This is a model identification exercise for Clemson Clearpath Husky. The model is to be of moderate fidelity such that it can be used for Husky control tasks. Following papers were refered for boot strapping.

[1] Yu, W., Chuy, O., Collins, E., & Hollis, P. (2009). *Dynamic Modeling of a Skid-Steered Wheeled Vehicle with Experimental Verification*. https://doi.org/10.1109/IROS.2009.5354381

[2] Kozłowski, K., & Pazderski, D. (2004). Modeling and Control of a 4-wheel Skid-steering Mobile Robot. In *Modeling and Control of a 4-wheel Skid-steering Mobile Robot* (Vol. 14, Issue 4, pp. 477–496).

[3] Caracciolo, L., De Luca, A., & Iannitti, S. (1999). Trajectory tracking control of a four-wheel differentially driven mobile robot. *Proceedings - IEEE International Conference on Robotics and Automation*, *4*(May), 2632–2638. https://doi.org/10.1109/robot.1999.773994

Extract Messages from relvant ROSBAG

```
clc
clear

bag = rosbag('temp\2023-02-16-15-51-57.bag');
clc

%% Parse Bag for relevant topics

piksi.navast_bf = select(bag,'Topic','/piksi_multi_position/navsatfix_best_fix'); %These topics
%piksi.navast_bf_hd = select(bag,'Topic','/piksi_multi_heading/navsatfix_best_fix');
%piksi.enu_pose_bf = select(bag,'Topic','/piksi_multi_position/enu_pose_best_fix'); %These topi
gx5.imu_data = select(bag,'Topic','/gx5/imu/data'); % IMU data from gx5
gx5.mag = select(bag,'Topic','/gx5/mag'); % Magnetometer data from gx5
husky.command_vel = select(bag,'Topic','/husky_velocity_controller/cmd_vel');
husky.status = select(bag,'Topic','/status'); % internal status information (motor currents)
husky.joint_states = select(bag,'Topic','/joint_states'); % Wheel encoder information
%}

msgStructs.piksi.navast_bf = readMessages(piksi.navast_bf,'DataFormat','struct');
%msgStructs.piksi.navast_bf_hd = readMessages(piksi.navast_bf_hd,'DataFormat','struct');
%msgStructs.piksi.enu_pose_bf = readMessages(piksi.enu_pose_bf,'DataFormat','struct');
msgStructs.gx5.imu_data = readMessages(gx5.imu_data,'DataFormat','struct');
msgStructs.gx5.mag = readMessages(gx5.mag,'DataFormat','struct');
msgStructs.cmd_vel = readMessages(husky.command_vel,'DataFormat','struct');
msgStructs.status = readMessages(husky.status,'DataFormat','struct');
msgStructs.joint_states = readMessages(husky.joint_states,'DataFormat','struct');
%}

clc

%save('data.mat')
%load('data.mat')
```

Kinematic Model : (from [1])

$$\begin{bmatrix} v_y \\ \dot{\varphi} \end{bmatrix} = \frac{r}{\alpha B} \begin{bmatrix} \frac{\alpha B}{2} & \frac{\alpha B}{2} \\ -1 & 1 \end{bmatrix} \begin{bmatrix} \omega_l \\ \omega_r \end{bmatrix}$$

α is a parameter that needs to be experimentally identified. It is dependent on friction. ω can be calculated from wheel odometry. $v_y \text{ and } \dot{\phi}$ can be estimated from the IMU readings. $B \text{ and } r$ are geometric constants. Instead of getting jusy one value for alpha, the idea is to create a look up table for various values of $v_y \text{ and } \dot{\phi}$

This model will then be used for RL training.

From the IMU data, find all available v and phi_dotThese linear velocities and angular velocities are realized values and different from the command velocities

```
for i =1:length(msgStructs.gx5.imu_data)
    lin_acc(i) = msgStructs.gx5.imu_data{i, 1}.LinearAcceleration.X;
    mean_lin_acc = sum(lin_acc)/length(lin_acc);
    ang_vel(i) = msgStructs.gx5.imu_data{i, 1}.AngularVelocity.Z;
end

% compute frequency of IMU messages
freq.IMU = (gx5.imu_data.EndTime  -gx5.imu_data.StartTime)/gx5.imu_data.NumMessages; % value ir

lin_vel(1) =0;
for i = 1:length(lin_acc)
    lin_vel(i+1) = lin_vel(i)+lin_acc(i)*1e-4;
end
```

Command velocity (V and w to right and left wheel velocity conversion)

if angular velocity is positive, the vehicle is turning left, so w_l >

w_r and vice versa

wheel radius r = 0.1650 m

track width B = 0.555 m

```
r = 0.1650;
B = 0.555;

freq.cmd_vel = (husky.command_vel.EndTime-husky.command_vel.StartTime)/husky.command_vel.NumMes

for i = 1:length(msgStructs.cmd_vel)
    V(i) = msgStructs.cmd_vel{i, 1}.Linear.X;
    omega(i) = msgStructs.cmd_vel{i, 1}.Angular.Z;
end
```

```
w_r = (V/r) + (B*omega)/(2*r);
w_l = (V/r) - (B*omega)/(2*r);
```

From the kinematic model:

[v_hat phi_dot_hat]' = A * [w_r w_l]';

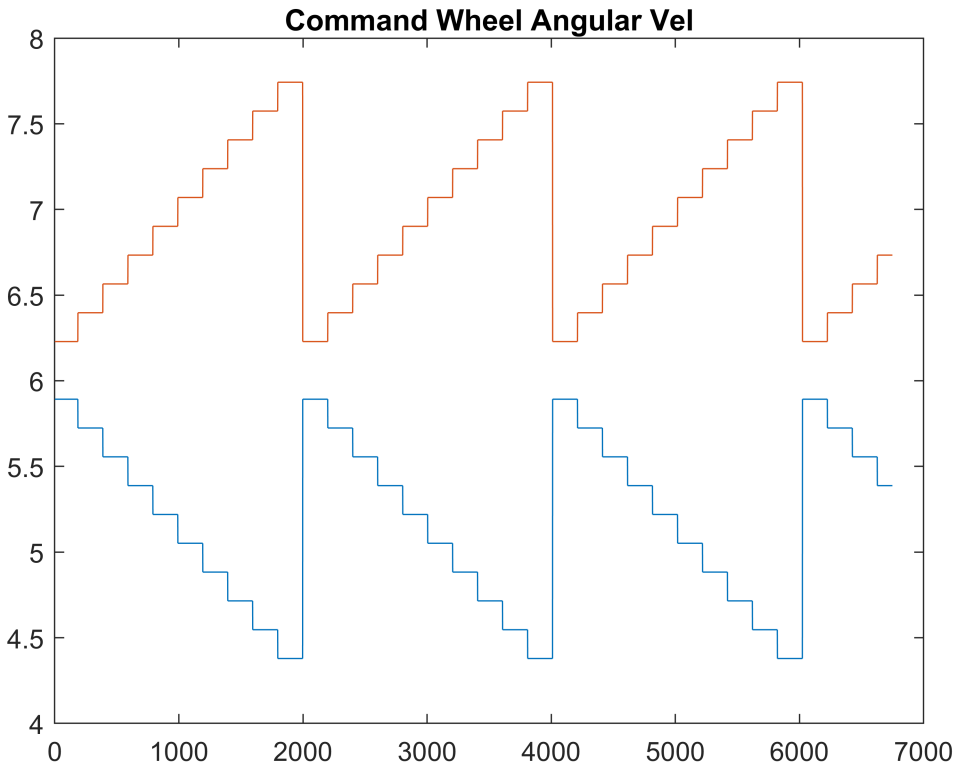
where A = [r/2 r/2; -r/alpha*B r/alpha*B]

alpha is identification param and B is track width

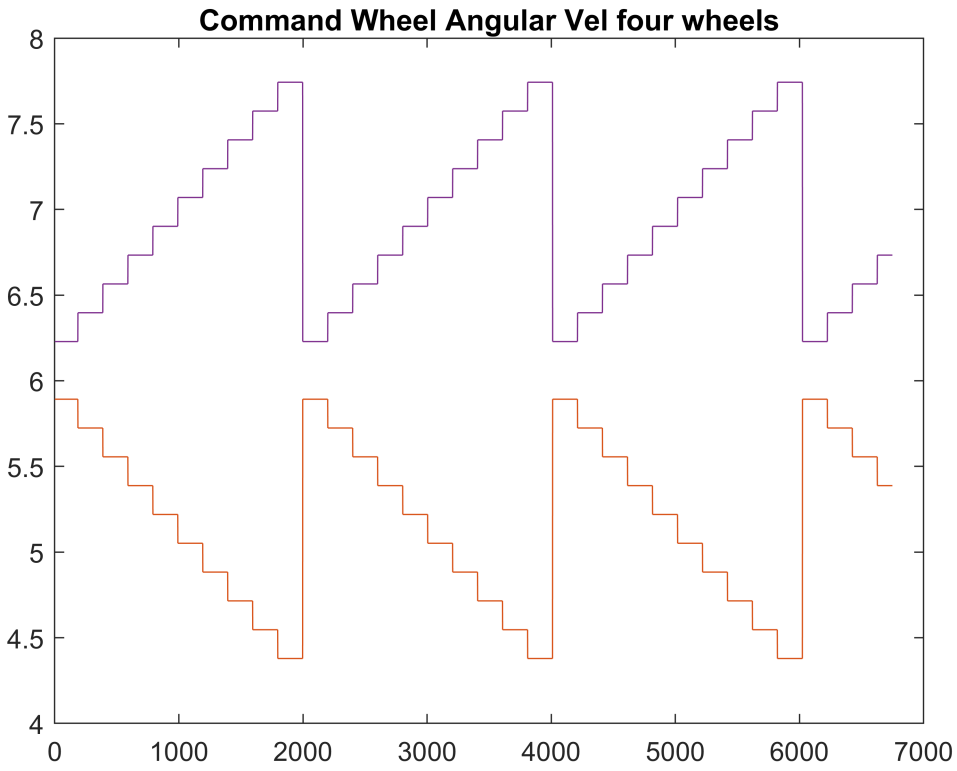
```
% for initialization guess assume alpha is 1.8
alpha = 2;

A = [r/2 r/2;-r/(alpha*B) r/(alpha*B)];

[vel_hat] = A*[w_r;w_l];


%% Four wheel variation
alpha_f = 2;
A_four = [r/4 r/4 r/4 r/4;-r/(2*alpha_f*B) -r/(2*alpha_f*B) r/(2*alpha_f*B) r/(2*alpha_f*B)];

w_rf=w_r;
w_rr=w_r;
w_lf=w_l;
w_lr=w_l;

[vel_hat_f] = A_four*[w_rf;w_rr;w_lf;w_lr];
```

```
%% Position from both the velocities

% Position from model
%x_hat = x_hat(i)
```

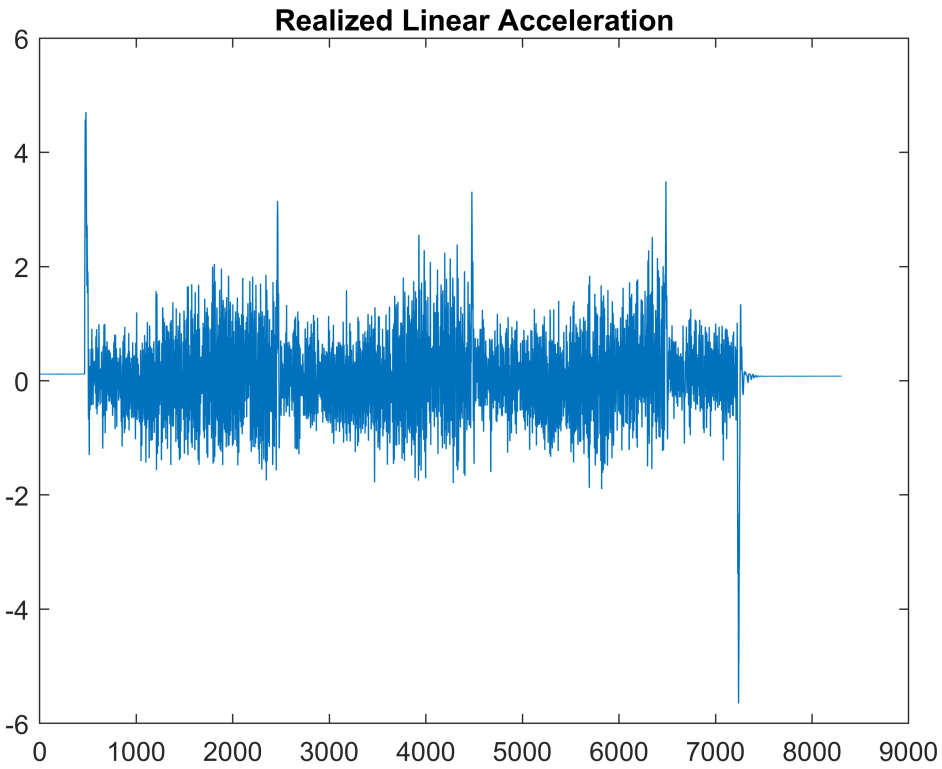
```
%% compare predicted and realized velocity profiles

figure
plot(w_l)
hold on
plot(w_r)
title('Command Wheel Angular Vel')
```



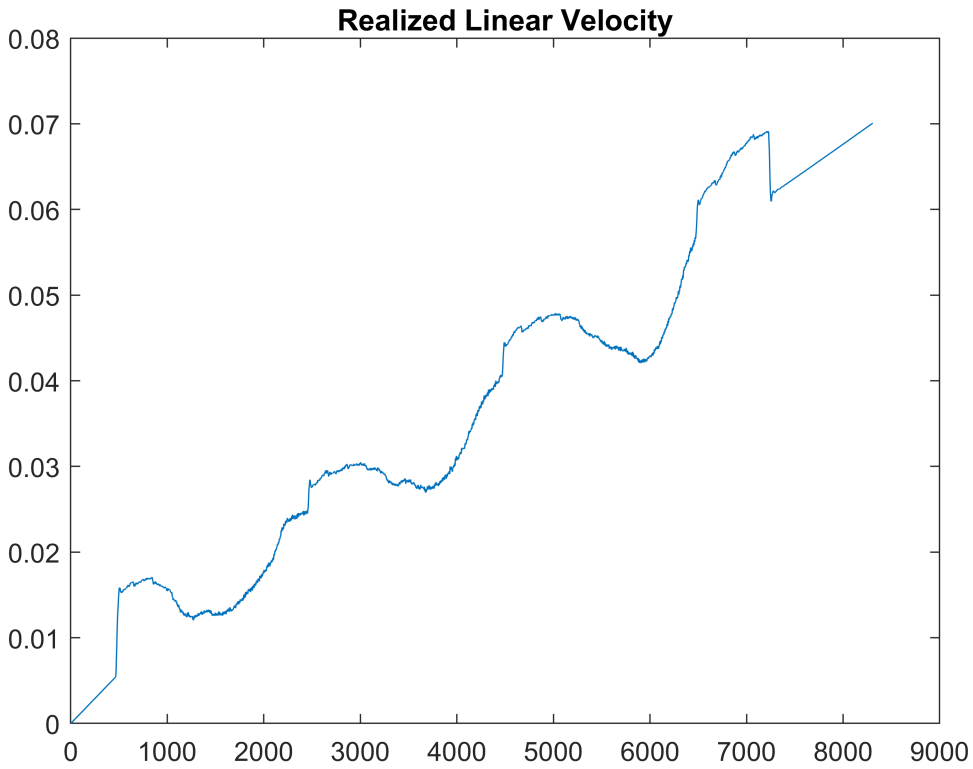

```
figure
plot(w_lr)
hold on
plot(w_lf)
hold on
plot(w_rf)
hold on
plot(w_rr)
title('Command Wheel Angular Vel four wheels')
```



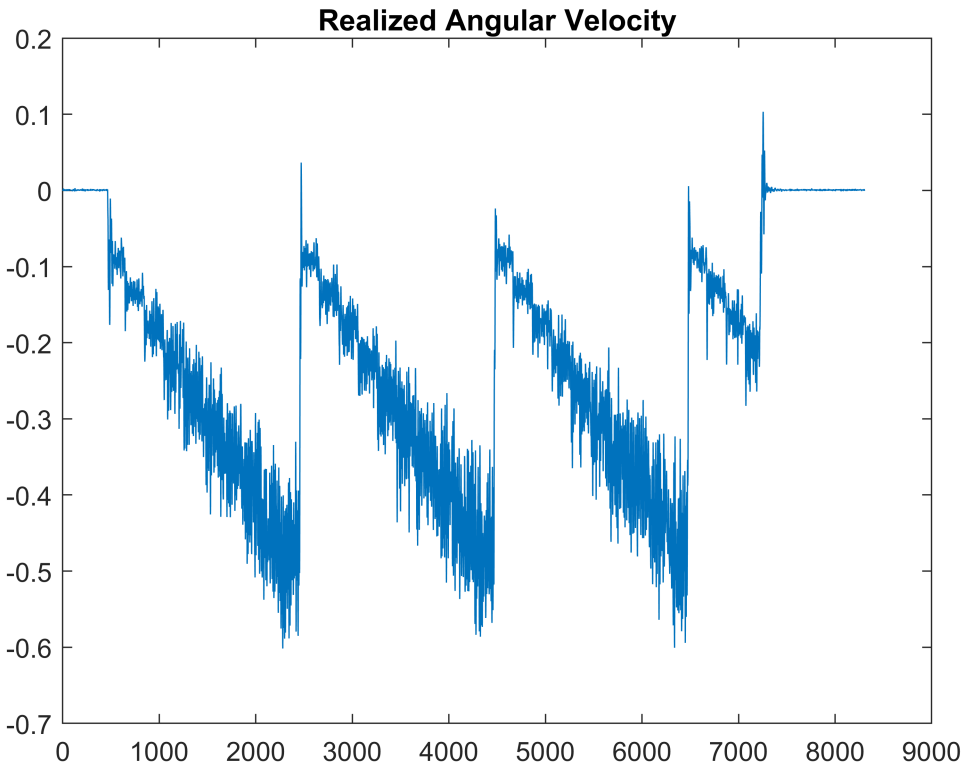

```
figure
plot(lin_acc)
title('Realized Linear Acceleration')
```



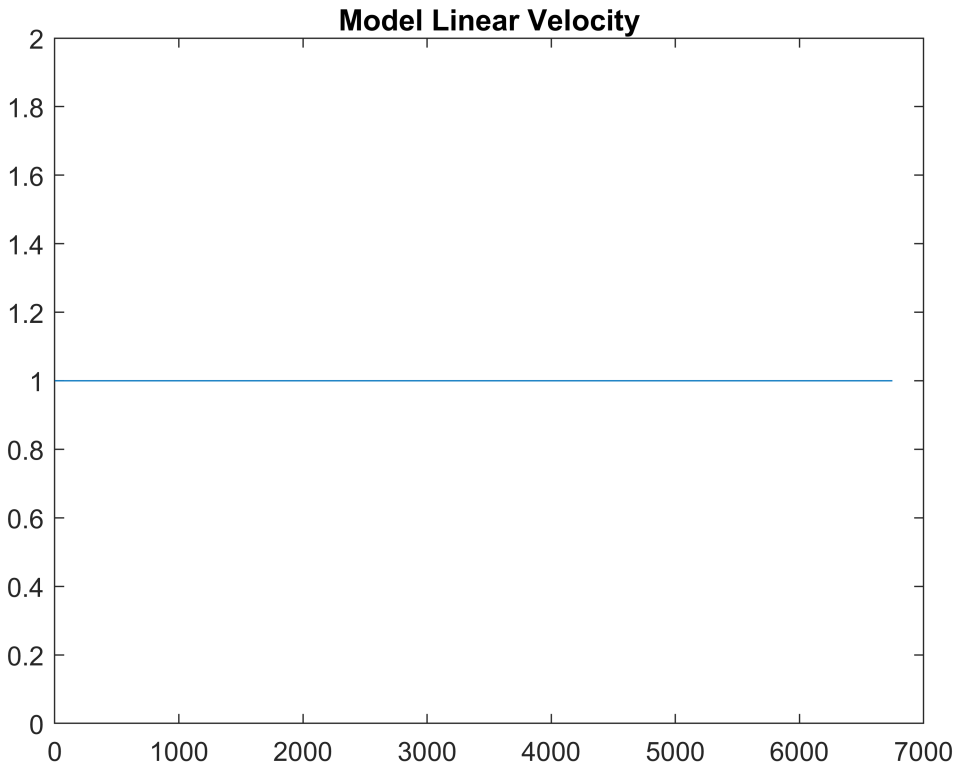

```
figure
plot(lin_vel(1:end-1))
title('Realized Linear Velocity')
```



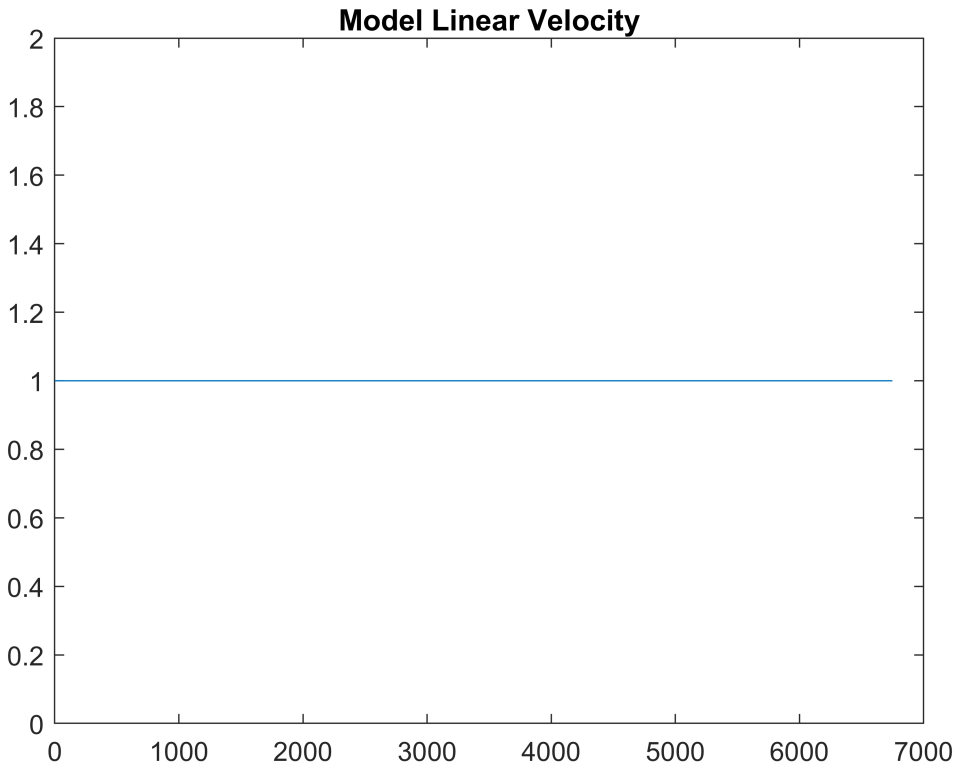

```
figure
plot(ang_vel)
title('Realized Angular Velocity')
```



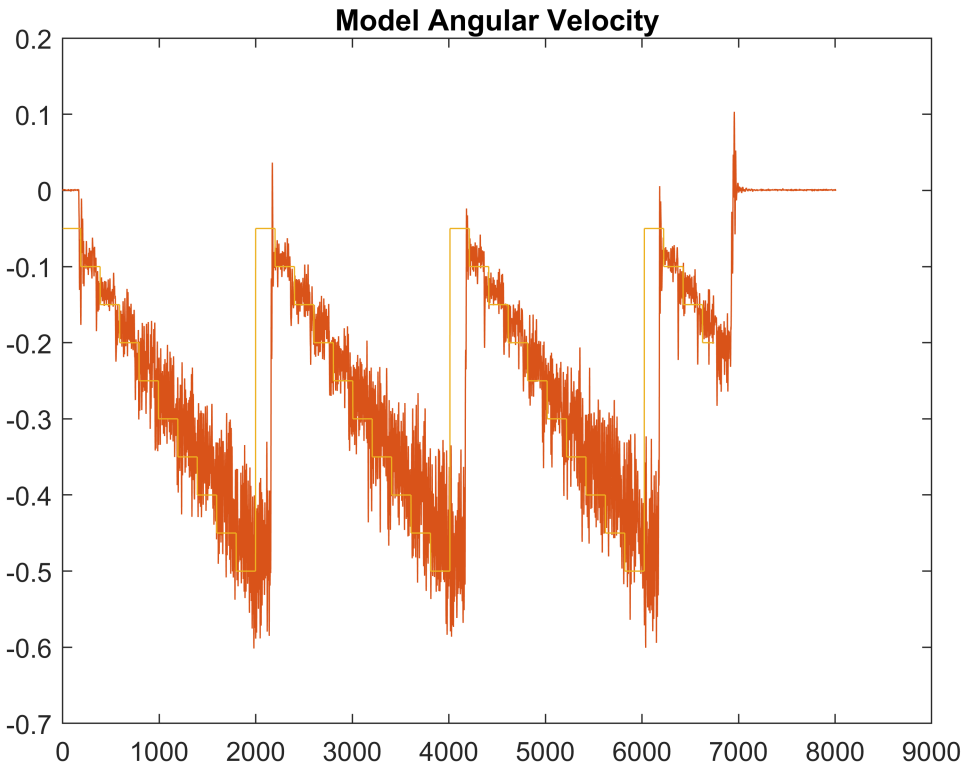

```
figure
plot(vel_hat(1,:))
title('Model Linear Velocity')
```




```
figure
plot(vel_hat_f(1,:))
title('Model Linear Velocity')
```




```
figure
plot(vel_hat(2,:))
title('Model Angular Velocity')
hold on
plot(ang_vel(300:end))
hold on
plot(vel_hat_f(2,:))
```


Model Angular Velocity
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
-0.7
0
1000
2000
3000
4000
5000
6000
7000
8000
9000