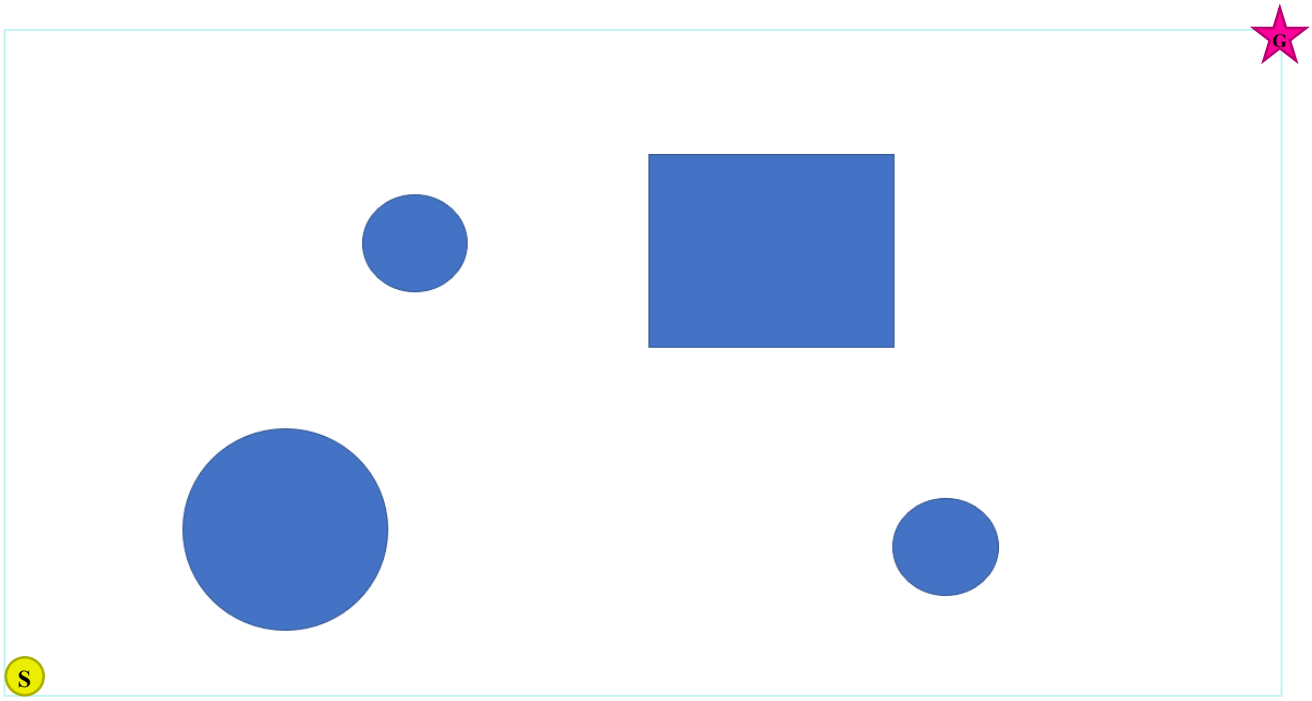


Figure 26 map 2 with start point at bottom left corner (0,0) and goal at upper right corner (6,7).

MATLAB

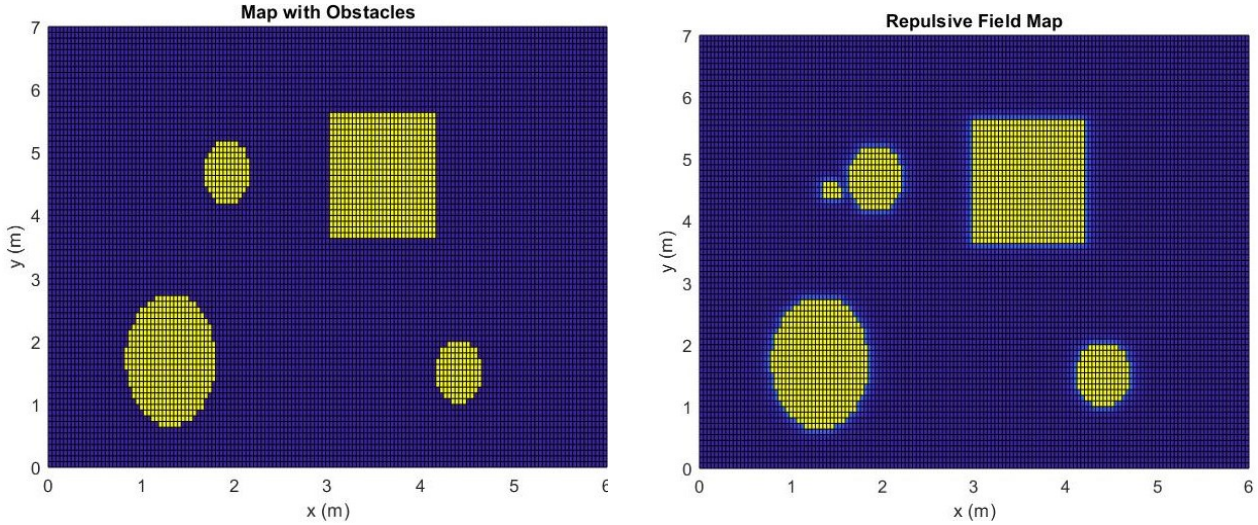


Figure 27 On the left is the map retrieved from the picture. On the right is a top view of the repulsive field. The blue hallow represents a fading repulsive force around the obstacle.

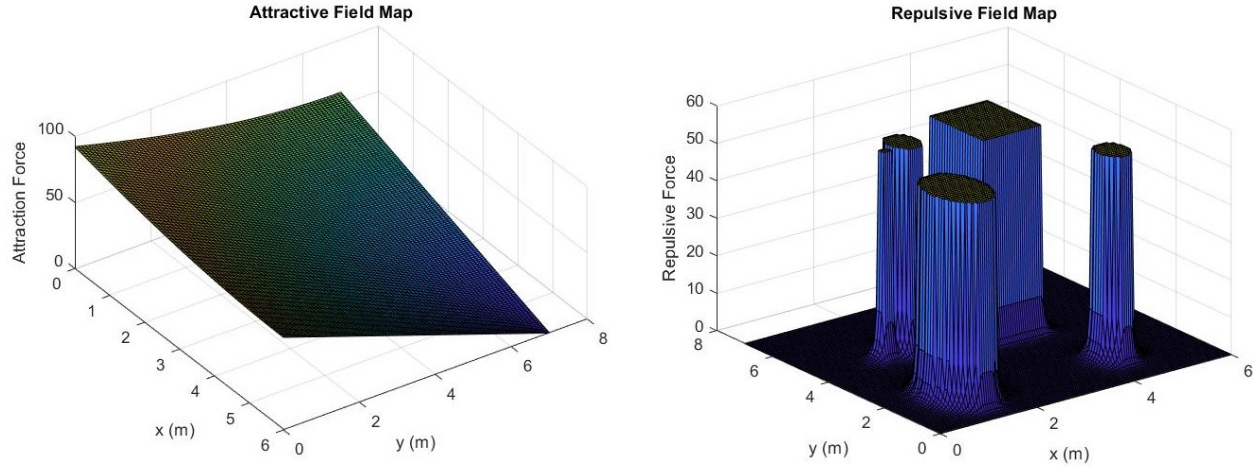


Figure 29 On the left is the attraction field. On the right is the repulsion field.

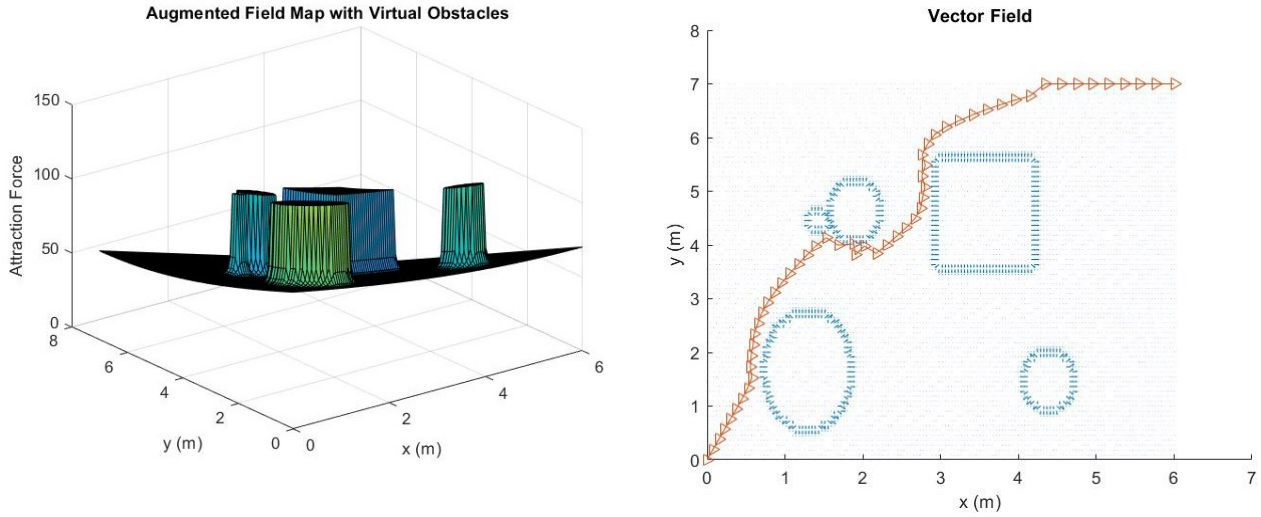


Figure 28 On the left is the augmented field with the virtual obstacles' repulsion add. On the right is the path of the robot on a vector field of map

Simulink: Lyupanov

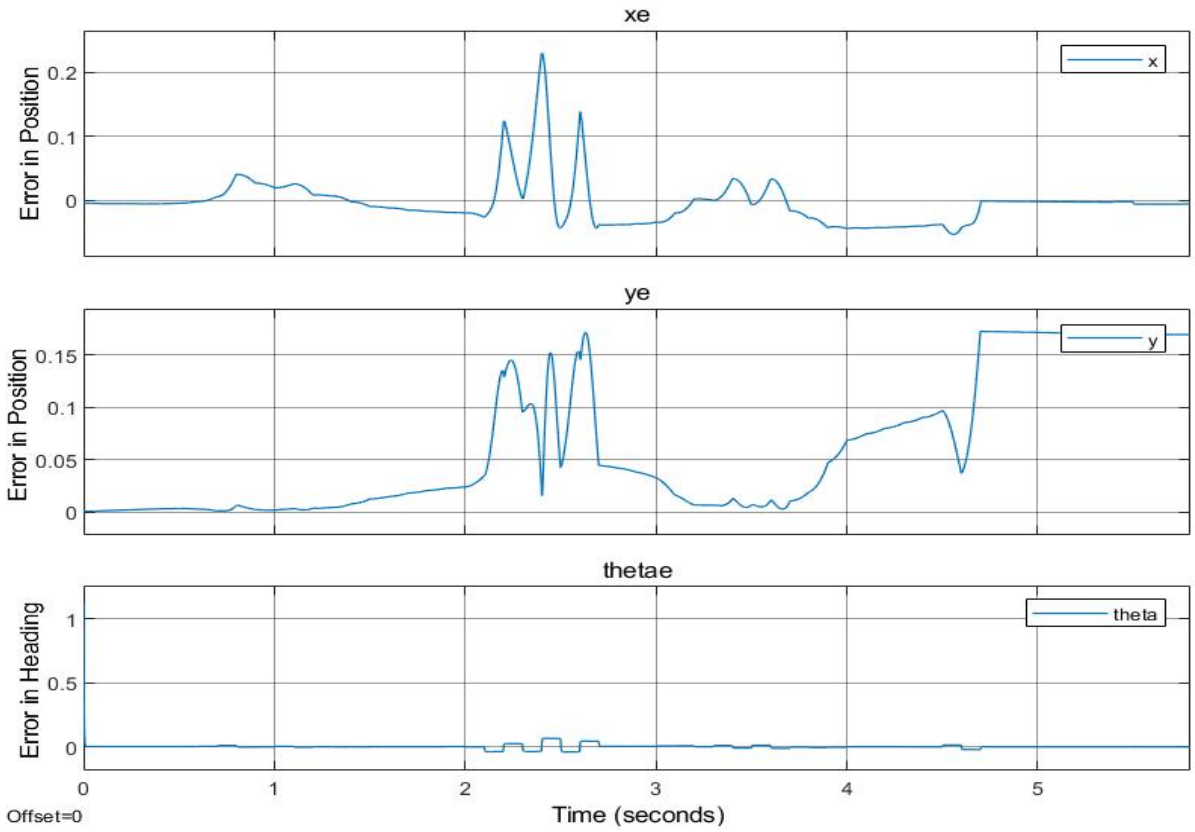


Figure 30 Error in states. k_x*=1000 and* k_θ*=500,* $T_s = 0.001$*, way points not interpolated.* $x_{e\_max} \approx 0.22$, $y_{e\_max} \approx 0.17, \theta_{e_m} \approx 0.05$

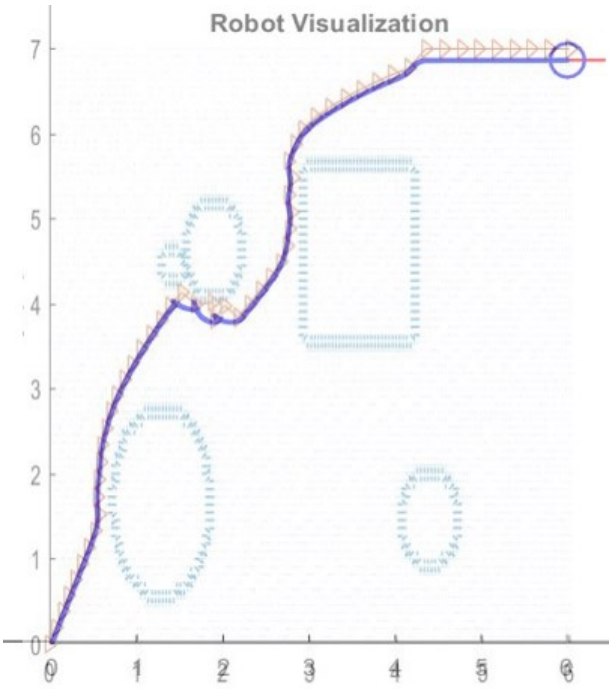


Figure 31 Lyupanov control output (blue) on input path (red)

The same analysis applied before is used here too to obtain the best gains.

	k_x	k_θ	$x_{e_{max}}$	$y_{e_{max}}$	$\theta_{e_{max}}$
D	1000	500	0.22	0.17	0.05

Map 3

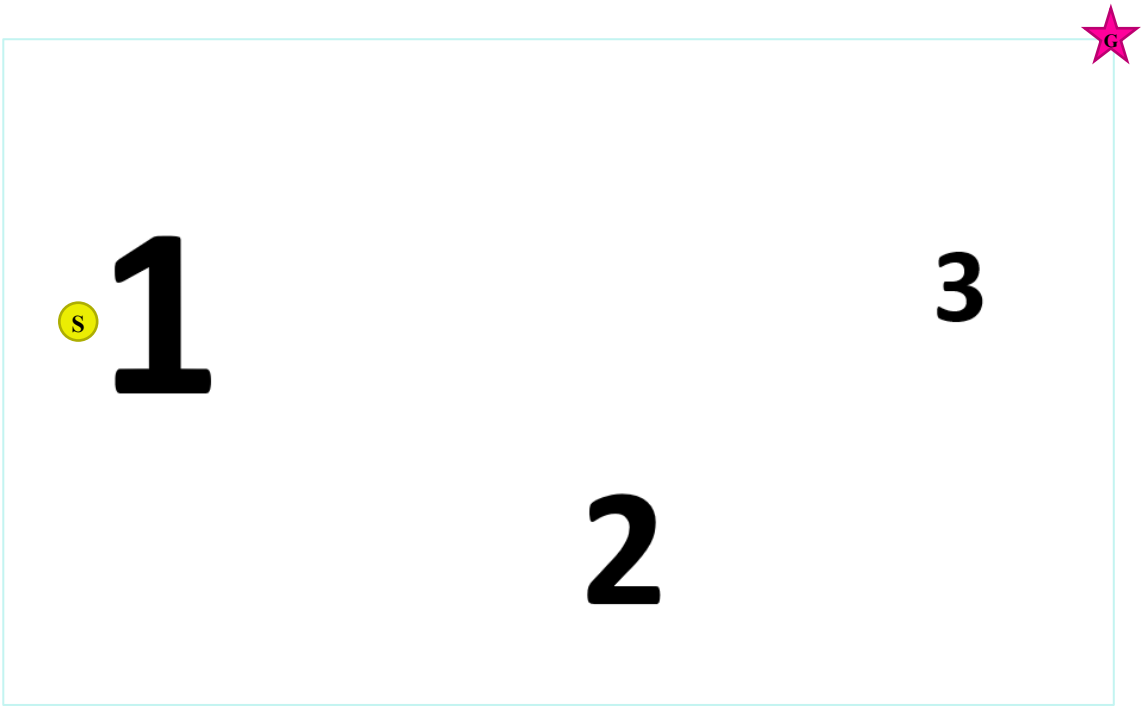


Figure 32 map 3 with start point at (0.3,3) and goal at upper right corner (6,7)

MATLAB

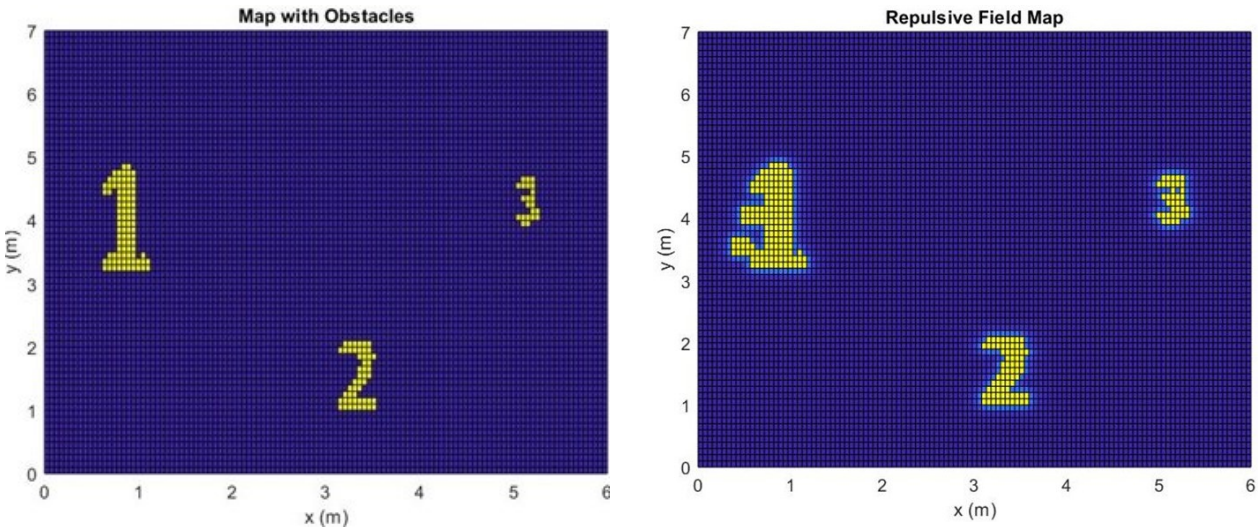


Figure 33 On the left is the map retrieved from the picture. On the right is a top view of the repulsive field. The blue hallow represents a fading repulsive force around the obstacle.

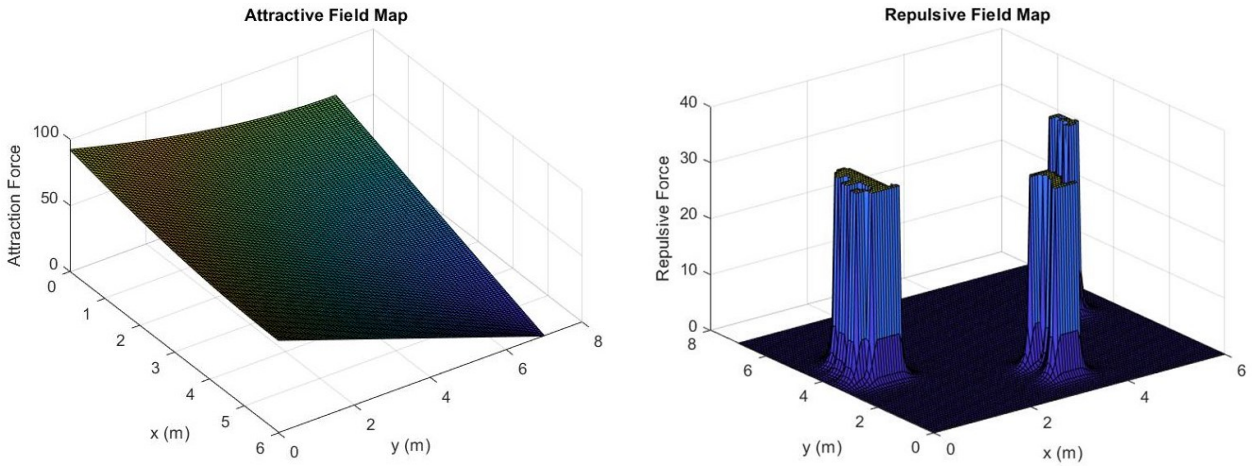


Figure 35 On the left is the attraction field. On the right is the repulsion field

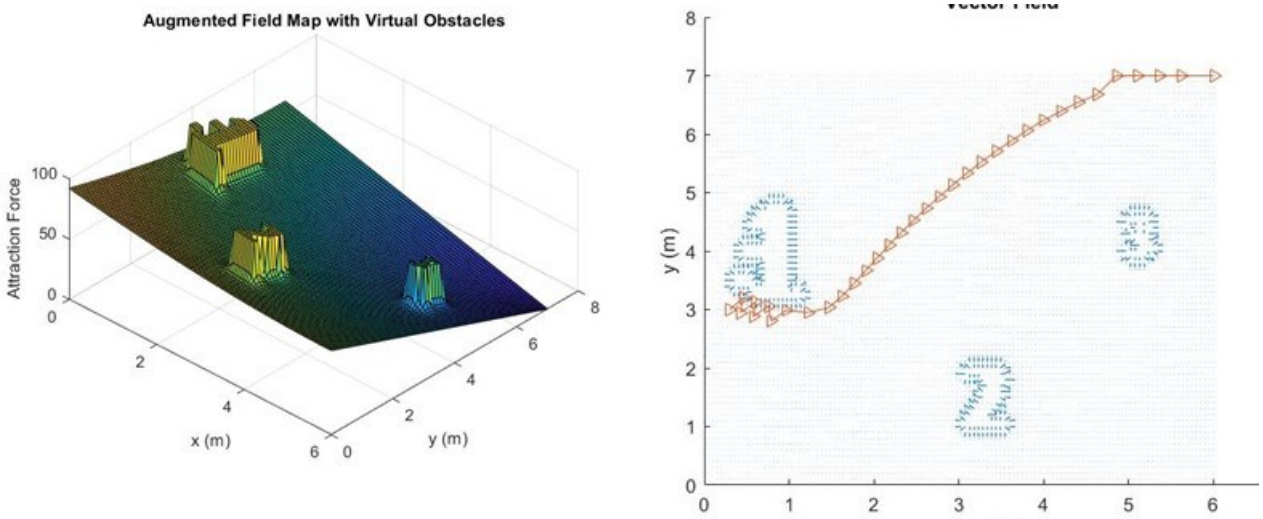


Figure 34 On the left is the augmented field with the virtual obstacles' repulsion add. On the right is the path of the robot on a vector field of map

Map 4

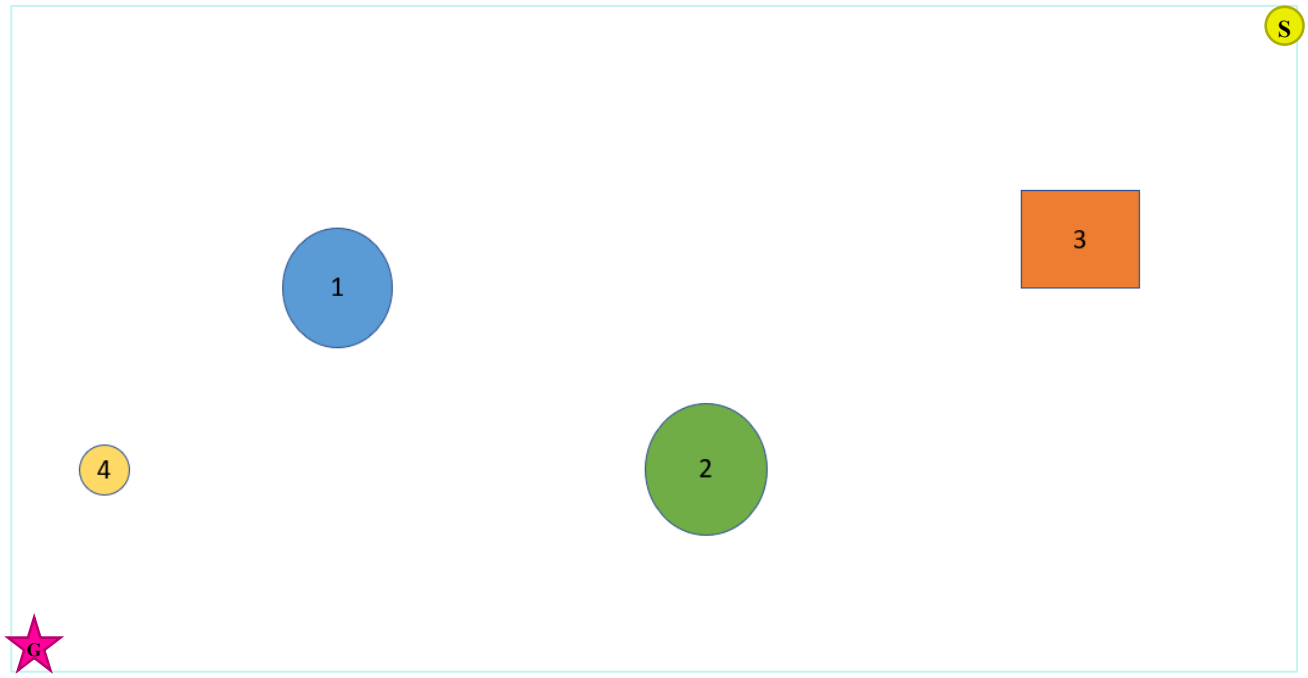


Figure 36 map 4 with start point at top right corner and goal at lower left corner.

MATLAB

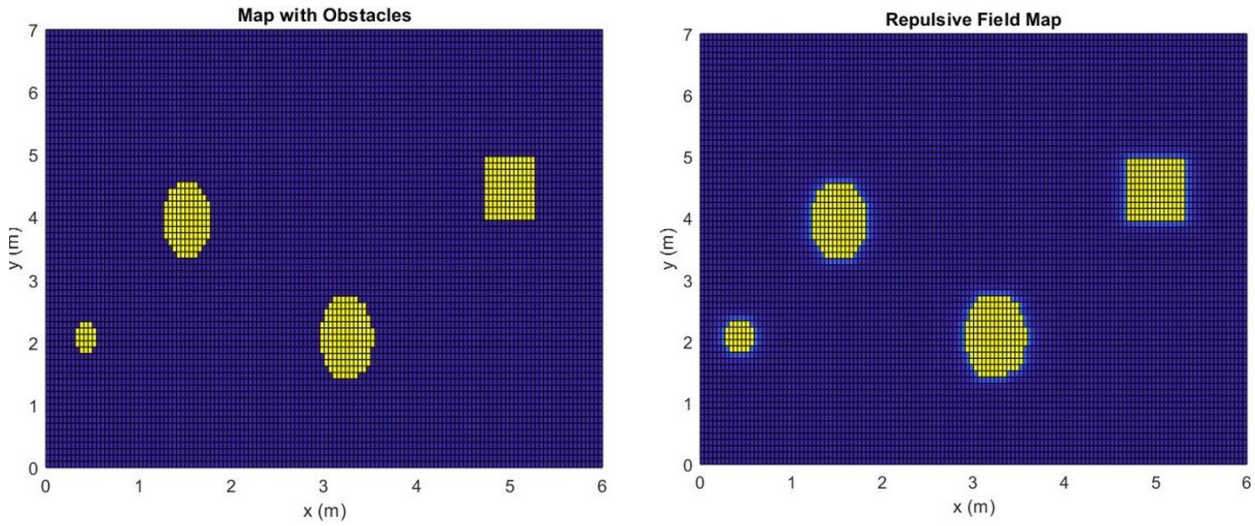


Figure 37 On the left is the map retrieved from the picture. On the right is a top view of the repulsive field. The blue hallow represents a fading repulsive force around the obstacle.

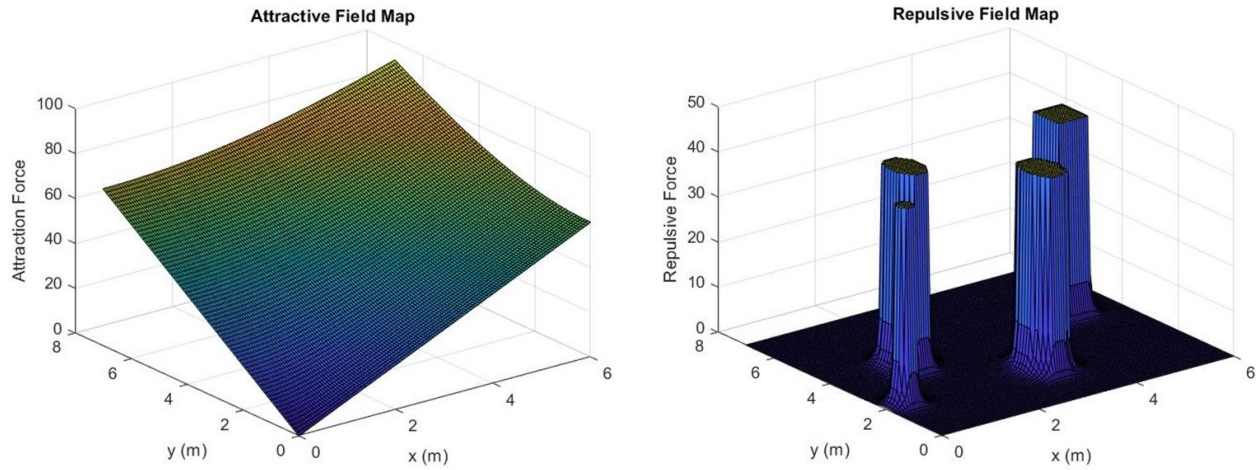


Figure 38 On the left is the attraction field. On the right is the repulsion field

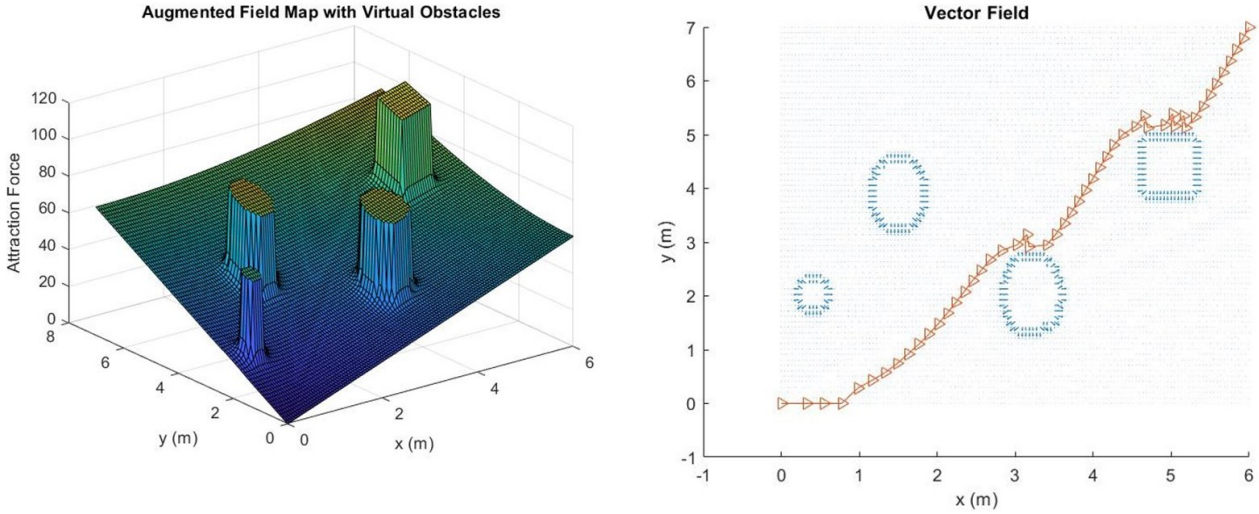


Figure 39 On the left is the augmented field with the virtual obstacles' repulsion add. On the right is the path of the robot on a vector field of map

Simulink

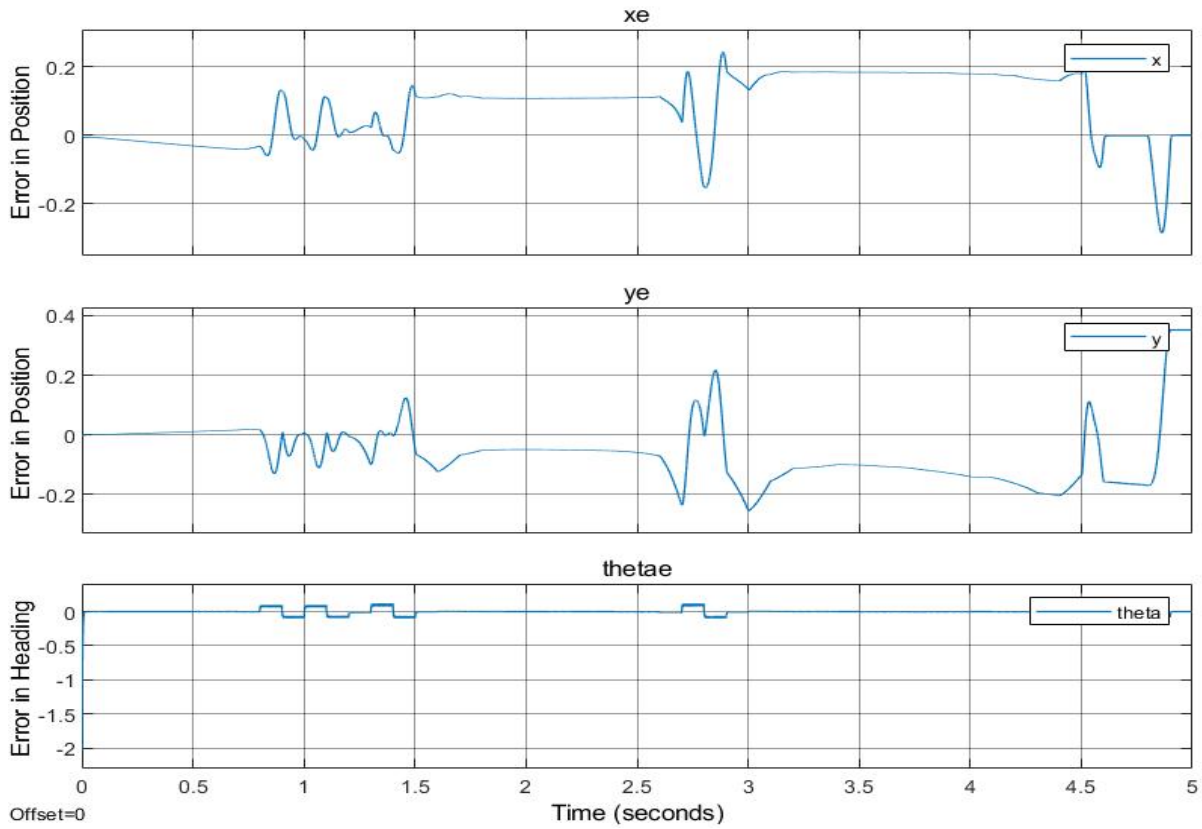


Figure 40 Error in states. k_x*=1000 and* k_θ*=500,* $T_s = 0.001$*, way points not interpolated.* $x_{e\_max} \approx 0.22$, $y_{e\_max} \approx 0.38$, $\theta_{e_{max}} \approx 0.05$

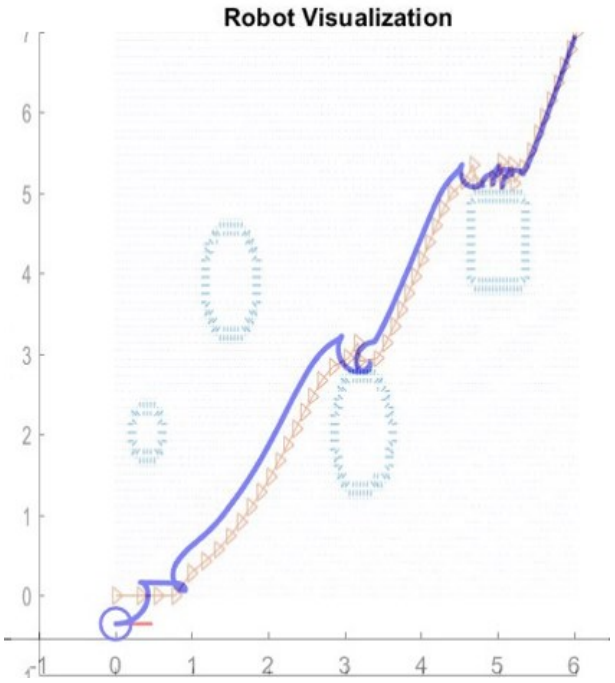


Figure 41 Lyupanov control output (blue) on input path (red)

The same analysis applied before is used here too to obtain the best gains.

	k_x	k_θ	$x_{e_{max}}$	$y_{e_{max}}$	$\theta_{e_{max}}$
D	1000	500	0.22	0.17	0.05

Kalman Filter

The linear Kalman filter was implemented on 2 paths using Lyupanov control.

Path 1

The path is taken from map 1 path.

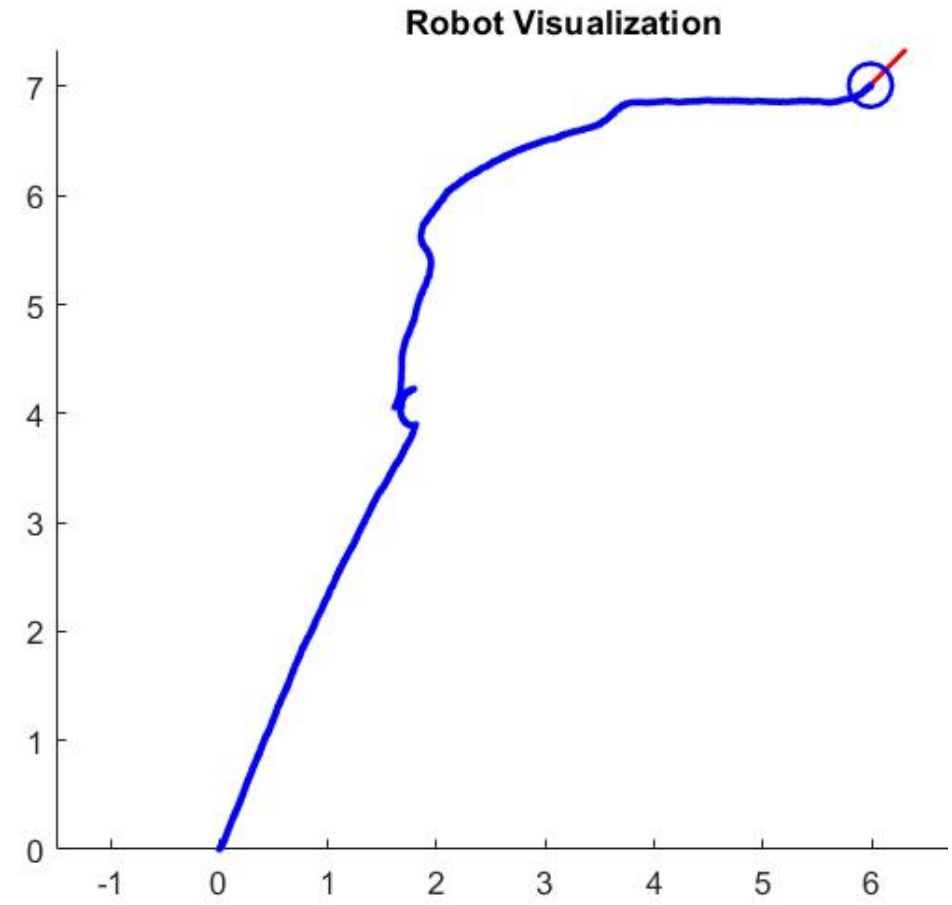


Figure 42 The path developed after Kalman filter.

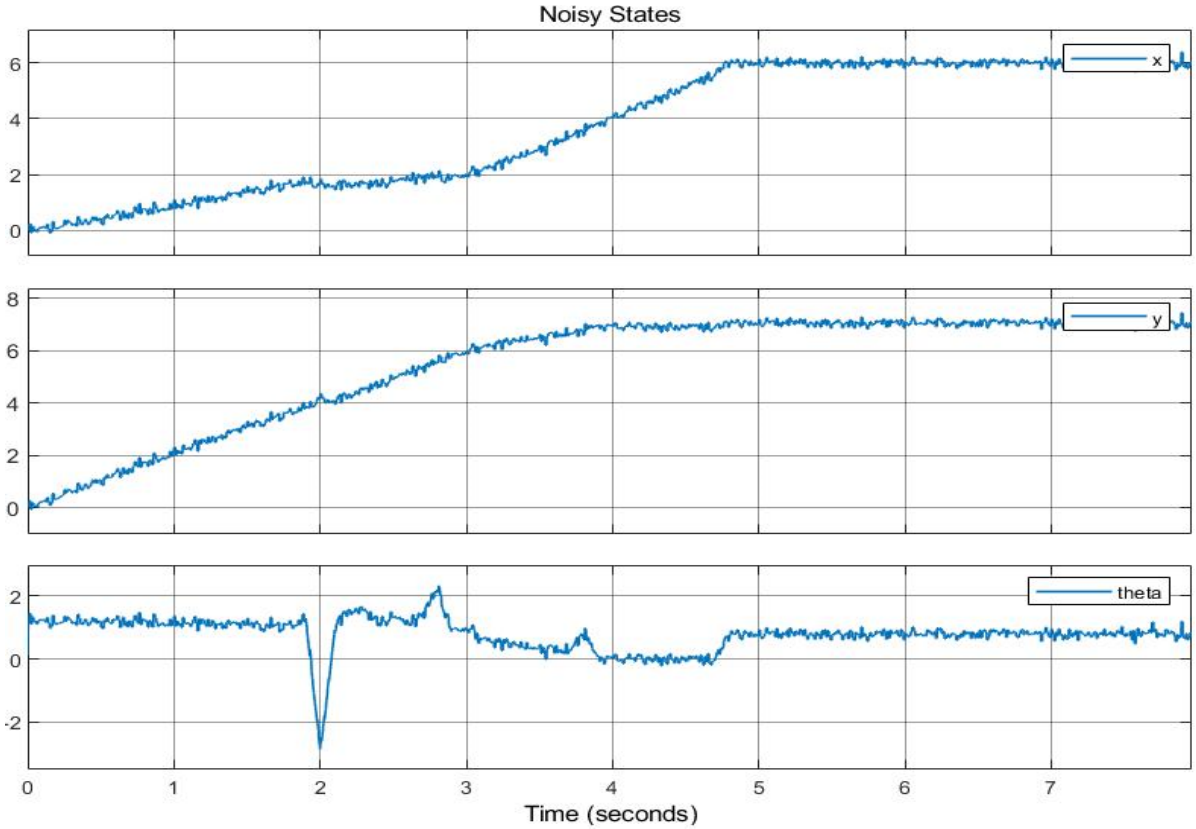


Figure 44 The states after adding white gaussian additive noise of power 0.0001.

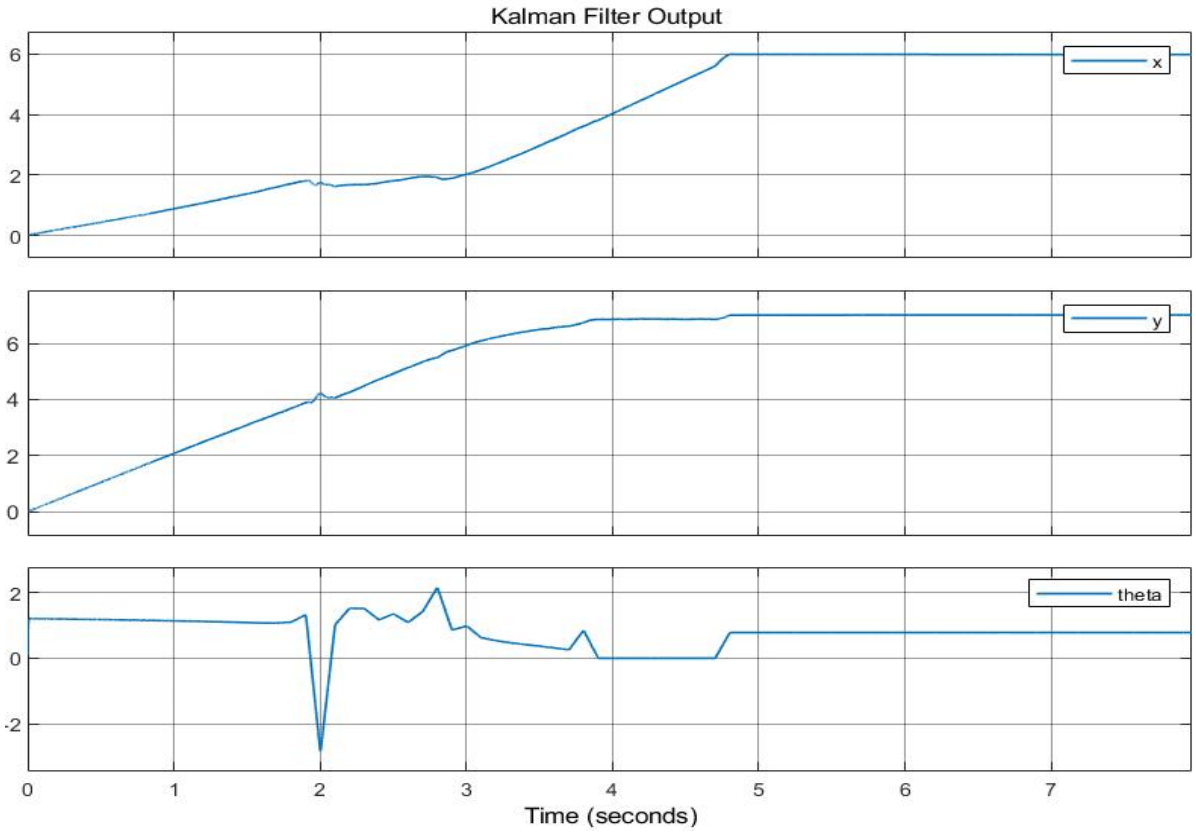


Figure 43 The states after correction by Kalman filter

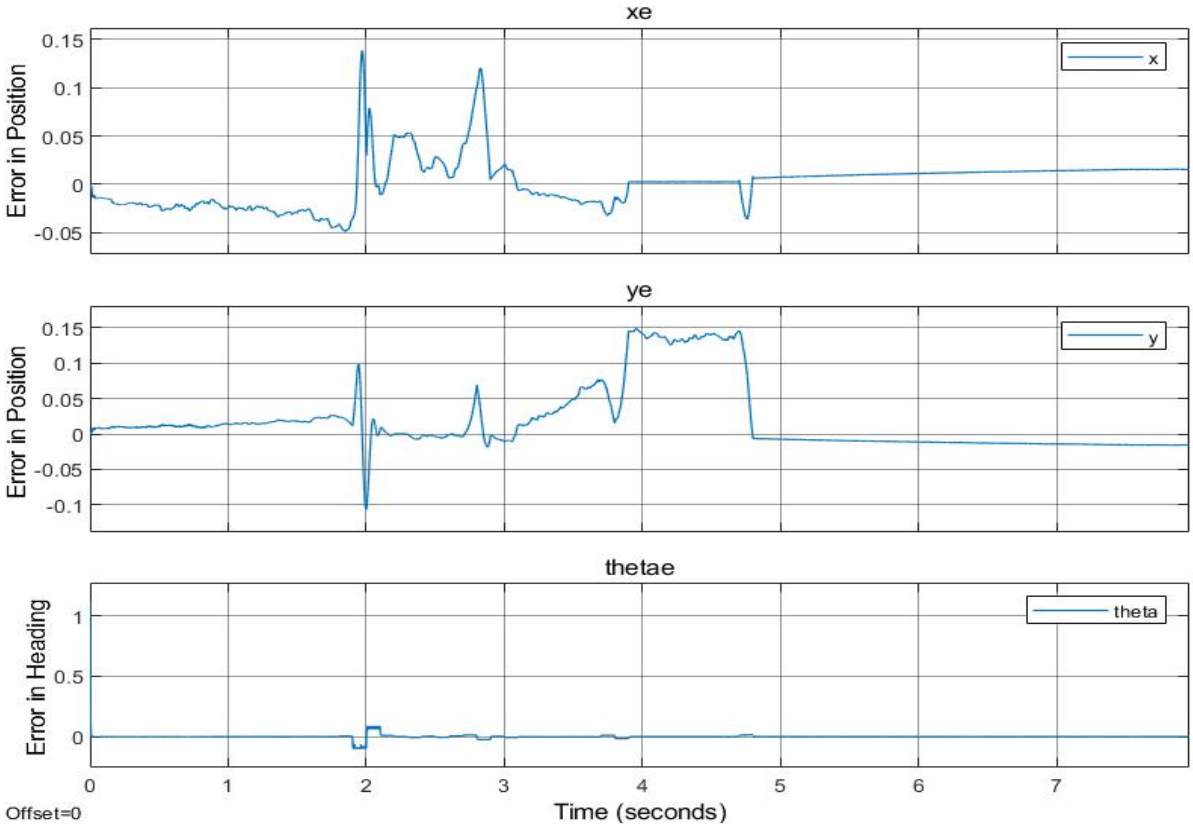


Figure 46 The error in states.

Path 2

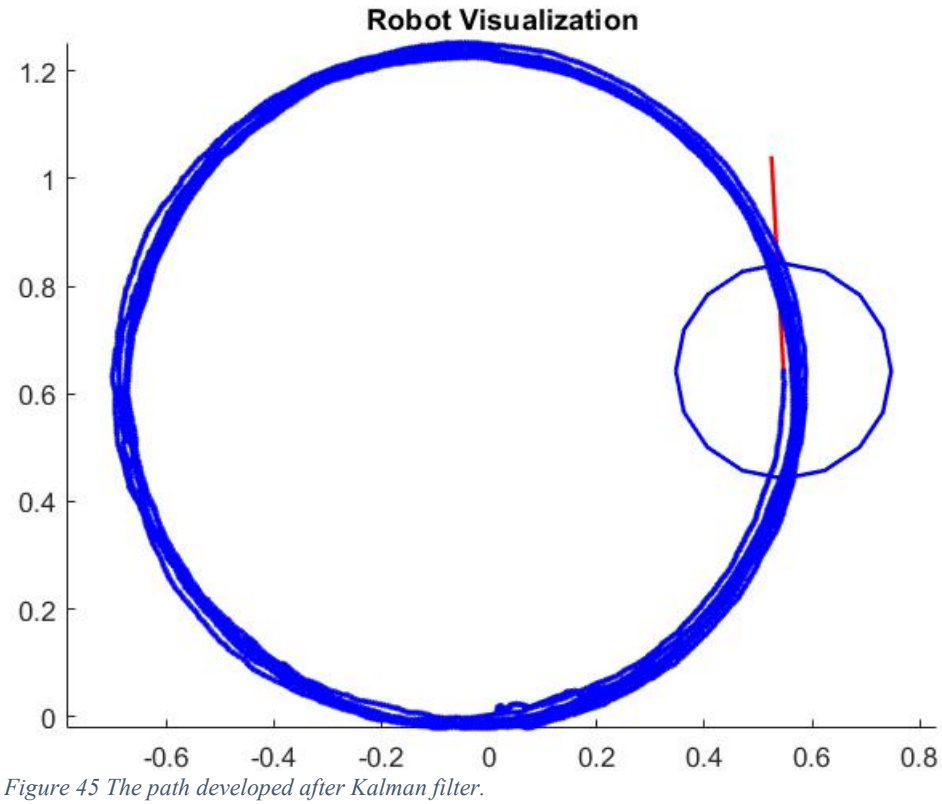


Figure 45 The path developed after Kalman filter.

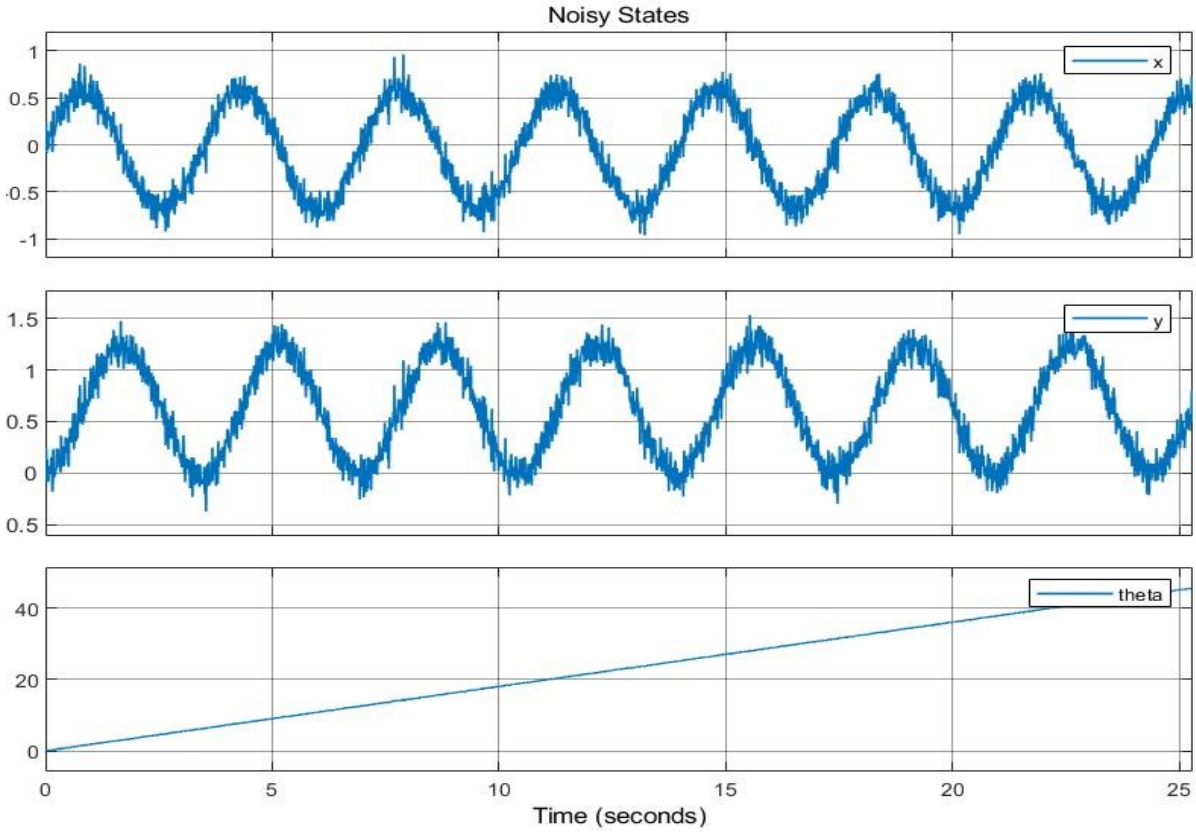


Figure 48 The states after adding white gaussian additive noise of power 0.0001.

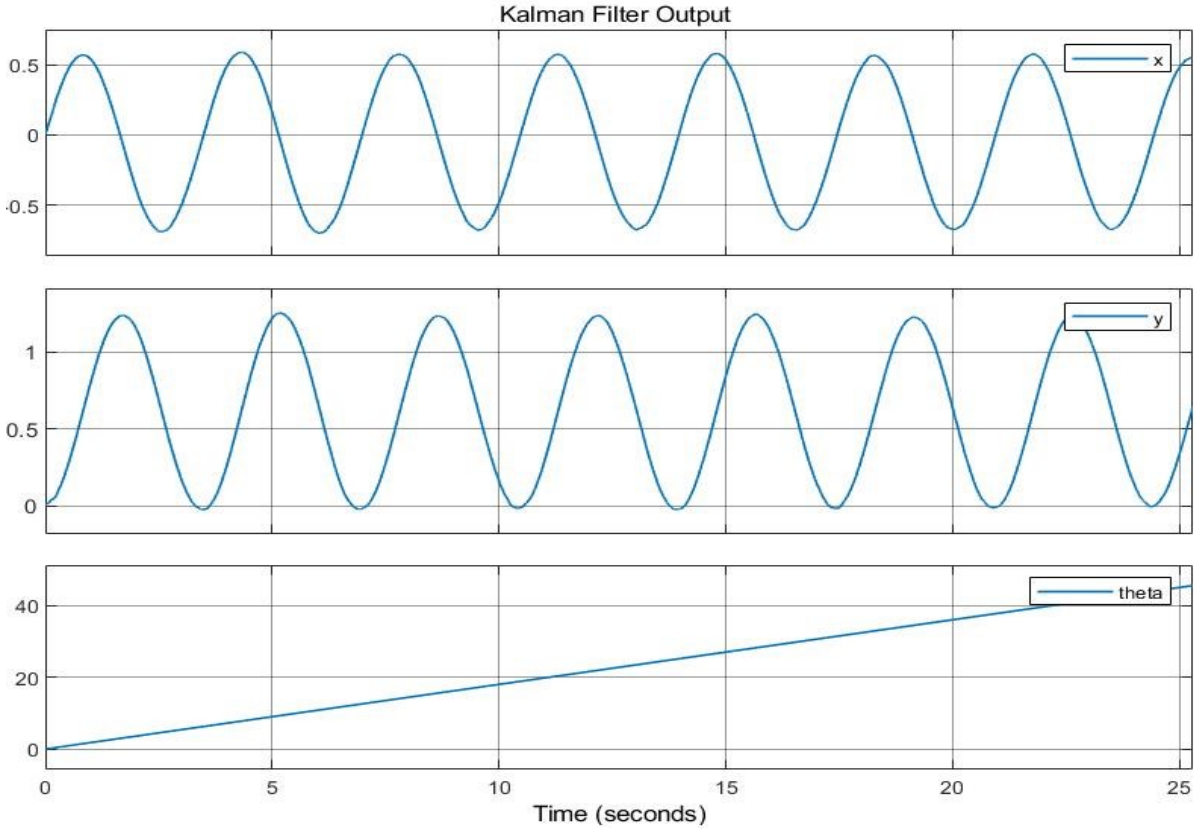


Figure 47 The states after correction by Kalman filter

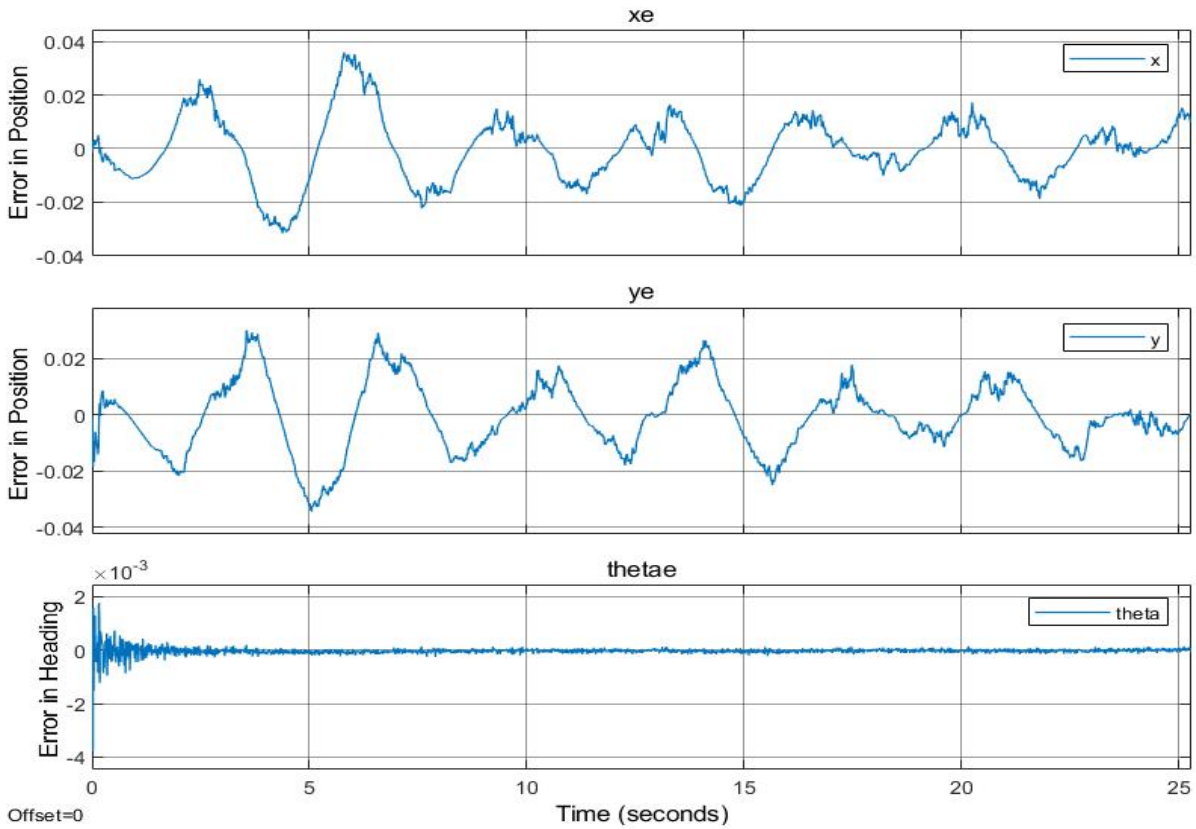


Figure 49 The error in states.

Appendix

Robot Dynamics Initialization Code

```
x0=0;
y0=0;
theta0=5*pi/180;
r_w=0.2; %20 cm
L=1; %1 m

xd=20; %in meter
yd=15;
thetad=45*pi/180;

xe0=xd-x0;
ye0=yd-y0;

rho0= (xe0^2+ye0^2)^(1/2);
alpha0=atan2(ye0,xe0);
beta0=-theta0-alpha0;

ini=[rho0; alpha0; beta0];
```

Path Planning Code

```
x0=0;
y0=0;
theta0=5*pi/180;
r_w=0.2; %20 cm
L=1; %30 cm

xd=6; %in meter
yd=7;
thetad=45*pi/180;

xe0=xd-x0;
ye0=yd-y0;

rho0= (xe0^2+ye0^2)^(1/2);
alpha0=atan2(ye0,xe0);
beta0=-theta0-alpha0;

ini=[rho0; alpha0; beta0];
```

```
%% setting map
%%map from 0 to 20 in x and y
I = imread('map5.png');
BW = im2bw(I,0.99); %% convert to b&w
BW = imresize(BW,1/8); %%compress image

viz= Visualizer2D;
viz.mapName='map';

grid = binaryOccupancyMap(BW);

%figure;
%imshow(BW)
[columns, rows] = size(BW);

xt=(xd-x0)/rows;
yt=(yd-y0)/columns;

x= x0:xt:xd; % discritizing domain
y= y0:yt:yd;

map=zeros(length(y),length(x));


rep2=zeros(length(y),length(x));

n=1;

% Loop over all locations  and deine obstacles
for col = 1:columns
    for row = 1:rows
        % color(row,col)=impixel( BW , row , col);
        if norm(impixel(BW , row , col) - [0,0,0])<1.5
            map(columns-col,row)=1;
            %index(n,:)=[col, row];
            obsi(n,:)=[columns-col,row];
            n=n+1;
        end
    end
end

map_v=map;

%% establishing fields
```

```
Katt=10;
Krep=0.1;
rlim=L;

[Uatt,Urep]=APF(columns, rows, x, y, Katt, Krep, obsi,rlim,1);
Nmap=Uatt+Urep;


[gx,gy] = gradient(Nmap);
figure;
surf(Nmap)
figure
quiver(-gx,-gy)
%surf(Nmap)
%% path
y1=y0;
x1=x0;
path(1,:)=[y1,x1]; %[3.7,1.6]; %(columns+2-col,row) (y,x)

i=1;
%% plot(path(:,2),path(:,1),'-o')
while i>0
    %% get pathhhhh


    py=floor(abs(path(i,1)/yt))+1;
    px=floor(abs(path(i,2)/xt))+1;

    G= [gy(py,px),gx(py,px)];
    G= G/norm(G); %%normalise the gradient

    a=xt*5; % speed, step length
    k=1;
    c=0;

    while k>0
        c=c+1;
        pos_new=path(i,:)-a*G;

        %% dont fall for gradients
        if abs(pos_new(1)-yd)<0.2 %any(pos_new<0)
            pos_new(1)=yd;

        elseif abs(pos_new(1)-yd)>0.2
            a=a*0.9;
        end
```

```
        if abs(pos_new(2)-xd)<0.2
            pos_new(2)=xd;

        elseif abs(pos_new(2)-xd)>0.2
            a=a*0.9;
        end

        %% check positivty
        if pos_new(1)<0
            a=a*0.9;
            continue;
        end

        if pos_new(2)<0
            a=a*0.9;
            continue;
        end

        pos_new_i=[floor(abs(pos_new(1)/yt))+1;
floor(abs(pos_new(2)/xt))+1];

        %% check remaining in domain
        if pos_new_i(1)<=columns+1 && pos_new_i(2) <=rows+1
            k=-1;
        end
    end

    path(i+1,:)=pos_new;
    rd=((path(i+1,1)-yd)^2+(path(i+1,2)-xd)^2)^0.5; %% don't
get stuck near goal

    %% dont get stuck
    if i>10 && abs(norm(path(i+1,:)-path(i+1-
10,:))/norm(path(i+1-10)))<0.1 && rd>1
        u=1;
        for col=-u:1:u %columns
            for row=-u:1:u %rows

obsi(length(obsi)+1,:)=[col+pos_new_i(1),row+pos_new_i(2)];
                map_v(col+pos_new_i(1),row+ pos_new_i(2))=1;
            end
        end

        %% repeat defining Urep and Nmap
        i=0;
```

```
        [Uatt, Urep]=APF(columns, rows, x, y, Katt, Krep,
obsi,rlim,i);
        Nmap=Uatt+Urep;
        [gx,gy] = gradient(Nmap);


         surf(x,y,Nmap)

    end

    if rd<0.1
        i=-100;
    end

    i=i+1;

end

%% Format Data to Simulink
for i =1: length(path)-1
    y1= path(i,1);
    x1= path(i,2);
    y2= path(i+1,1);
    x2= path(i+1,2);
    path(i,3)=atan2((y2-y1),(x2-x1));
end

%path(1,3)=theta0;

Ts=0.001;
path(length(path),3)=thetad;

time=(0:length(path)-1)/10;
xfun=[time' path(:,2)];
yfun=[time' path(:,1)];
thetafun=[time' path(:,3) ];

%% display Results
if xt<0
    x=flip(x);
end
if yt<0
   y=flip(y);
end

figure;
```

```
hold on
quiver(x,y,-gx,-gy)
plot(path(:,2),path(:,1),'->')
title("Vector Field");
xlabel('x (m)');
ylabel('y (m)');
hold off

figure;
surf(x,y,map);
title("Map with Obstacles");
xlabel('x (m)');
ylabel('y (m)');
zlabel('Attraction Force');

figure;
surf(x,y,Uatt)
title("Attractive Field Map");
xlabel('x (m)');
ylabel('y (m)');
zlabel('Attraction Force');

figure;
surf(x,y,Urep)
title("Repulsive Field Map");
xlabel('x (m)');
ylabel('y (m)');
zlabel('Repulsive Force');

figure;
surf(x,y,Nmap)
title("Augmented Field Map with Virtual Obstacles");
xlabel('x (m)');
ylabel('y (m)');
zlabel('Attraction Force');
```