Error Dynamics Control for TB3_OM

```
clear all; close all; clc;
load('odomdataburger.mat');
load('odomdata.mat');

%Initial Configuration
x0=0; y0=0; theta0=pi/4;
q(1,:)=[x0; y0; theta0];
%Define total time (see Events.xlsx)
tf_via=13.7990; %sec
tf=16.067; %sec
%Define initial desired configuration
xdes=-0.001; ydes=-0.001; thetades=pi/4;
qdes(1,:)=[xdes; ydes; thetades];

%Define final desired configurations
xdes_via_f=-2; ydes_via_f=1;
xdes_f=-1.5; ydes_f=3.75; thetades_f=pi/4;

%Define initial configuration error
error(1,:)=qdes(1,:)-q(1,:);
%Define via configuration error
qdes(2,:)=[-0.001; 0.999; 0.99*pi/4];

%Call ODE45 Solver, edot=A*e
[T,X] = ode45(@(t,x)errordynamics(t,x),[0 tf_via],error(1,:));
[T2,X2] = ode45(@(t,x)errordynamics(t,x),[tf_via tf],X(end,:));
T=[T;T2]; X=[X;X2];
```

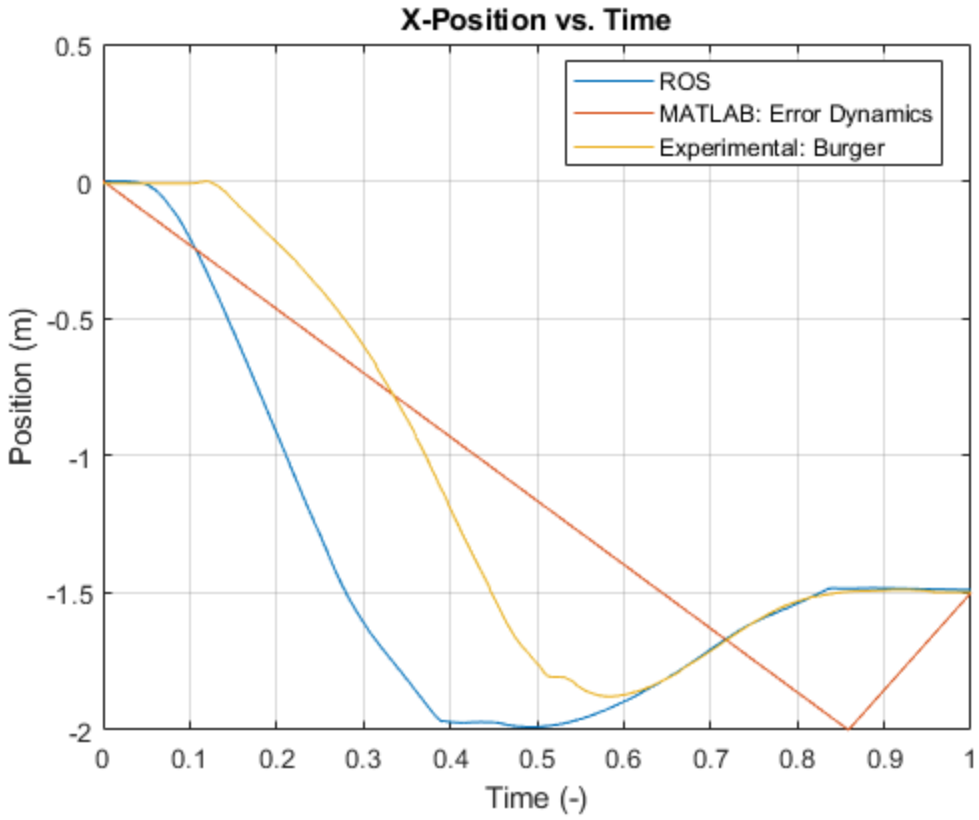
The output of the ode45 for edot=A*e is e! Therefore, e=X, so:

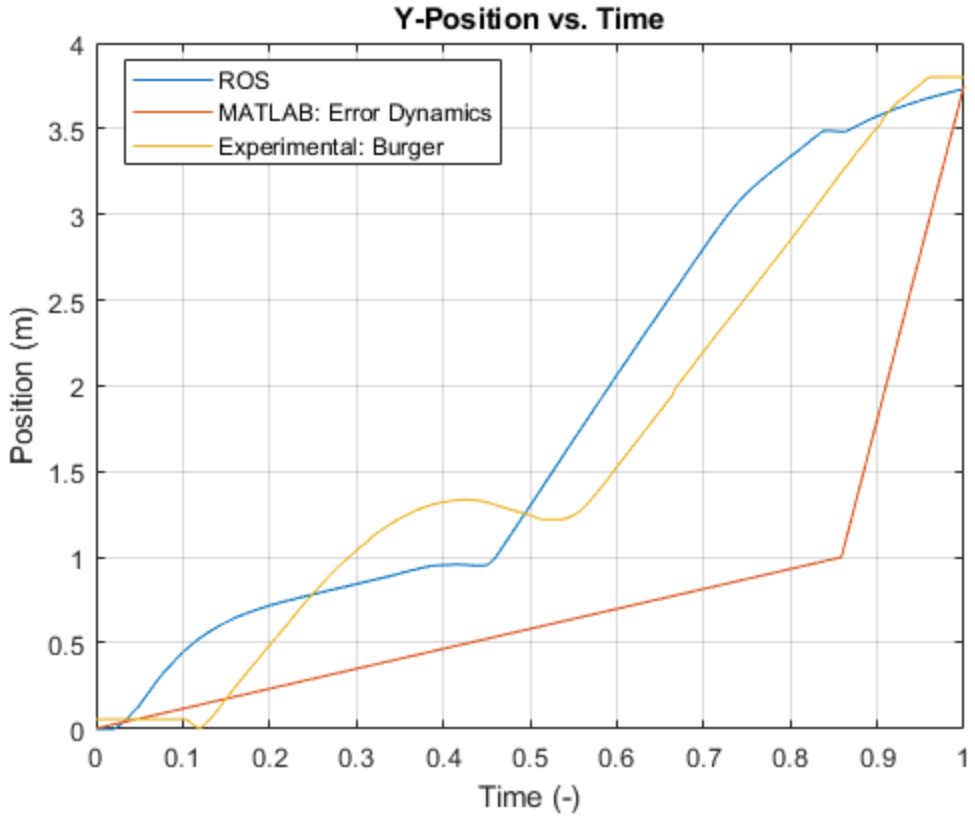
```
xerror=X(:,1); % error in x
yerror=X(:,2); % error in y
thetaerror=X(:,3); % error in theta
error_vec=[xerror, yerror, thetaerror]; % error vector

%Preallocate
qdes_vec=zeros(length(T),3);
vel_vec=zeros(length(T),2);

% Calculate the desired trajectory
for i=1:size(T)
    if T(i)<=tf_via
        xdes=xdes_via_f*(T(i)/tf_via); % COMING FROM THE (PLANNED)
 TRAJECTORY INFORMATION
        ydes=ydes_via_f*(T(i)/tf_via); % COMING FROM THE TRAJECTORY
 INFORMATION
    else
        xdes=xdes_via_f+(xdes_f-xdes_via_f)*((T(i)-tf_via)/(T(end)-
tf_via)); % COMING FROM THE (PLANNED) TRAJECTORY INFORMATION
        ydes=ydes_via_f+(ydes_f-ydes_via_f)*((T(i)-tf_via)/(T(end)-
tf_via)); % COMING FROM THE TRAJECTORY INFORMATION
```

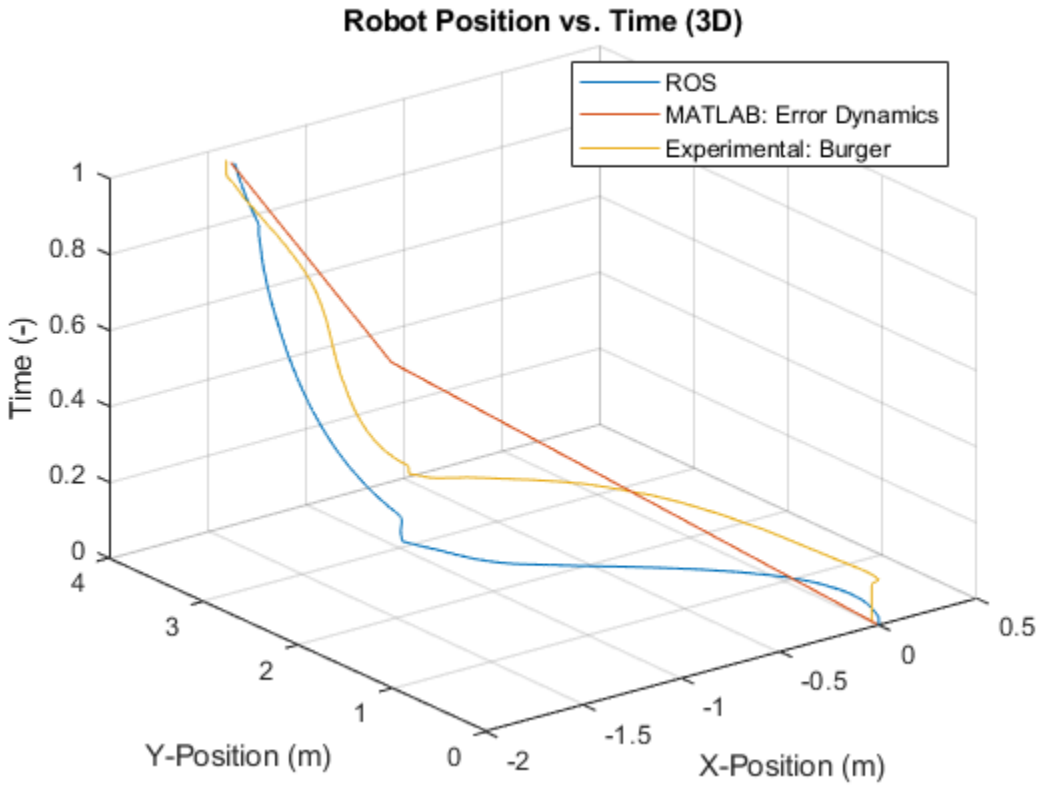
```
    end

    %Calculate orientation
    if i>1
        dt=T(i)-T(i-1);
        dxdes=(xdes-xdes_old)/dt; dydes=(ydes-ydes_old)/dt;
        tdes=atan2(dydes,dxdes);
        vel_vec(i,1)=sqrt(dxdes^2+dydes^2); %linear velocity
        vel_vec(i,2)=(tdes-tdes_old)/dt; %angular velocity
    else
        dxdes=0; %Differentiating
        dydes=0; %Differentiating
        tdes=pi/2;
        vel_vec(i,1)=0;
        vel_vec(i,2)=0;
    end

    qdes_vec(i,:)=[xdes, ydes, tdes]; % desired trajectory
    xdes_old=xdes; ydes_old=ydes; tdes_old=tdes;
end

q=qdes_vec-error_vec; % Actual configuration of the robot

% Data for plots
tt=0:1000000:1000000*length(odomdataburger.x)-1;
pos=[odomdataburger.x, odomdataburger.y, odomdataburger.z,
 1*ones(length(odomdataburger.x),1)]';
%Define homogeneous transform
H=computeRzh(pi*1.325)*computeTxh(-0.2521)*computeTyh(2.062);
%Rotate position data
for i=1:length(odomdataburger.x)
    pos(:,i)=H*pos(:,i);
end

% Plots Position vs. Time
t_odom=(odomdata.timensec(1:897)-odomdata.timensec(1)*ones(897,1))/
max(odomdata.timensec(1:897)-odomdata.timensec(1)*ones(897,1));
figure('Name','X-Position vs. Time');
plot(t_odom, odomdata.x(1:897), ... %ROS
     T/(max(T)), q(:,1), ...  %MATLAB
     tt/(max(tt)), pos(1,:)); %Experimental: Burger
hold on;
xlabel('Time (-)');
ylabel('Position (m)');
legend('ROS','MATLAB: Error Dynamics', 'Experimental:
 Burger','Location','Best');
title('X-Position vs. Time');
grid on; hold on;

figure('Name','Y-Position vs. Time');
plot(t_odom, odomdata.y(1:897), ... %ROS
     T/(max(T)), q(:,2), ...  %MATLAB
     tt/(max(tt)), pos(2,:)); %Experimental: Burger
xlabel('Time (-)');
```

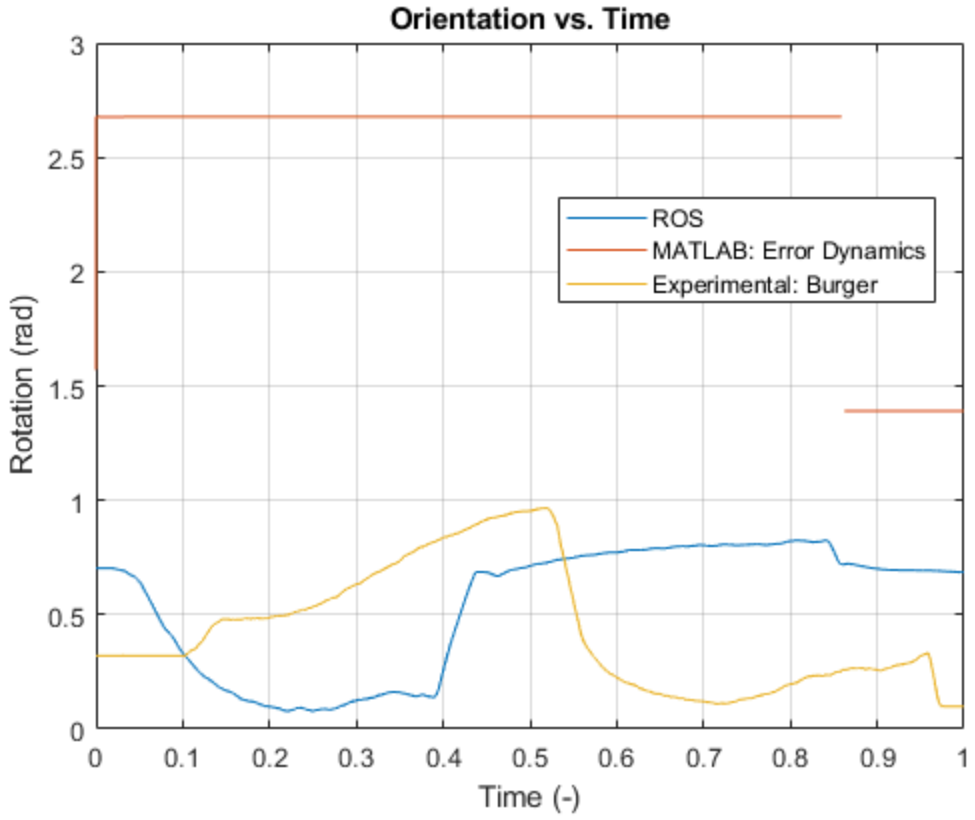
```
ylabel('Position (m)');
legend('ROS','MATLAB: Error Dynamics', 'Experimental:
 Burger','Location','Best');
title('Y-Position vs. Time');
grid on;

figure('Name','Robot Position vs. Time (3D)');
plot3(odomdata.x(1:897), odomdata.y(1:897), t_odom,... %ROS
      q(:,1), q(:,2), T/(max(T)), ...  %MATLAB
      pos(1,:), pos(2,:), tt/(max(tt))); %Experimental: Burger
hold on;
xlabel('X-Position (m)'); ylabel('Y-Position (m)'); zlabel('Time
 (-)');
legend('ROS','MATLAB: Error Dynamics', 'Experimental:
 Burger','Location','Best');
title('Robot Position vs. Time (3D)');
grid on;

%Plots Orientation vs. Time
figure('Name','Rotation vs. Time');
plot(t_odom, odomdata.Worientation(1:897), ... %ROS
     T/(max(T)), q(:,3), ...  %MATLAB
     tt/(max(tt)), odomdataburger.Worientation); %Experimental: Burger
xlabel('Time (-)');
ylabel('Rotation (rad)');
legend('ROS','MATLAB: Error Dynamics', 'Experimental:
 Burger','Location','Best');
title('Orientation vs. Time');
grid on;

% Plot Mobile Robot Position (ROS)
figure('Name','Mobile Robot Position');
plot(odomdata.x(1:938), odomdata.y(1:938), ...
     q(:,1), q(:,2), ...
     pos(1,:), pos(2,:));
hold on; grid on;
title('Mobile Robot Position');
xlabel('X (m)');ylabel('Y (m)');
legend('ROS','MATLAB: Error Dynamics', 'Experimental:
 Burger','Location','Best');

% Plots Linear Velocity vs. Time
t_odom=(odomdata.timensec(1:897)-odomdata.timensec(1)*ones(897,1))/
max(odomdata.timensec(1:897)-odomdata.timensec(1)*ones(897,1));
figure('Name','Linear Velocity vs. Time');
plot(t_odom, odomdata.Zvelocity(1:897), ... %ROS
     T/(max(T)), vel_vec(:,1), ...  %MATLAB
     tt/(max(tt)), odomdataburger.Zvelocity); %Experimental: Burger
hold on;
xlabel('Time (-)');
ylabel('Linear Velocity (m/s)');
legend('ROS','MATLAB: Error Dynamics', 'Experimental:
 Burger','Location','Best');
title('Linear Velocity vs. Time');
```

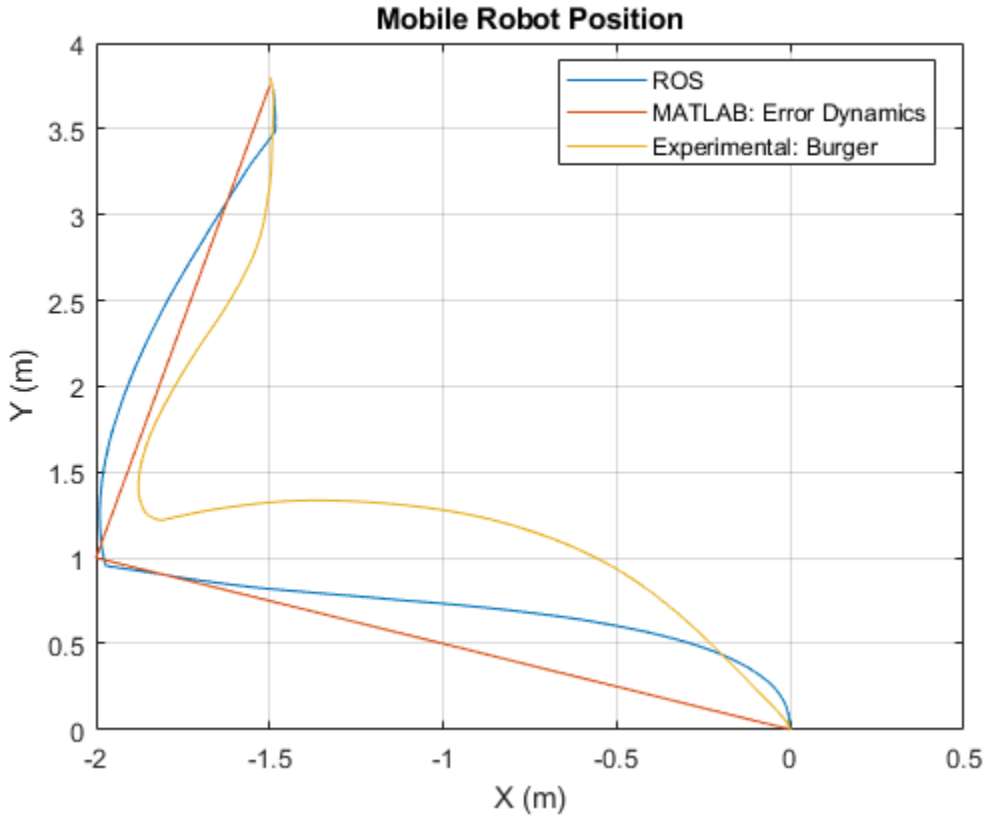
```
grid on; hold on;

% Plots Angular Velocity vs. Time
figure('Name','Angular Velocity vs. Time');
plot(t_odom, odomdata.Xvelorientation(1:897), ... %ROS
     T/(max(T)), vel_vec(:,2), ...  %MATLAB
     tt/(max(tt)), odomdataburger.Xvelorientation); %Experimental:
 Burger
xlabel('Time (-)'); ylabel('Angular Velocity (rad/s)'); ylim([-0.5
 0.5])
legend('ROS','MATLAB: Error Dynamics', 'Experimental:
 Burger','Location','Best');
title('Angular Velocity vs. Time');
grid on;
```

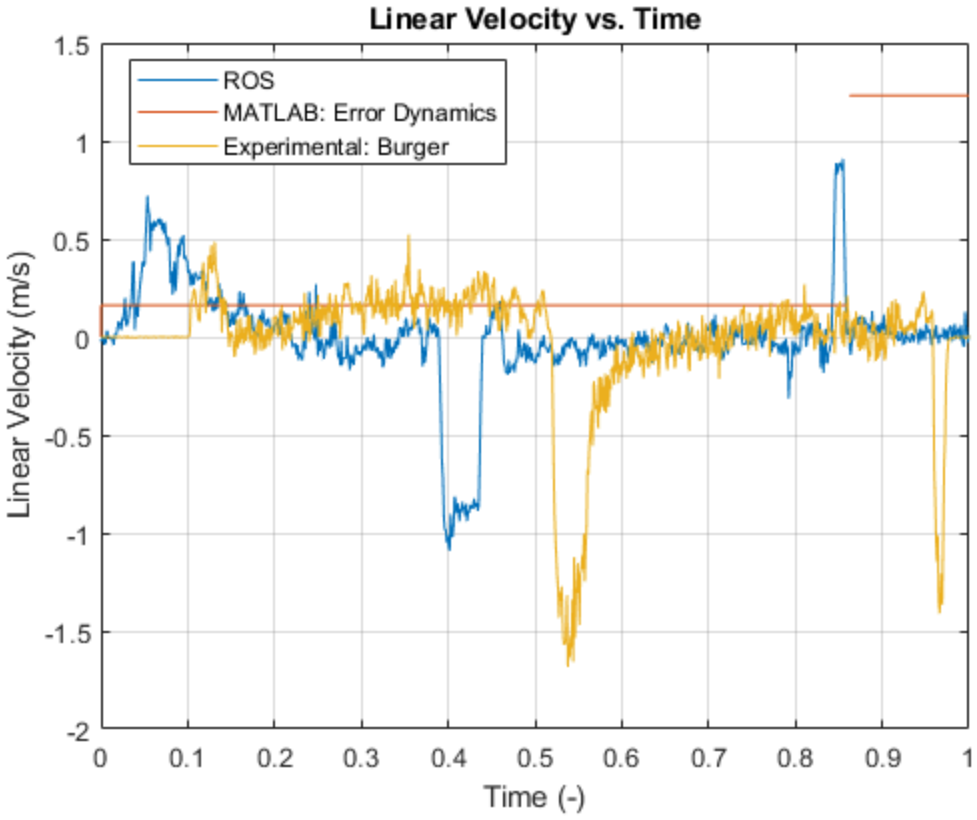


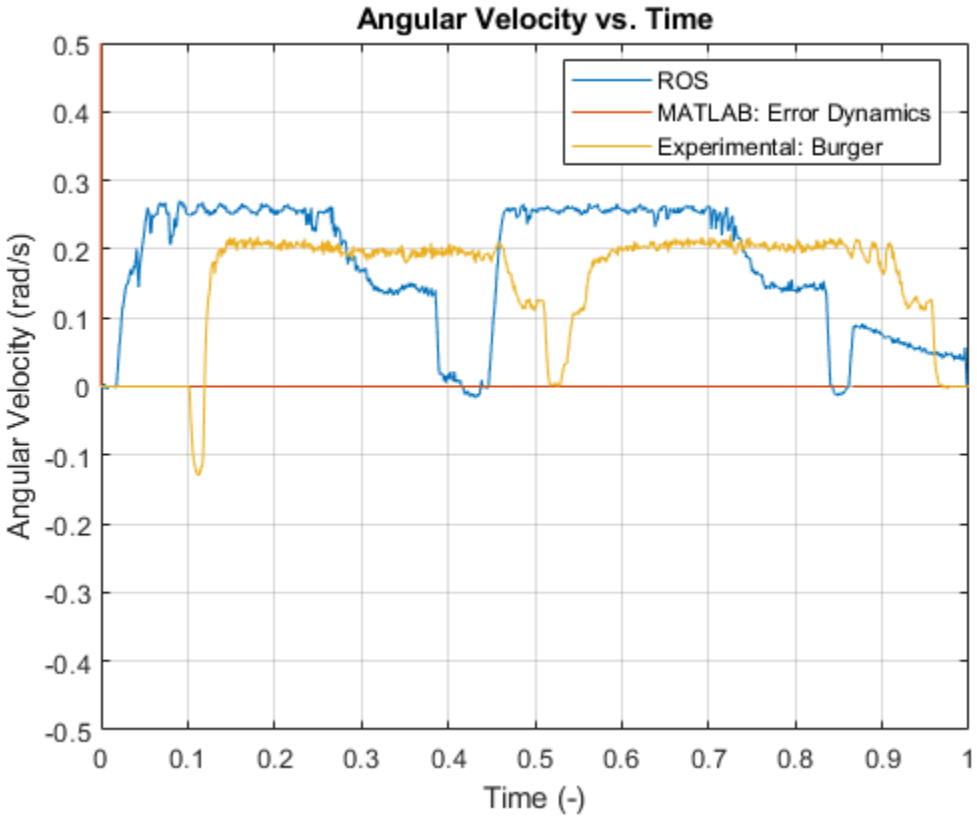

Y-Position vs. Time
ROS
MATLAB: Error Dynamics
Experimental: Burger
Position (m)
Time (-)
0
0.5
1
1.5
2
2.5
3
3.5
4
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Robot Position vs. Time (3D)
ROS
MATLAB: Error Dynamics
Experimental: Burger
Time (-)
Y-Position (m)
X-Position (m)
0
0.2
0.4
0.6
0.8
1
4
3
2
1
0
-2
-1.5
-1
-0.5
0
0.5


Orientation vs. Time
Rotation (rad)
Time (-)
ROS
MATLAB: Error Dynamics
Experimental: Burger
0
0.5
1
1.5
2
2.5
3
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


Mobile Robot Position
Y (m)
X (m)
ROS
MATLAB: Error Dynamics
Experimental: Burger
0
0.5
1
1.5
2
2.5
3
3.5
4
-2
-1.5
-1
-0.5
0
0.5


Linear Velocity vs. Time
ROS
MATLAB: Error Dynamics
Experimental: Burger
Linear Velocity (m/s)
Time (-)
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Angular Velocity vs. Time
ROS
MATLAB: Error Dynamics
Experimental: Burger
Angular Velocity (rad/s)
Time (-)
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

```
function [dx]=errordynamics(t,x)
    % Control based on Approximate Linearization
    % Define System Parameters per L10P2, P.4 (Modified for optimal
    % trajectory tracking)
    dr=0.8; % damping ratio (0.7).  Changed to 0.8.
    omegad=1/3; % desired steering velocity
    vd=1; % desired linear velocity
    nf=5; % natural frequency (1).  Changed to 5.
    % Define Gains per 11.69
    k1=2*dr*nf; % controller gain
    k3=k1; % controller gain
    k2=(nf^2-omegad^2)/vd; % controller gain

    % Equation 11.68, Siciliano, State-Space Format
    dx=zeros(3,1); %dx=edot
    dx(1) = -k1*x(1)+omegad*x(2);
    dx(2) = -omegad*x(1)+vd*x(3);
    dx(3) = -k2*x(2)-k3*x(3);
end

function [Rxh] = computeRxh(theta)
% Compute Basic Homogeneous Transform Matrix for
% rotation of theta (radians) about x-axis
Rxh=[1 0 0 0; ...
     0 cos(theta) -sin(theta) 0; ...
     0 sin(theta) cos(theta) 0; ...
     0 0 0 1;];
end
% -----------------------
% -----------------------
function [Ryh] = computeRyh(theta)
% Compute Basic Homogeneous Transform Matrix for
% rotation of theta (radians) about y-axis
Ryh=[cos(theta) 0 sin(theta) 0; ...
     0 1 0 0; ...
     -sin(theta) 0 cos(theta) 0; ...
     0 0 0 1];
end
% -----------------------
% -----------------------
function [Rzh] = computeRzh(theta)
% Compute Basic Homogeneous Transform Matrix for
% rotation of theta (radians) about z-axis
Rzh=[cos(theta) -sin(theta) 0 0; ...
     sin(theta) cos(theta) 0 0; ...
     0 0 1 0; ...
     0 0 0 1];
end
% -----------------------
% -----------------------
function [Txh] = computeTxh(d)
% ------------------------------------
% Calculate Basic Homogeneous Transform Matrix for
% translation of d (m) along x-axis
```

```
Txh=[1 0 0 d; 0 1 0 0; 0 0 1 0; 0 0 0 1];
end
% -----------------------
% -----------------------
function [Tyh] = computeTyh(d)
% -------------------------------------
% Calculate Basic Homogeneous Transform Matrix for
% translation of d (m) along y-axis
Tyh=[1 0 0 0; 0 1 0 d; 0 0 1 0; 0 0 0 1];
end
% -----------------------
% -----------------------
function [Tzh] = computeTzh(d)
% -------------------------------------
% Calculate Basic Homogeneous Transform Matrix for
% translation of d (m) along z-axis
Tzh=[1 0 0 0; 0 1 0 0; 0 0 1 d; 0 0 0 1];
end
```

Published with MATLAB® R2018b