# EVER 2024 Autonomous Track

Assiut University / Assiut Motorsport

Identification.....

**Overview**

**Part 1:**

**Applying Gaussian noise:**

based on the CoppeliaSim Software, we started to collect data of x-position, y-position, and yaw data of our system -the car relative to the ground-. At first, we tried ROS Data Analyzer Matlab application, but the real time plotting of the coordinates was too messy. So, we tried another approach.

Since we have the data from Milestone-1 for the four tracks in CSV-files. So, after a deep understanding of Kalman filter and its implementation in Matlab we were able to define our system parameters like (P, Q, A, B, C). Then, we applied Gaussian noise and Kalman filter over these 3 coordinates through Matlab over these stored data. And plotted the results to make sure that the Noise function and its filter work.

Then to make a closed loop controller we first subscribed over the Odometery topic from Matlab, then Matlab applied the noise and filter to the data then the data is pushed using ROS to  a Python based Vehicle Path Tracking Controller to control the steering, gas pedal, and brake pedal of the car following almost the same original tracks.

In general, we noticed that with more real known data about the system, a Kalman-filter can be a trusted to be applied as its accurate final result.

Here you must conclude an overview to the system you have implemented, the different experiments that you conducted, and the overall perspective that you conducted from these experiments.

## Methodology Used

### Applying Noise:

we started by storing the data of x, y, and yaw coordinates of the original track made in Milestone-1.
We used (**data = readtable(filename);**) function in Matlab to read the original CSV file of the track.
Then we used the function
(xPointsM= awgn(columnData_N,10,"measured") ;)
to apply a Gaussian noise with an snr =10.
Finally, we plotted the data and saved the new CSV file with noisy readings.

```
plot(columnData_Y, xPointsM) %Noisy Data
legend ('original signal','signal with AWGN')
% Combine the header and the data
% Write the data with header to a new CSV file
%writecell(dataWithHeader, newFilename);
```

### Applying Filter:

We imported these data in a new Matlab file to apply Kalman filter to them. Using the following assumptions for our system:

```
A = [ 1 0 0;          B= [ 0 1 1;          H= [ 1;
     0 1 0;               0 1 1;               0;

     0 0 1];              1 0 0];              0];

Q = 0.629*eye(3);          R = 0.1*eye(3);

P = eye(3);

x = [0;

     0;

     0];
```

State transition matrix (A):

by calculating all 9 partial derivatives we got the A matrix [nxn].

$$A_{t-1} = \begin{bmatrix} \frac{\partial f_1}{\partial x_{t-1}} & \frac{\partial f_1}{\partial y_{t-1}} & \frac{\partial f_1}{\partial \gamma_{t-1}} \\ \frac{\partial f_2}{\partial x_{t-1}} & \frac{\partial f_2}{\partial y_{t-1}} & \frac{\partial f_2}{\partial \gamma_{t-1}} \\ \frac{\partial f_3}{\partial x_{t-1}} & \frac{\partial f_3}{\partial y_{t-1}} & \frac{\partial f_3}{\partial \gamma_{t-1}} \end{bmatrix}$$

Control Input Matrix (B):

The B matrix has the same number of rows as the number of states (x,y, yaw) and has the same number of columns as the number of control inputs (steering angle, gas pedal, Brake Pedal). [nxm]

Observation Matrix:

is a virtual sensor that converts from state of system that is measured (x,y, yaw) to steering, gas pedal, and brake pedal. In some other cases it called (C) and it is a [pxn] matrix.

## Closed loop control :

path 1: we used PID control for the the path by using the error form the difference between our current position which was by monitoring y mainly and the desired set point which was in this case 75 m in y , so we started tuning the parameters and applying the PID output on the gas pedal which was higher than one for the most part and then starts to slow down and getting lower than one as we get closer to our set point , while tuning the parameters , we couldn't allow any oscillation in the system as we can't go backwards if we pass the set point and that is why we neglected the integral parameters in the PID and focused manly on PD control while also increase the damping to nearly zeroing the overshoot for the same no backwards moving reason , thus making the whole model close to being critically damped and finish it's track in a relatively high settling time but with very relatively small steady state error of 0.6% .

Path 2 :in this path we used PD control to maintain an accurate output along with the circle function to keep the car on track of a circle with little oscillations as possible and with a steering angle achieving the desired set point and continuously redirect itself with a PD output if it deviates from the steering angle set point ,thus maintains the circle radius and then applying a reasonable brakes to stop the car without excessive slipping and finishing the path.

path 3 : we used nearly the same approach as in path one ,a PD control on the path for the same reasons as in path one , then after reaching the first 75 meters after that it start steering with a yaw feedback and a PD controlled gas pedal value to reach about 1.75 m at which it switch back to straight forward making a total width of 3.5m which is the lane width and then again repeat the second 75 meters with the same criteria until it reaches it's goal with a very small steady state error as before.

Path 4: in this path ,we decided to use a direct closed loop control that takes the feedback and keep moving with a constant steering angle to achieve and maintain the desired radius for the first circle and when it close it and finish the circle or at least nearly doing it , it then shifts it's steering angle with same value but in the opposite direction achieving the same result after reversing the logic for the other side and then finally stops with a logical braking value as the one have is ridiculously high causing the wheel to slip even in low speeds as it directly lock the wheels when applied which is why we used in all the passes a special

value of nearly 1.6*10e-47 at changing it's value within this range to apply a reasonable braking so that it achieve the desired infinity path.

**RMS Value:**

**path1**

**RMS_X = 0.000960063**

**RMS_Y = 1.745773391**

**RMS_YAW = 0.018305783**

**path2**

**RMS_X = 0.519641575**

**RMS_Y = 0.519182174**

**RMS_YAW = 3.319783993**

**Path3**

**RMS_X = 0.028591671**

**RMS_Y = 3.251362656**

**RMS_YAW = 0.011996643**

**Path4**

**RMS_X = 0.333448361**

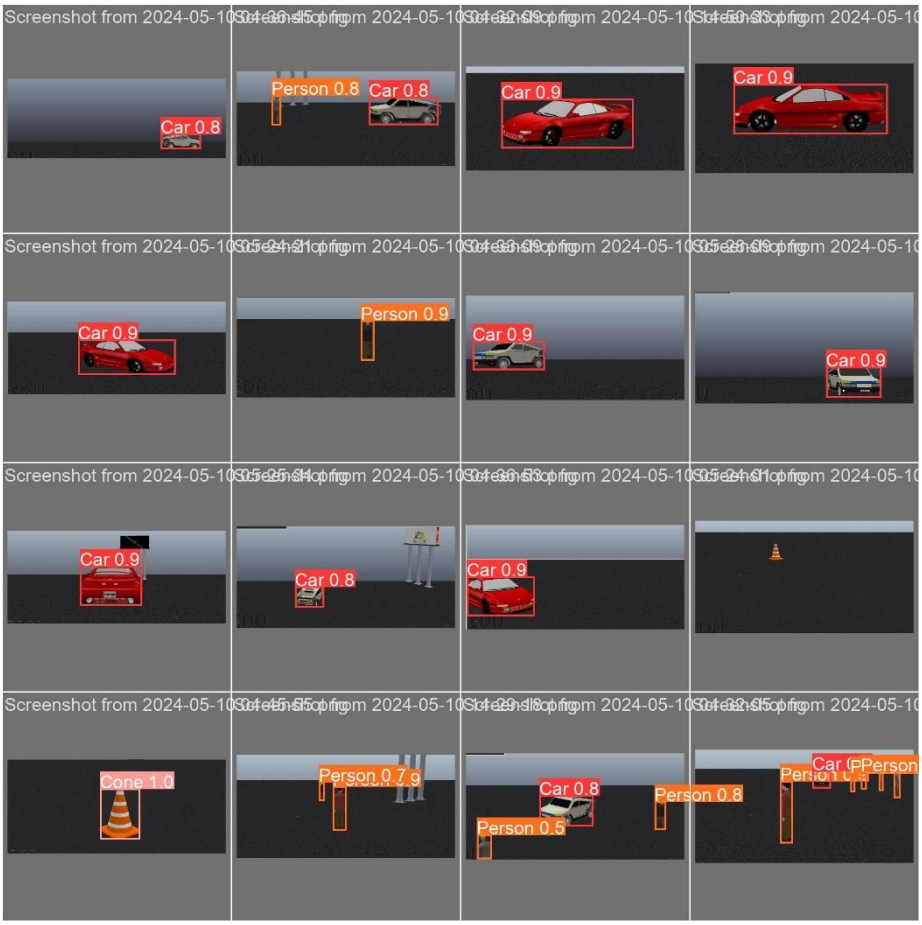
**RMS_Y = 0.380913341**

**RMS_YAW = 18.8839401**

**Part 2:**

First, we decide to train our model to learn more and more about training models It was a bit of a journey to find the best model that suits the required performance, we used TensorFlow as the library for detecting objects required (humans, cones, and Cars). Our main reason for using TensorFlow is the wide range of models that can be chosen from (SSD, Faster CNN, Center Net, etc.) causing us to train multiple

models with almost the same parameters, hence, choose the most efficient one, We started with Center_net as it had the highest MAP (respectively), but the model was hard to train on our machines and didn't have very good detection time, So we decided on using different model, which was SSD - we tried Faster_RCNN as well but it has the problems of Center_Net-and after 20000 steps, the performance was good but classification loss was not as expected, so after training multiple times, we decided to change the test and training data, the classification loss was better, but the regularizer loss was increased for a very high number, we did some research and found out that removing it from the whole function was -in our case only- better for the whole performance, and after more steps, the SSD model was ready after that we discover that that  accuracy isn't good then we decide to train on YOLO V8 because after search we discover it will give us large accuracy with small data so we train many time with different data and compare it with SSD we use google colab instead of our machine it stop from itself as no improvement observed in last 100 epochs and the last train was good and the validation was  almost good as you can see in the next image




But after Tring it give us bad accuracy, we decide we use a pretrained

model in the YOLO, but it contains two category cars and peoples, so we decide to use image processing to detect cones the algorithm that we do using opencv

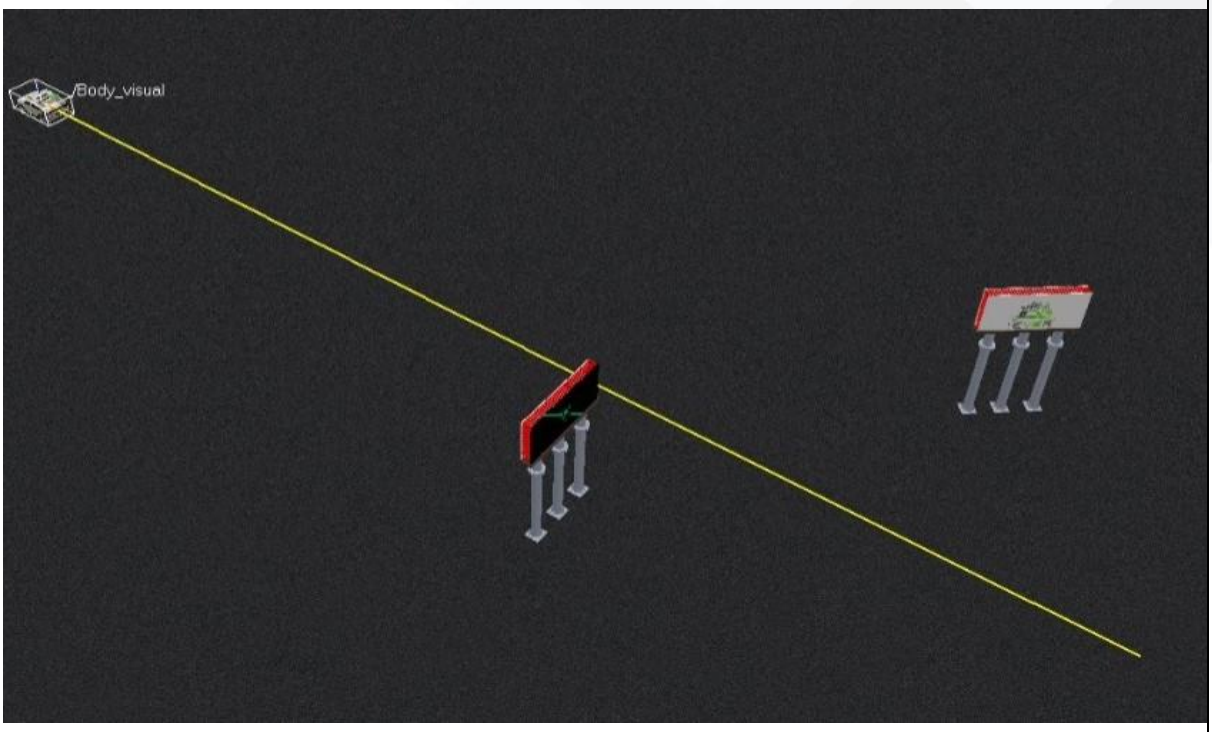
1- Take in the original image from camera.
2- Convert () image to HSV (HUE, Saturation, Value).
3- Isolate Oranges in Range() creating a low Threshold image and high Threshold image Create new image by using Logical OR the threshold images together.
4- Creating new image by smoothing the threshold image using, Erode(), Dllate() and Smooth Gaussian(), Run canny Edge Detection algorithm on smoothed image to create a canny image. Cany()
5- Create a Contour image by using findContours() and approxpoly() on the Canny image. Create an image of convex hull (cone Shape) for each contour in the listofContours by using getConvexHull.
6- Creare an image with convex hulls in the range of 3 to 10 by checking if convexHull.Total>=3 and convexHull.Total <= 10. Then check if(convexHullPointintUp(convexHull). (if the traffic cone is upright)
7- Draw line around remaining convexHulls (cones) and overlap with original image.

This was the algorithm that we use.
After that we discover a 3 pretrained model on the different site and give us high accuracy than YOLO each model for one label so we uses three PyTorch file in our code.

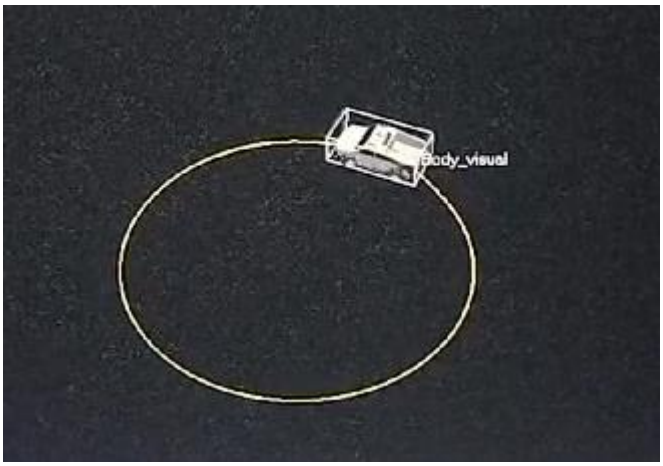
Here you should clearly mention the techniques you used to achieve the results you got and plan the trajectories of your motion.
You must write in a full detailed manners ex: (mention any equations you used, tools, methods, libraries, packages, etc.)

**Built Tracks**

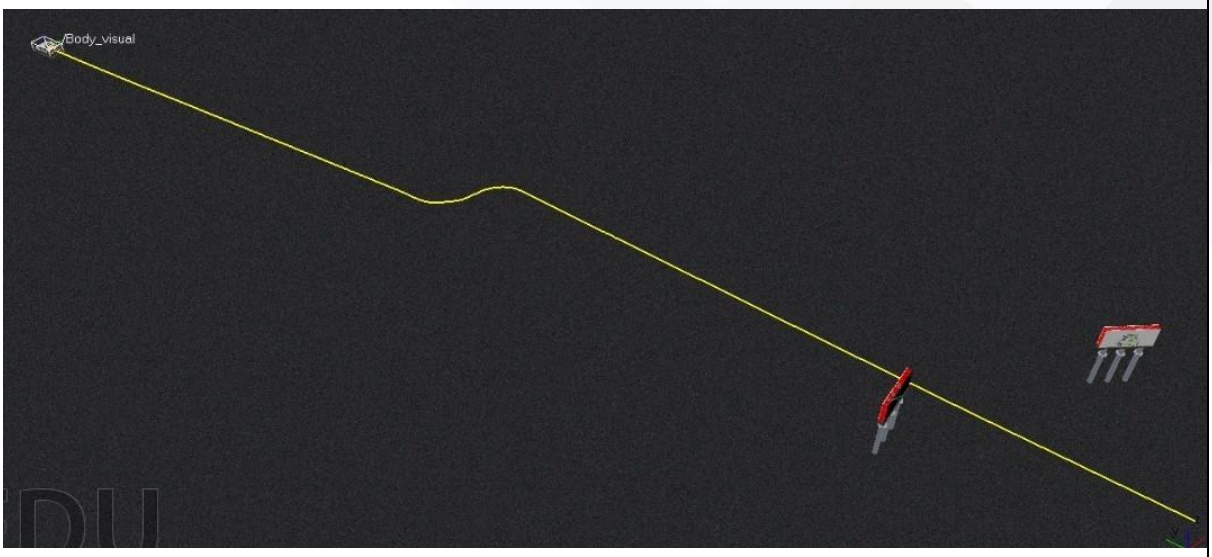
**Path1:**



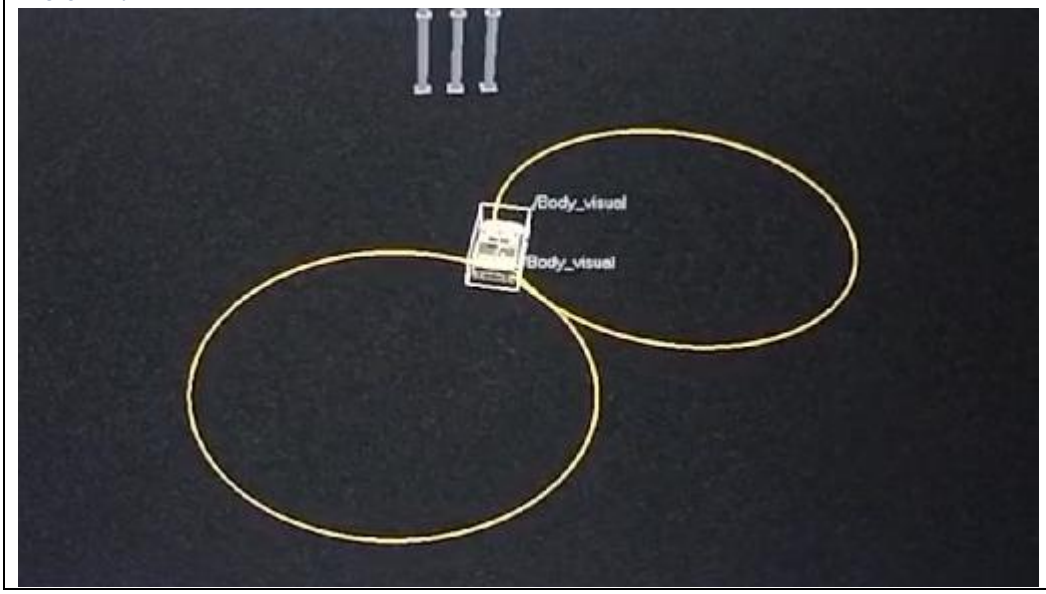

**Path2:**




**Path3:**



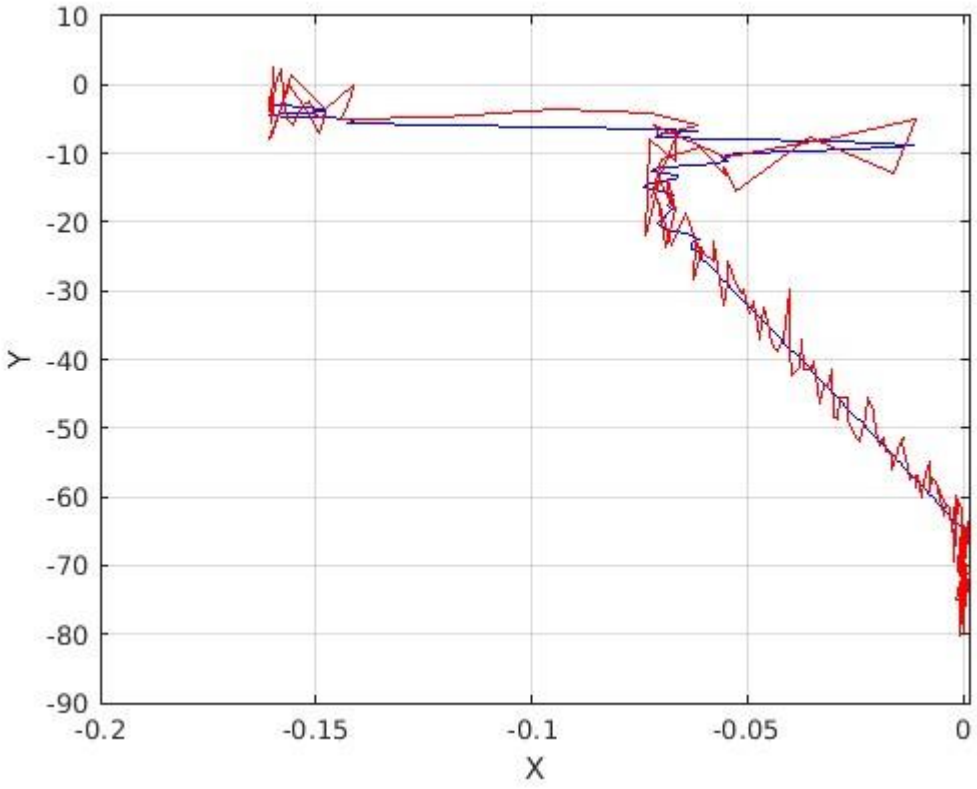

**Path4:**



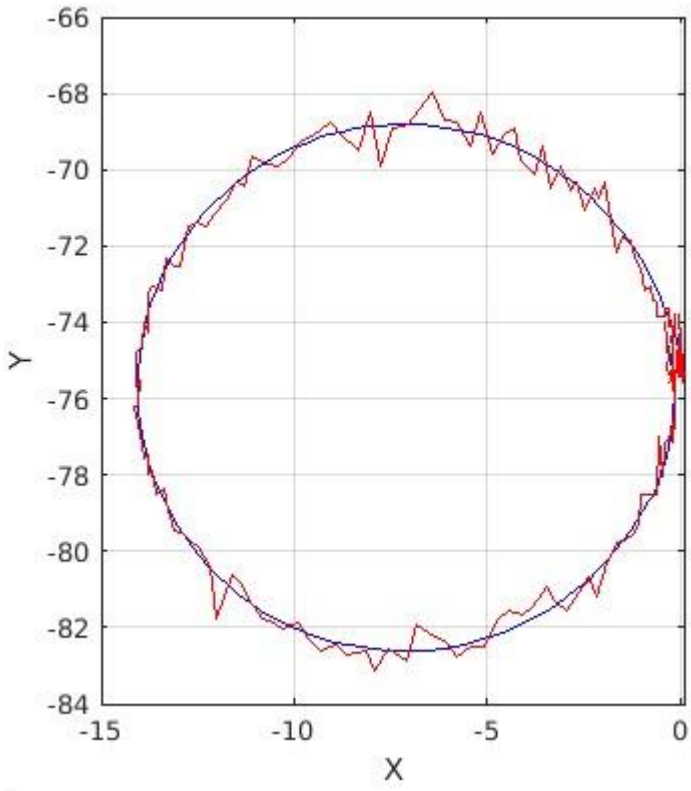

**Results**

**Open loop**

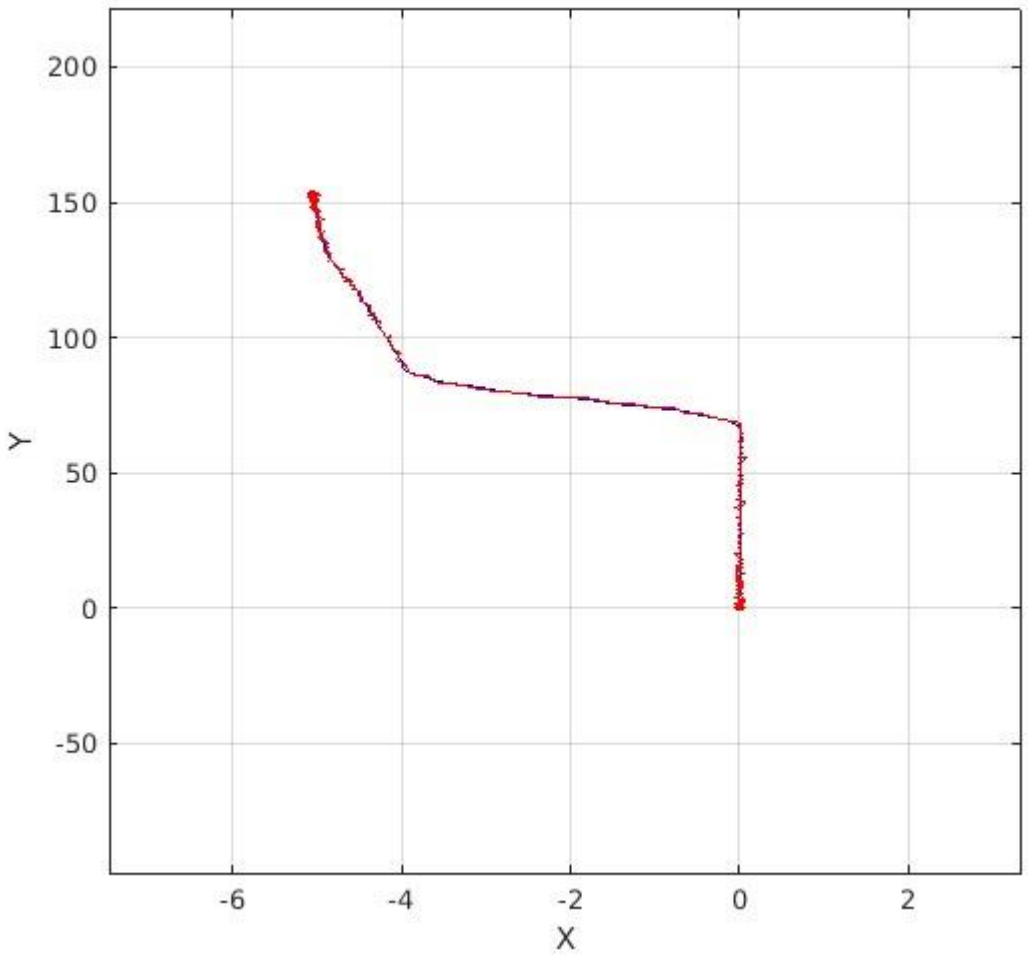
**Path1**


10
0
-10
-20
-30
-40
-50
-60
-70
-80
-90
Y
-0.2
-0.15
-0.1
-0.05
0
X

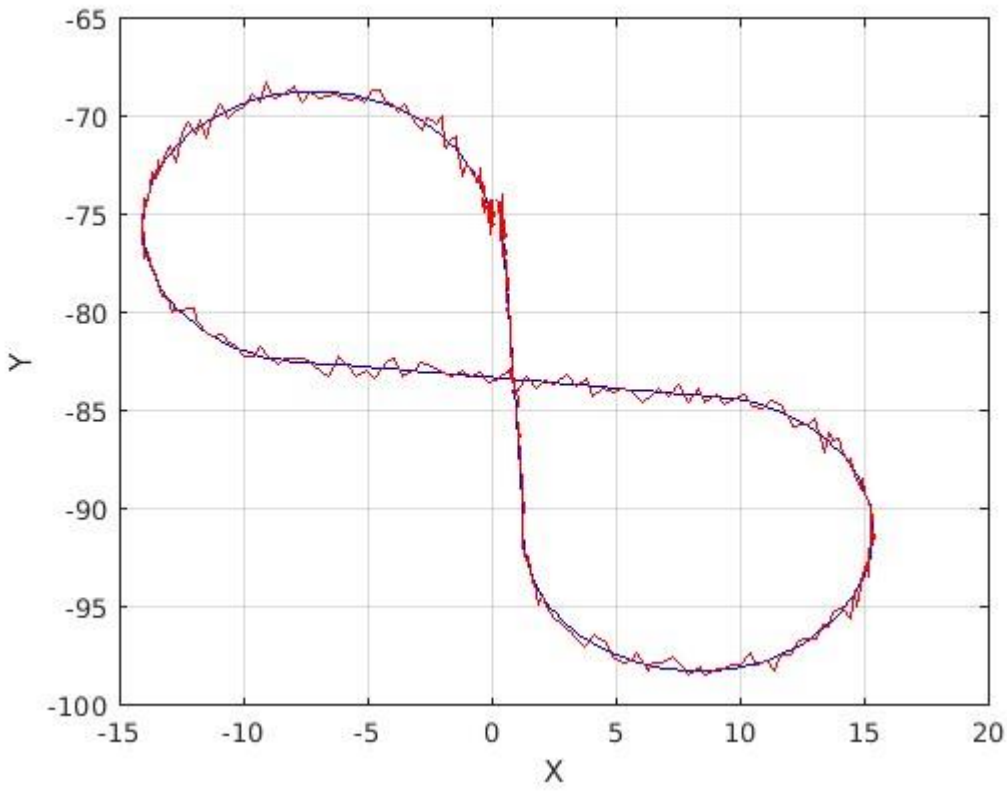
**Path2**


-66
-68
-70
-72
-74
-76
-78
-80
-82
-84
Y
-15
-10
-5
0
X

**Path3**



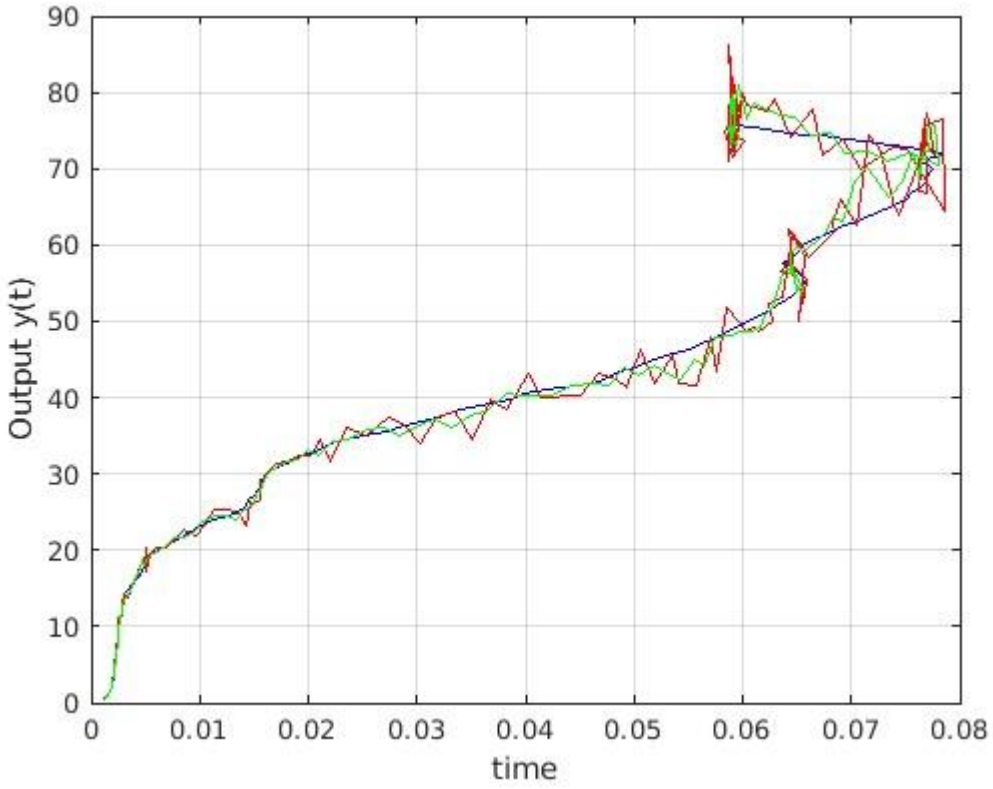

**Path4**



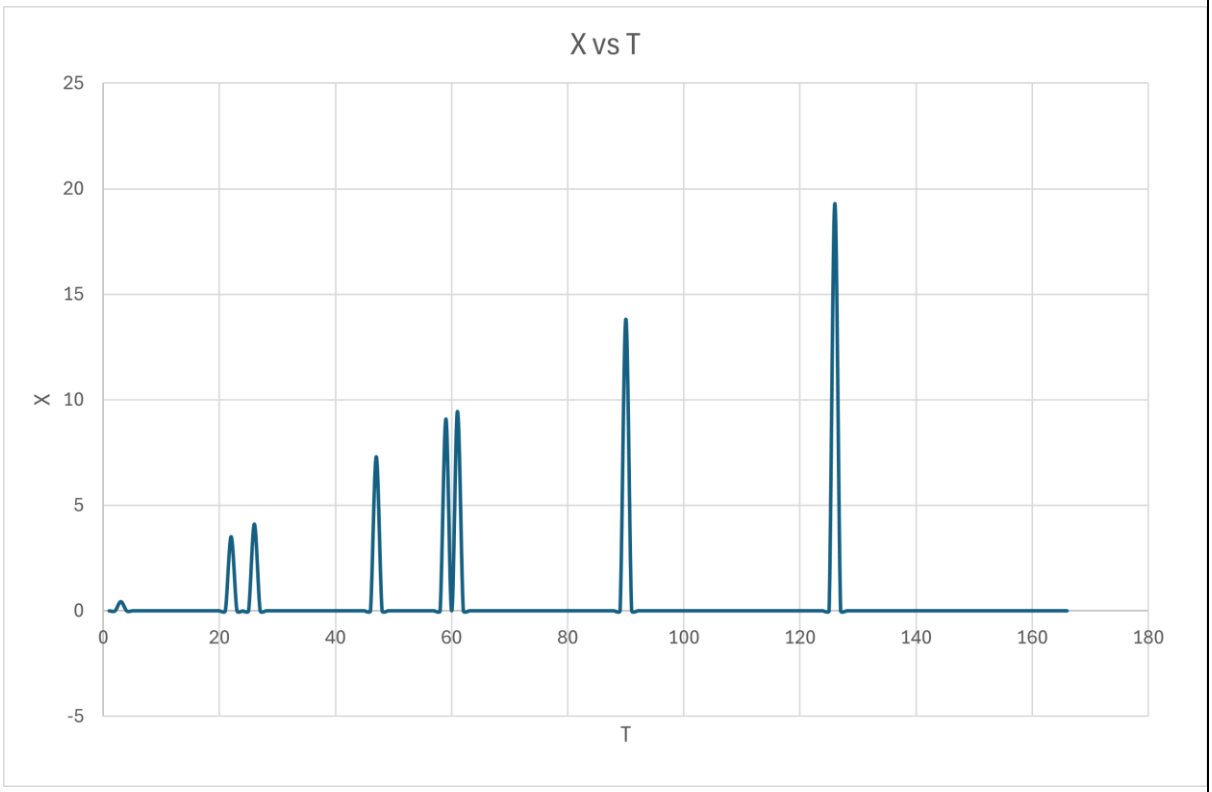

## Closed loop

### Path1

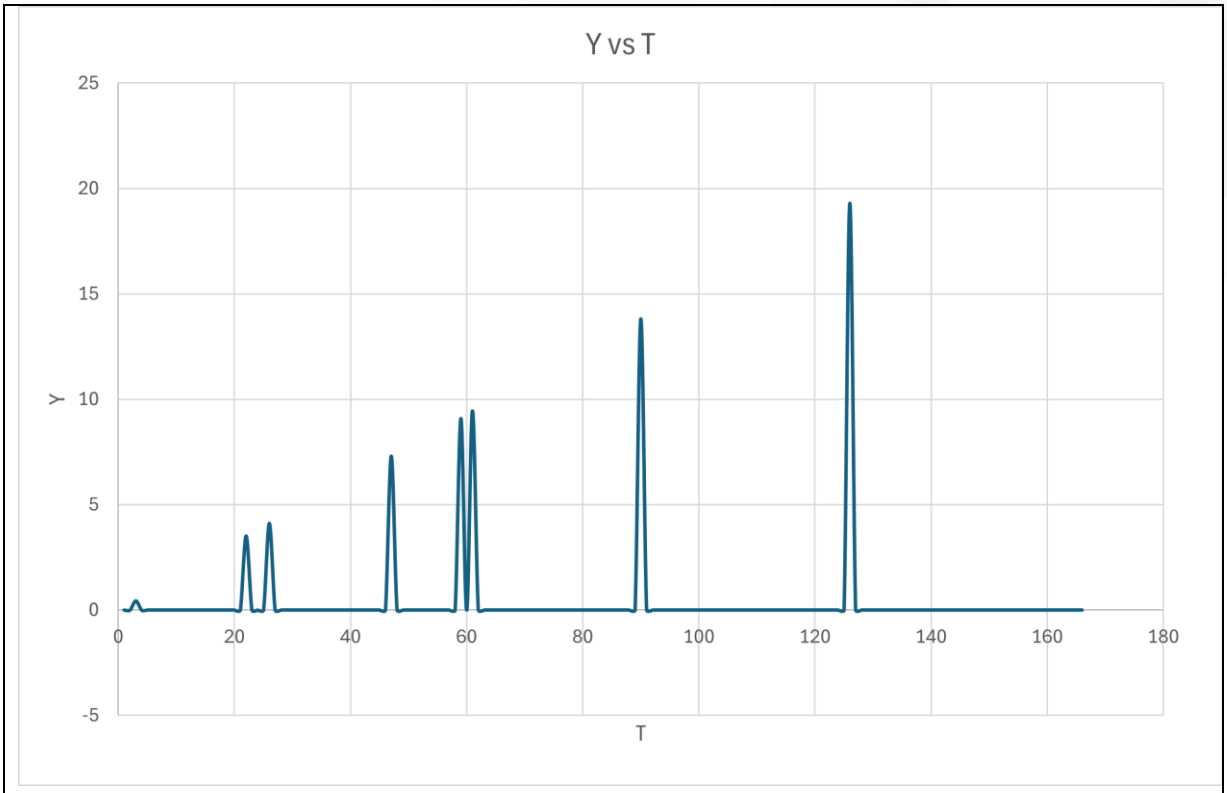
1- X VS Y


Output y(t)
time
0
10
20
30
40
50
60
70
80
90
0
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08

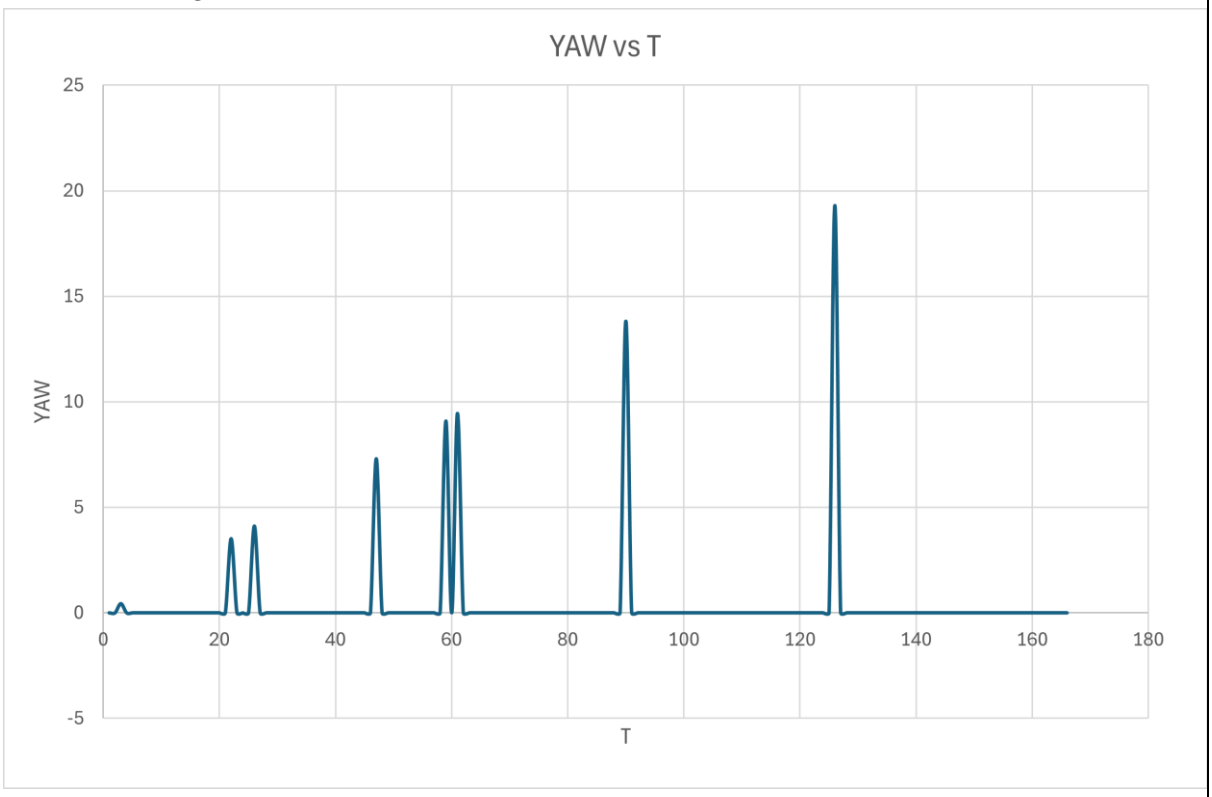
**2- X VS T**


X vs T
X
T
25
20
15
10
5
0
-5
0
20
40
60
80
100
120
140
160
180

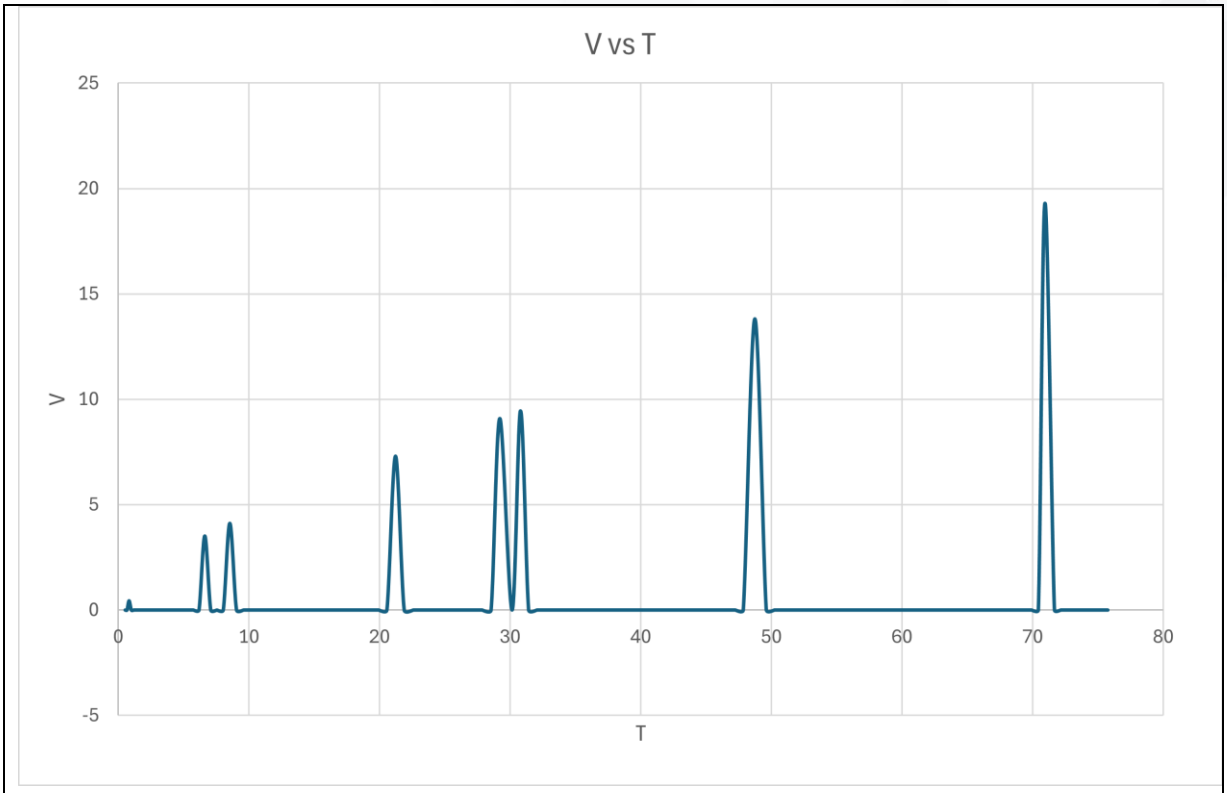
**3- Y VS T**



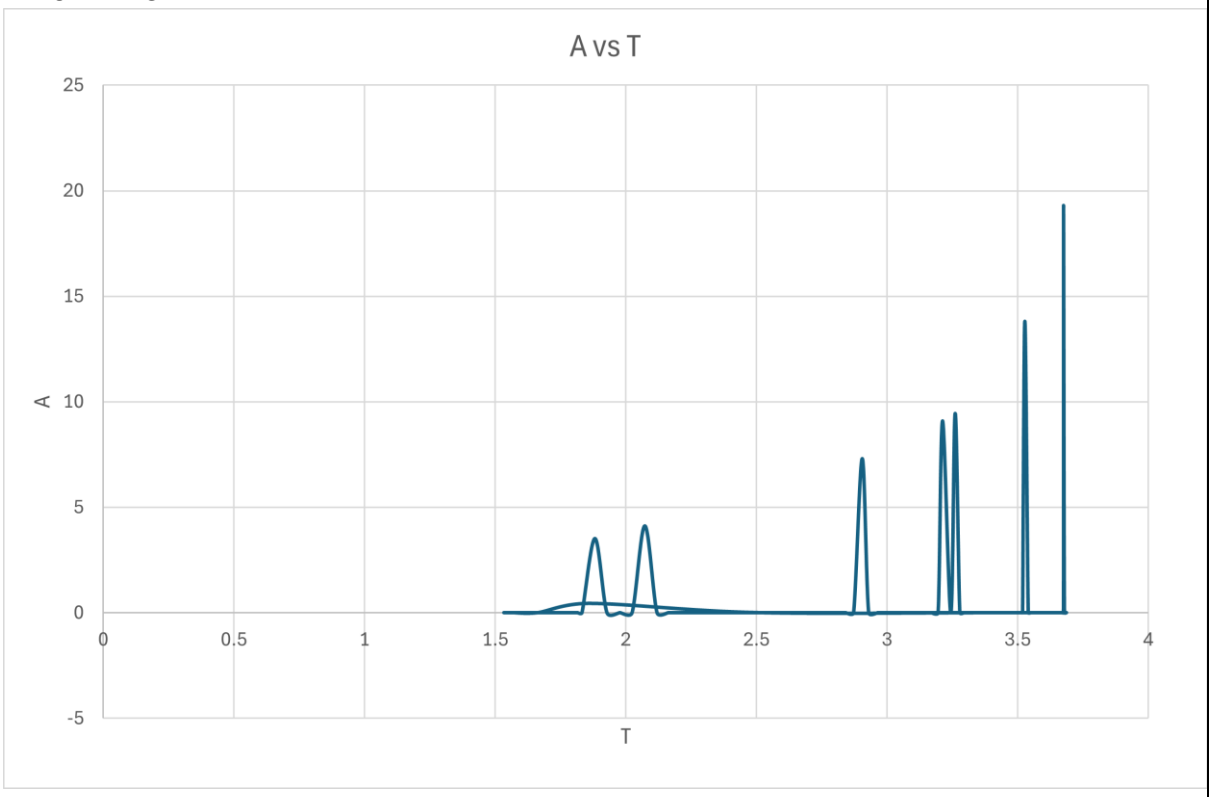

**4- YAW VS T**



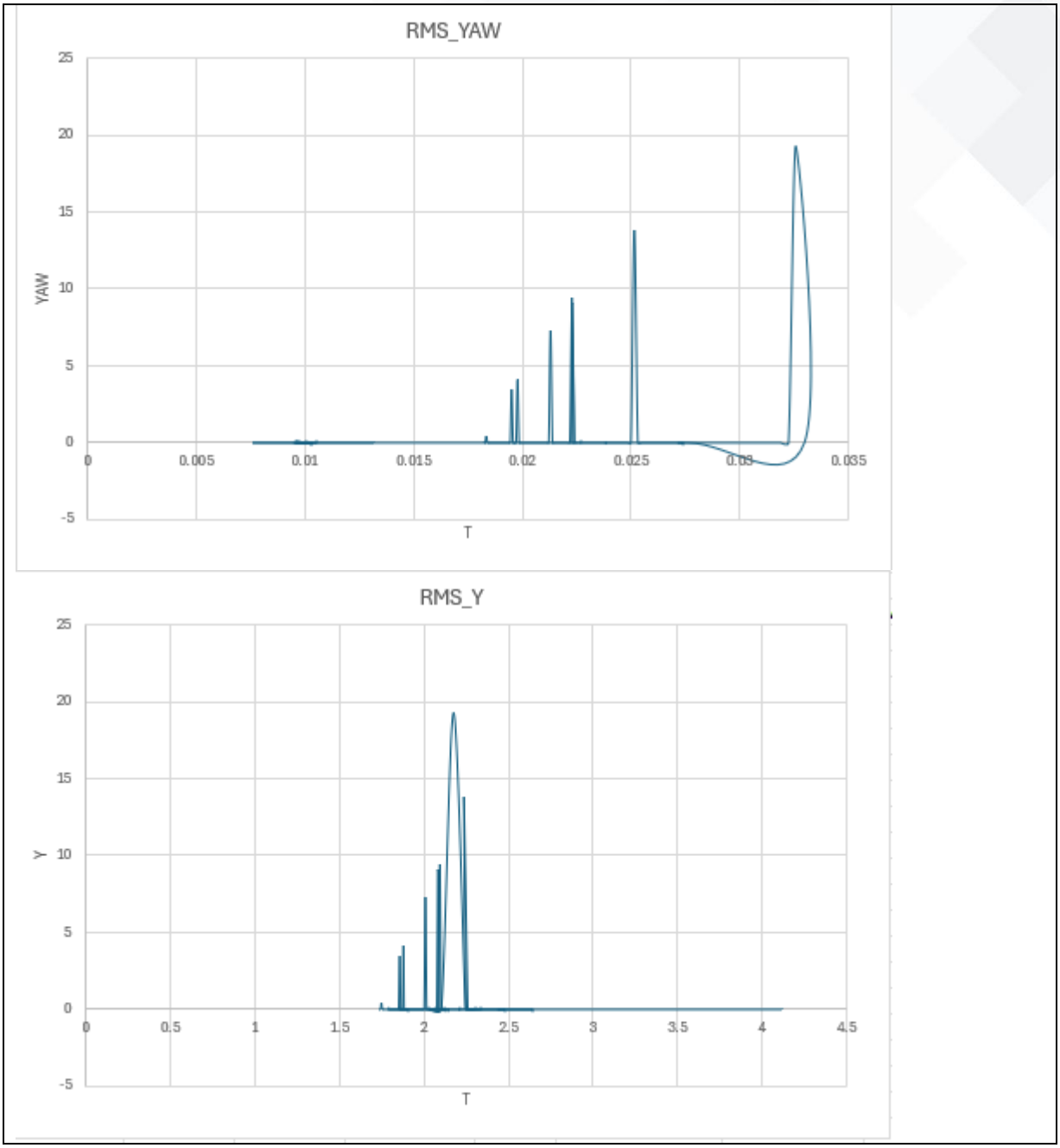

**5- V VS T**

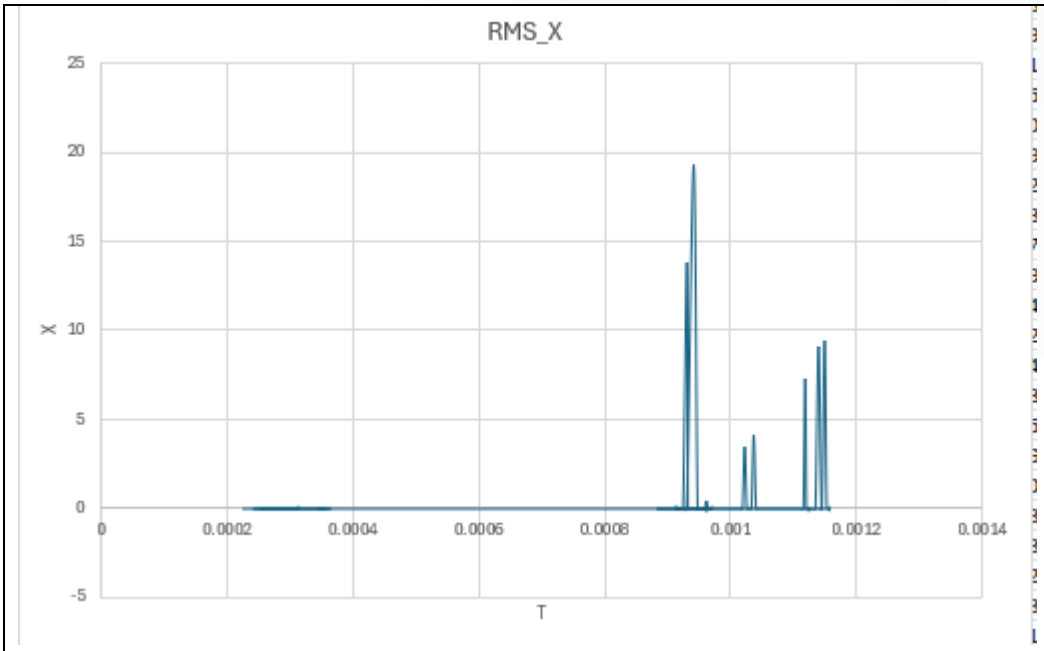



**6- A VS T**



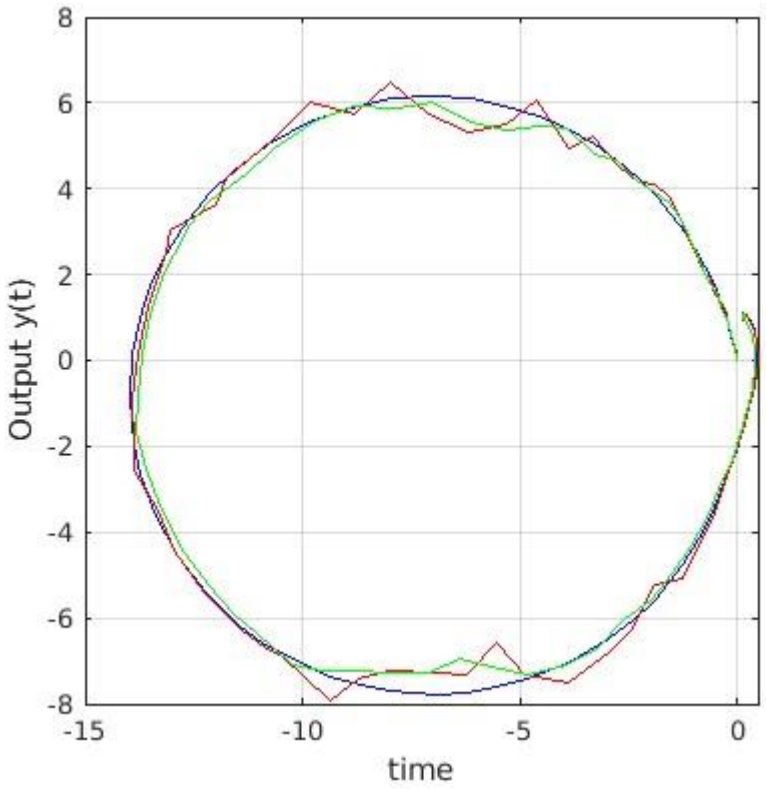

**7- RMS VS T**


RMS_YAW
YAW
T
RMS_Y
Y
T

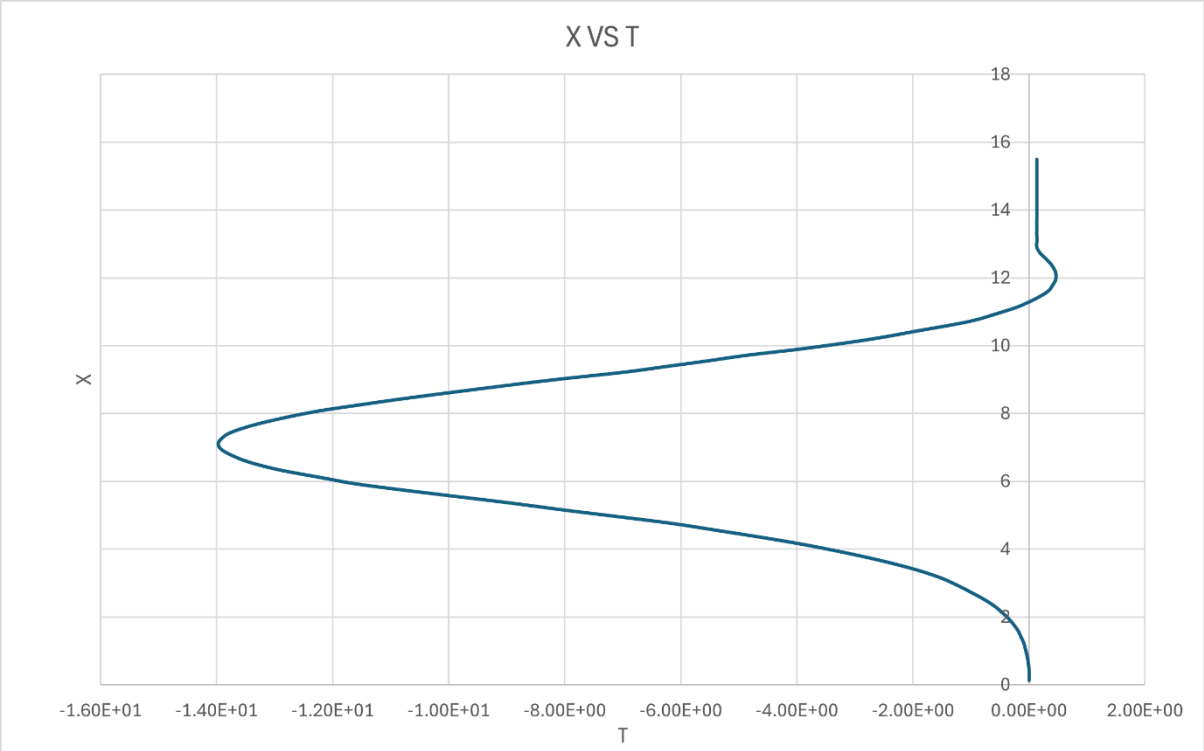

RMS_X
25
20
15
10
5
0
-5
X
0
0.0002
0.0004
0.0006
0.0008
0.001
0.0012
0.0014
T

**Path2**

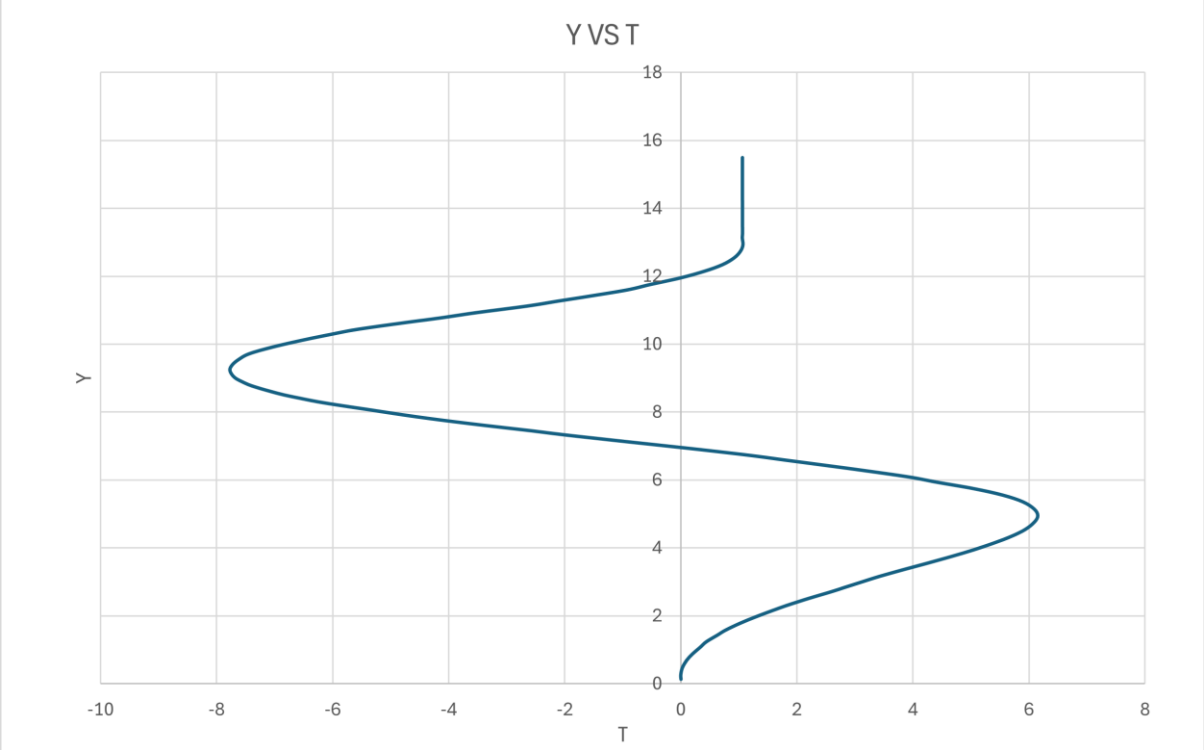
**1- X VS Y**


8
6
4
2
0
-2
-4
-6
-8
Output y(t)
-15
-10
-5
0
time

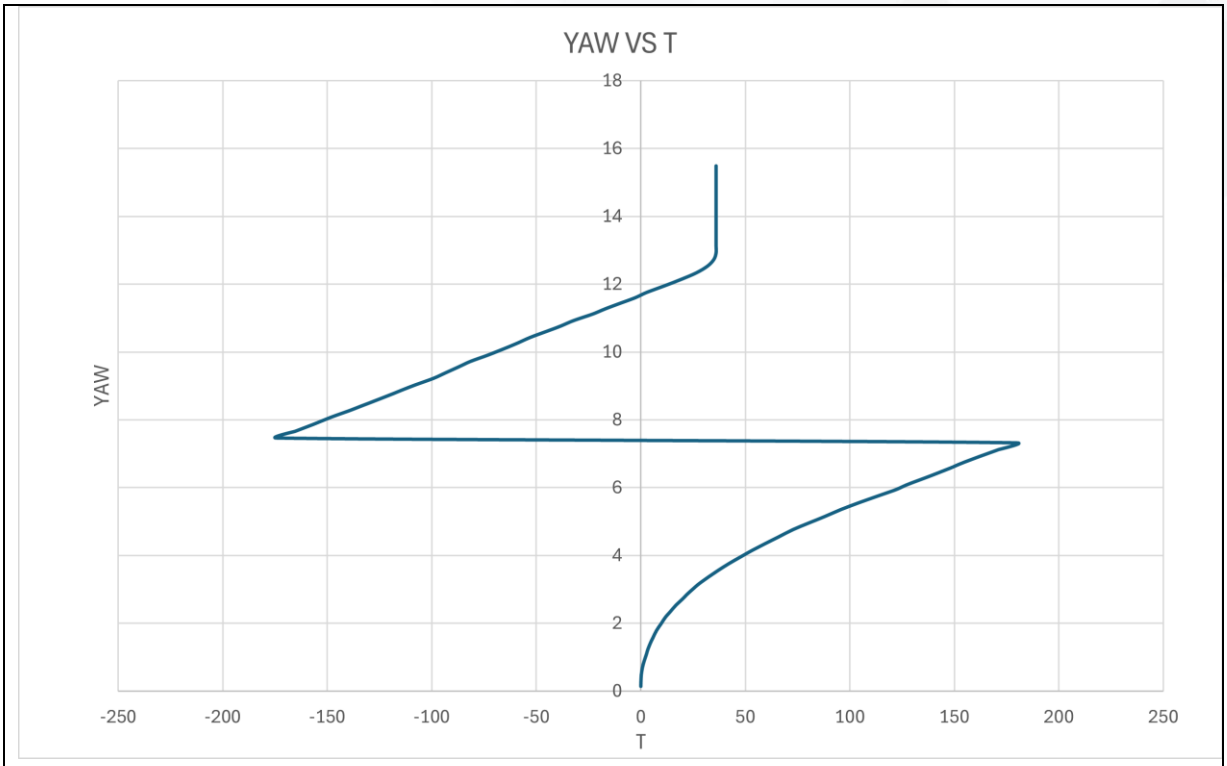
**2- X VS T**



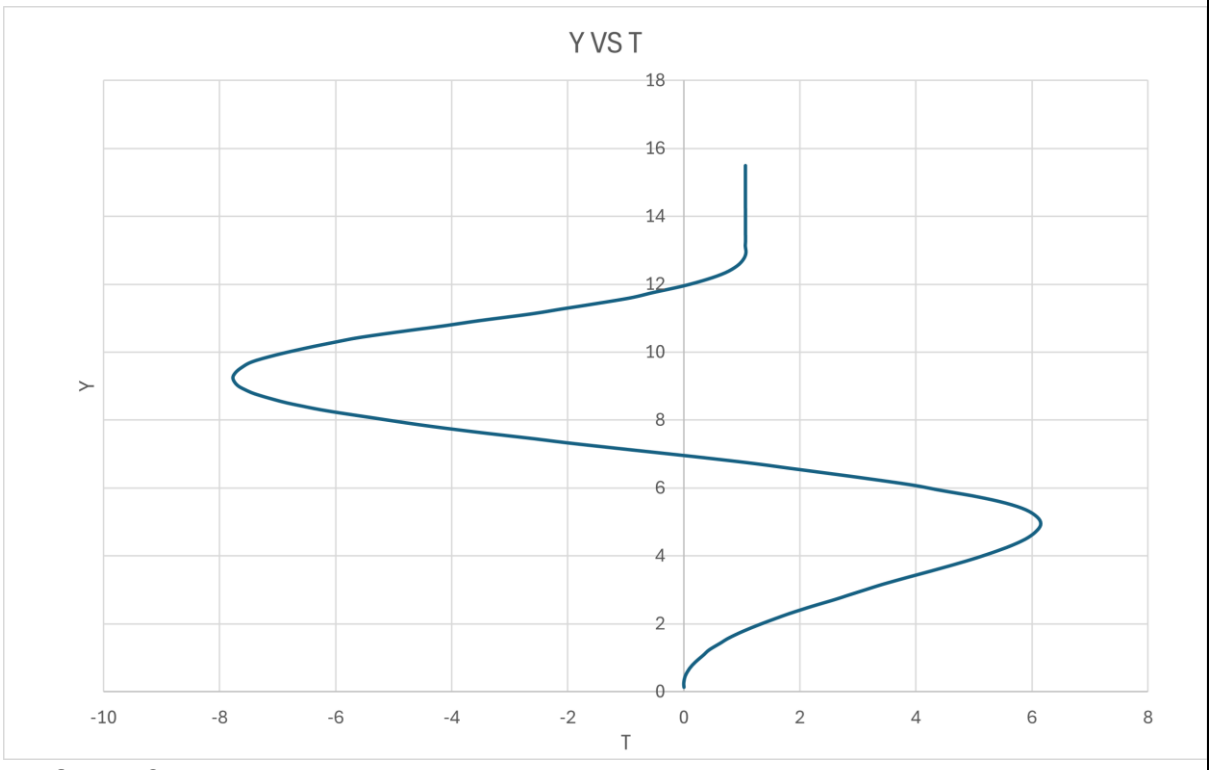

**3- Y VS T**



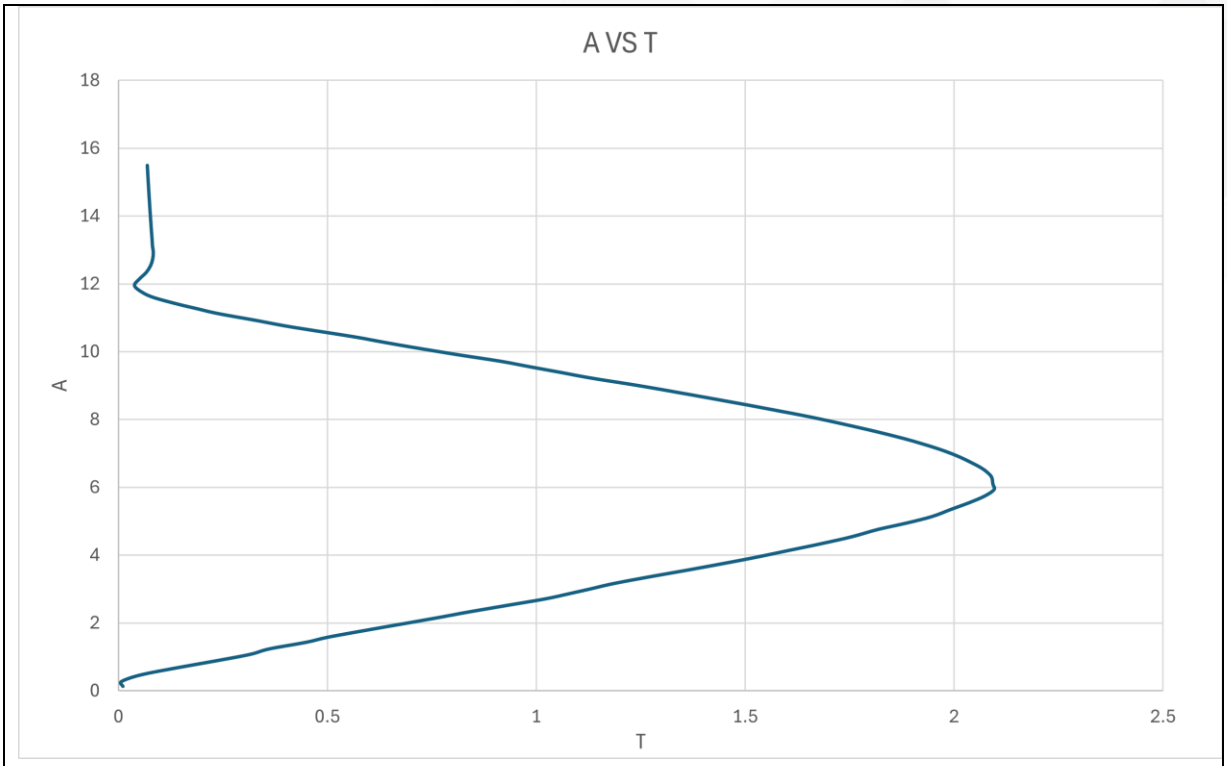

**4- YAW VS T**


YAW VS T
YAW
T
18
16
14
12
10
8
6
4
2
0
-250
-200
-150
-100
-50
0
50
100
150
200
250

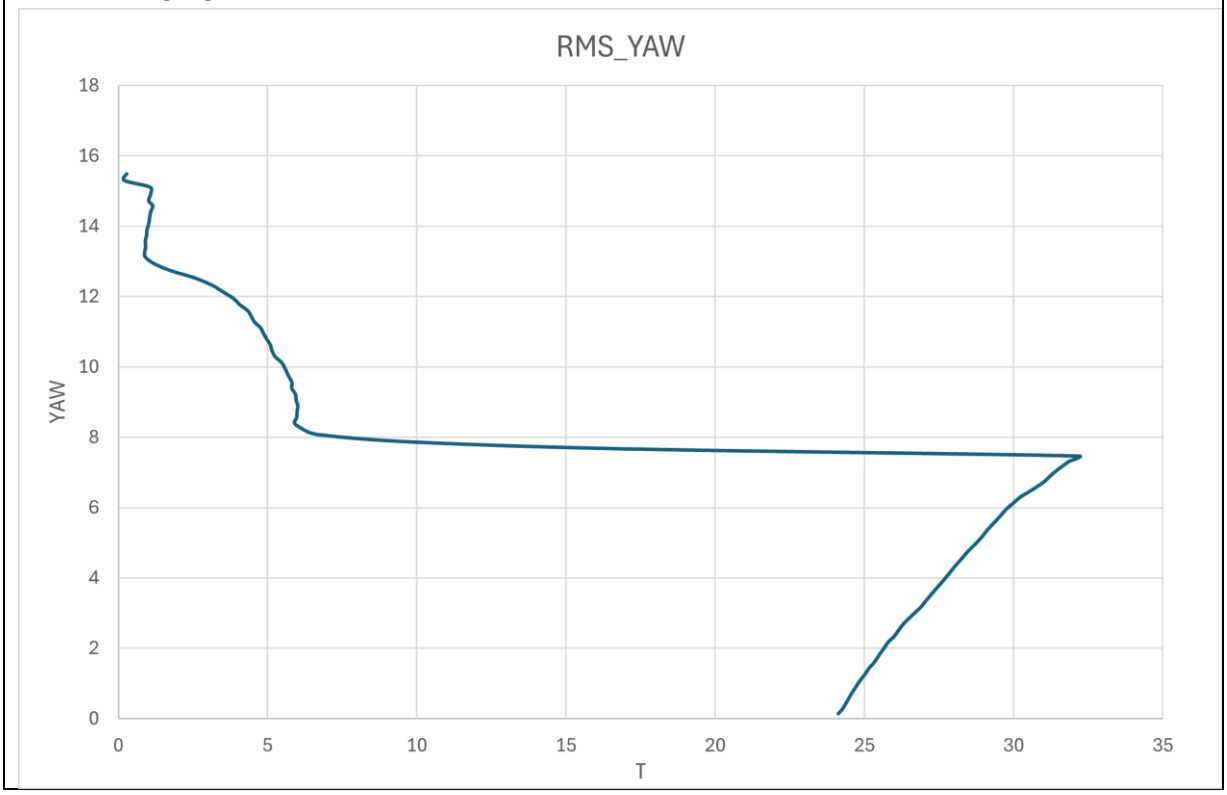
**5- V VS T**

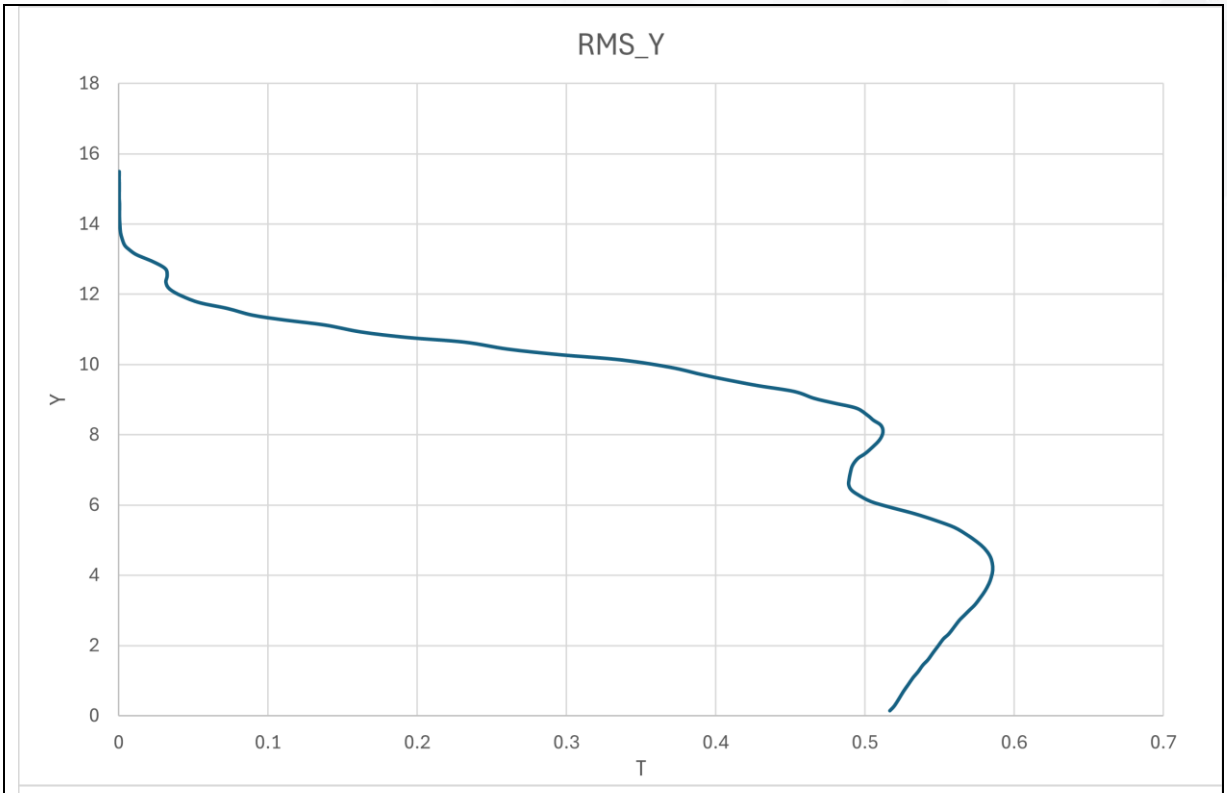

Y VS T
Y
T
18
16
14
12
10
8
6
4
2
0
-10
-8
-6
-4
-2
0
2
4
6
8

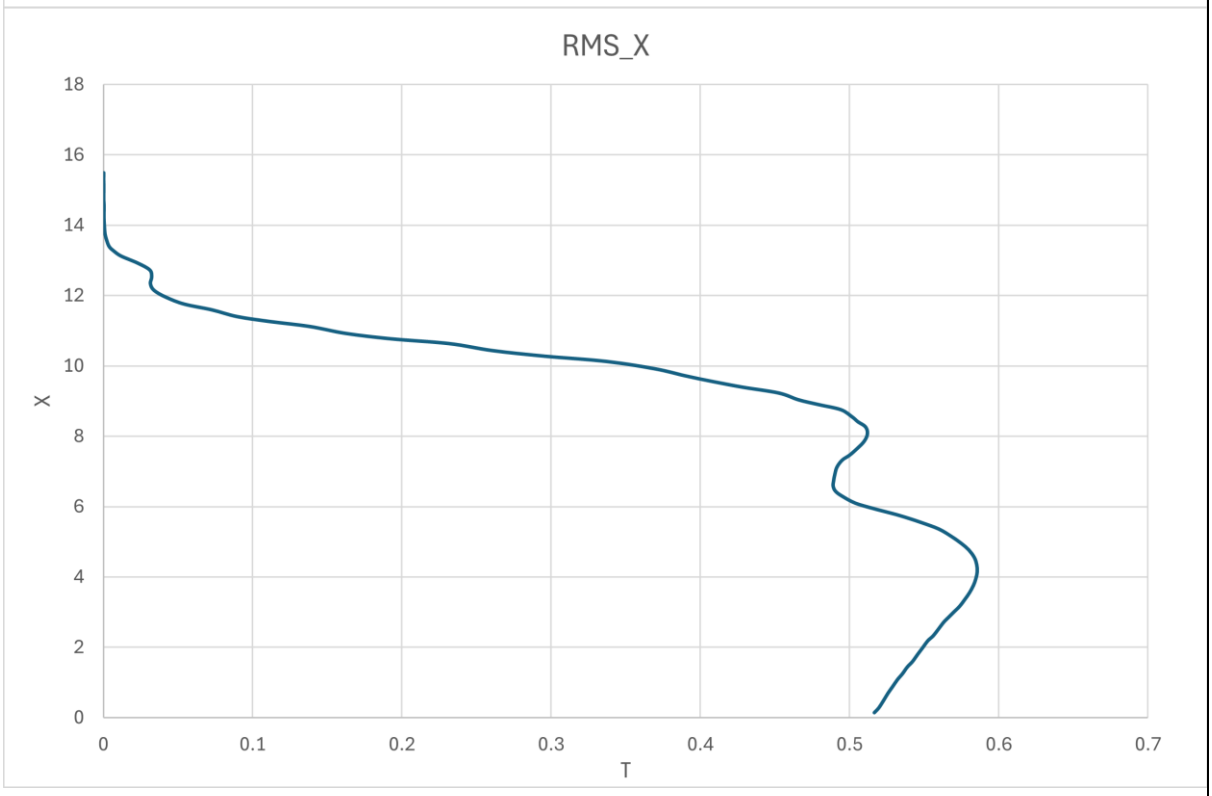
**6- A VS T**


A VS T
A
T
0
2
4
6
8
10
12
14
16
18
0
0.5
1
1.5
2
2.5

**7- RMS VS T**

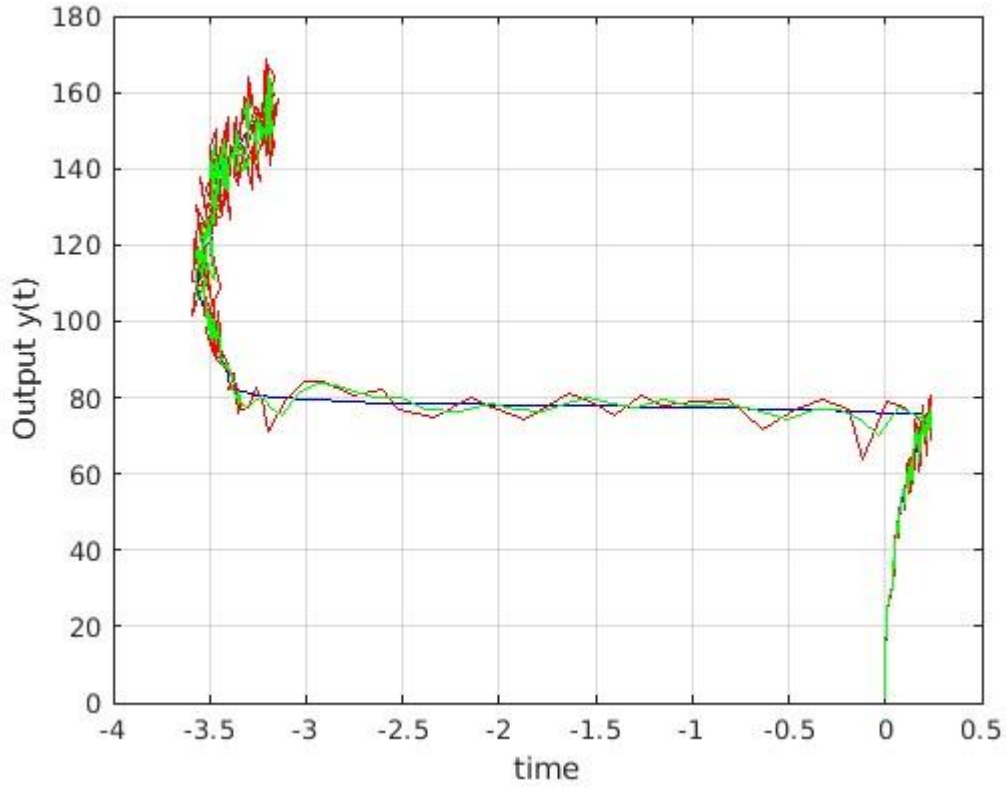

RMS_YAW
YAW
T
0
2
4
6
8
10
12
14
16
18
0
5
10
15
20
25
30
35


RMS_Y
18
16
14
12
10
8
6
4
2
0
Y
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
T

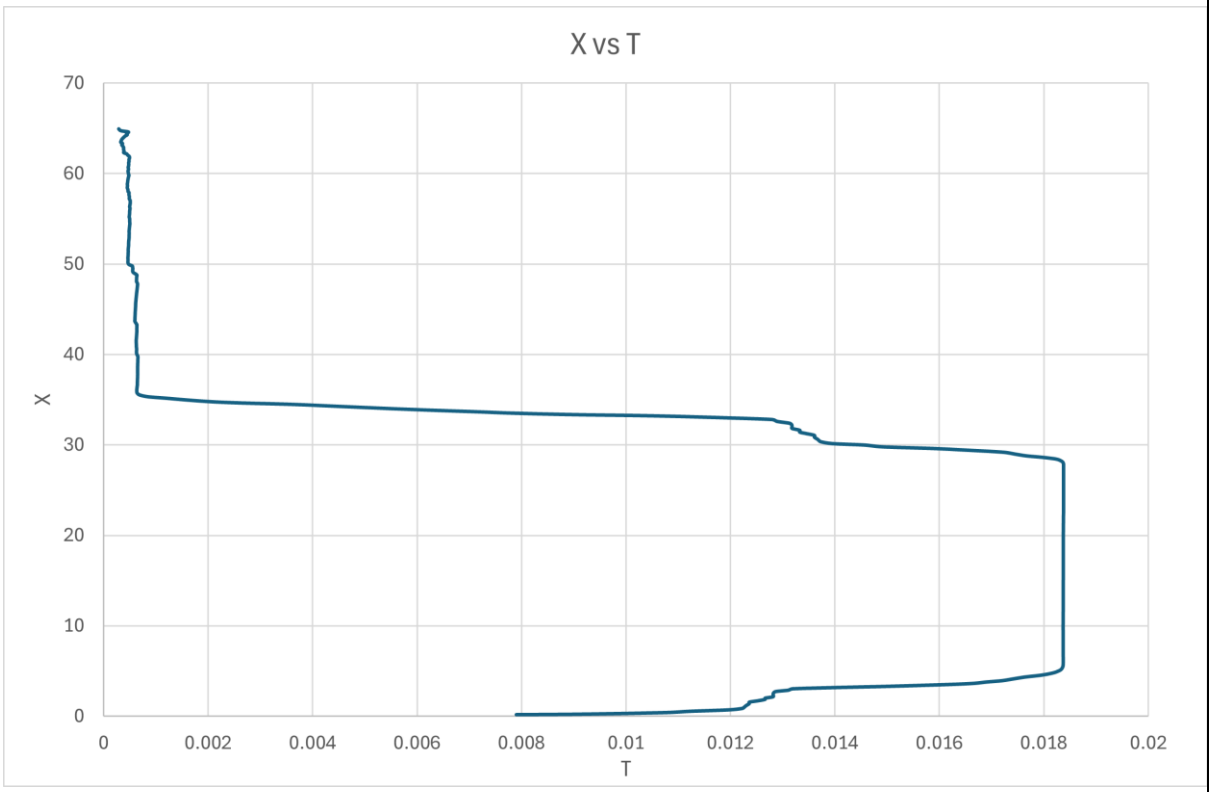

RMS_X
18
16
14
12
10
8
6
4
2
0
X
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
T

**Path3**

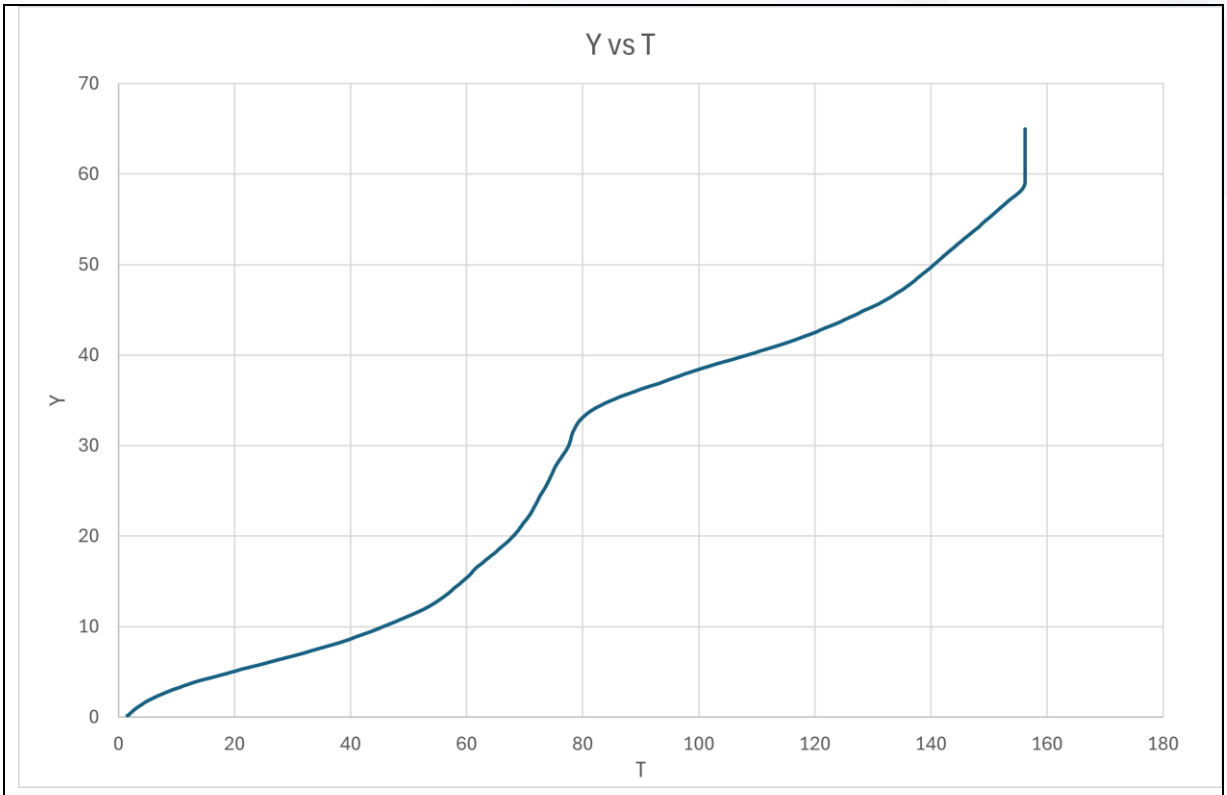
**1- X VS Y**


Output y(t)
time
180
160
140
120
100
80
60
40
20
0
-4
-3.5
-3
-2.5
-2
-1.5
-1
-0.5
0
0.5

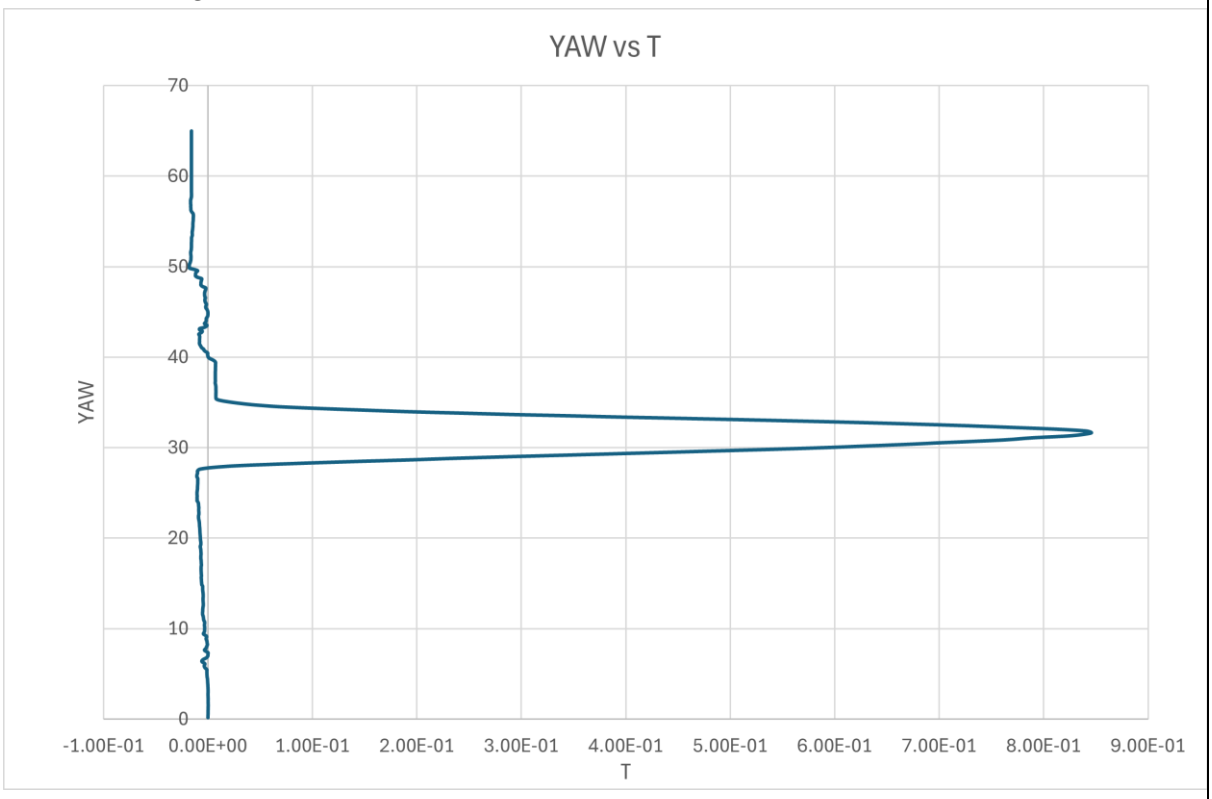
**2- X VS T**


X vs T
X
T
70
60
50
40
30
20
10
0
0
0.002
0.004
0.006
0.008
0.01
0.012
0.014
0.016
0.018
0.02

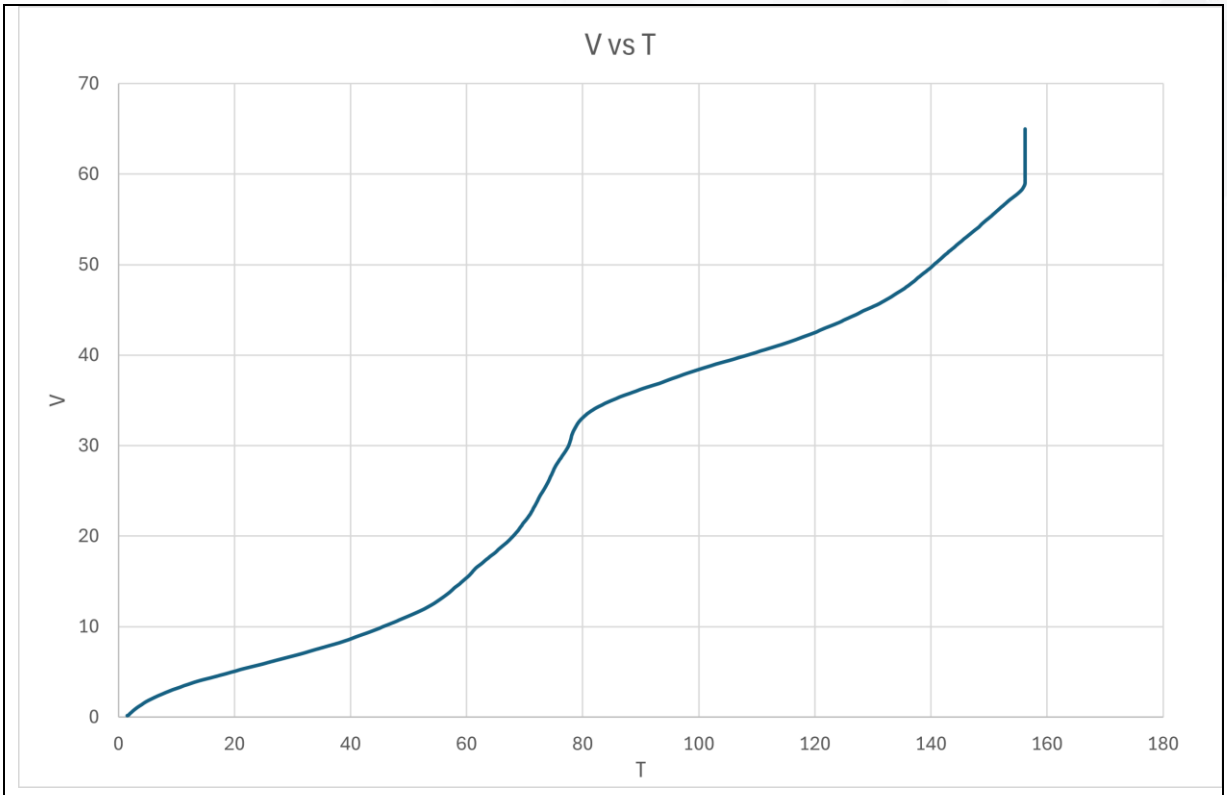
**3- Y VS T**



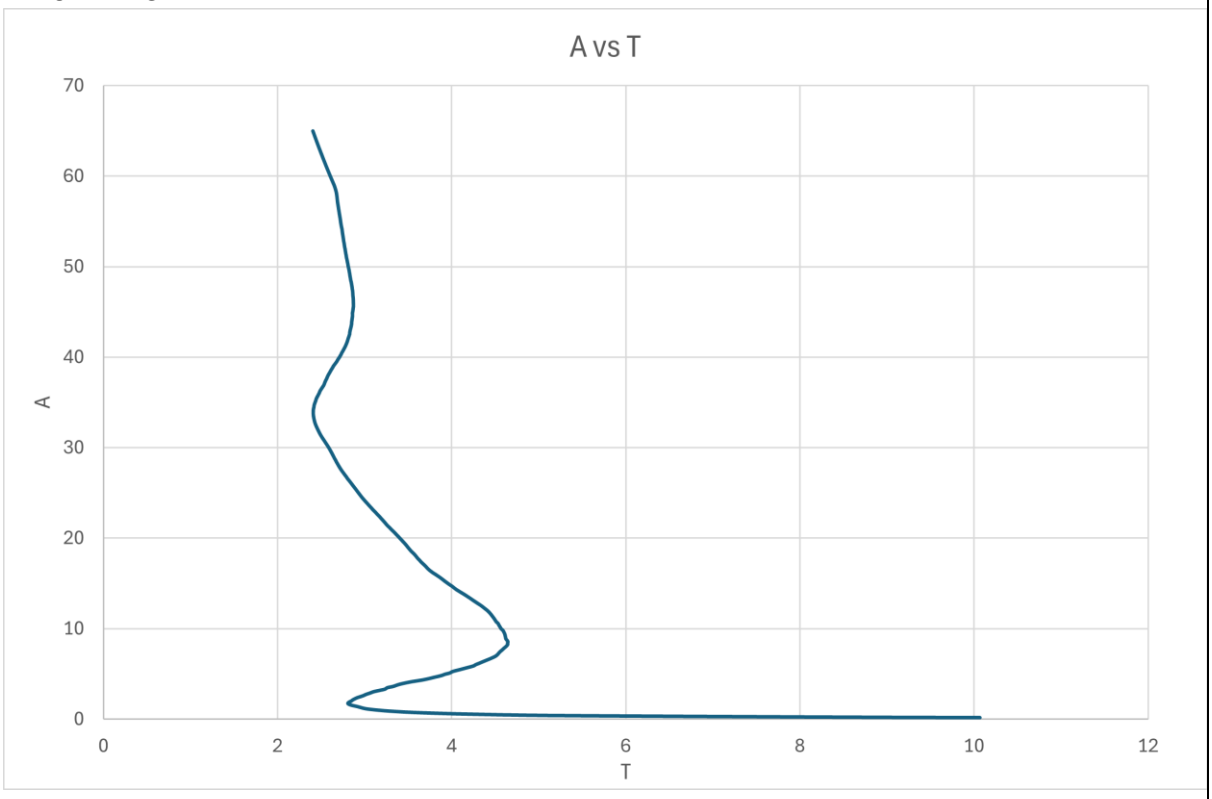

**4- YAW VS T**



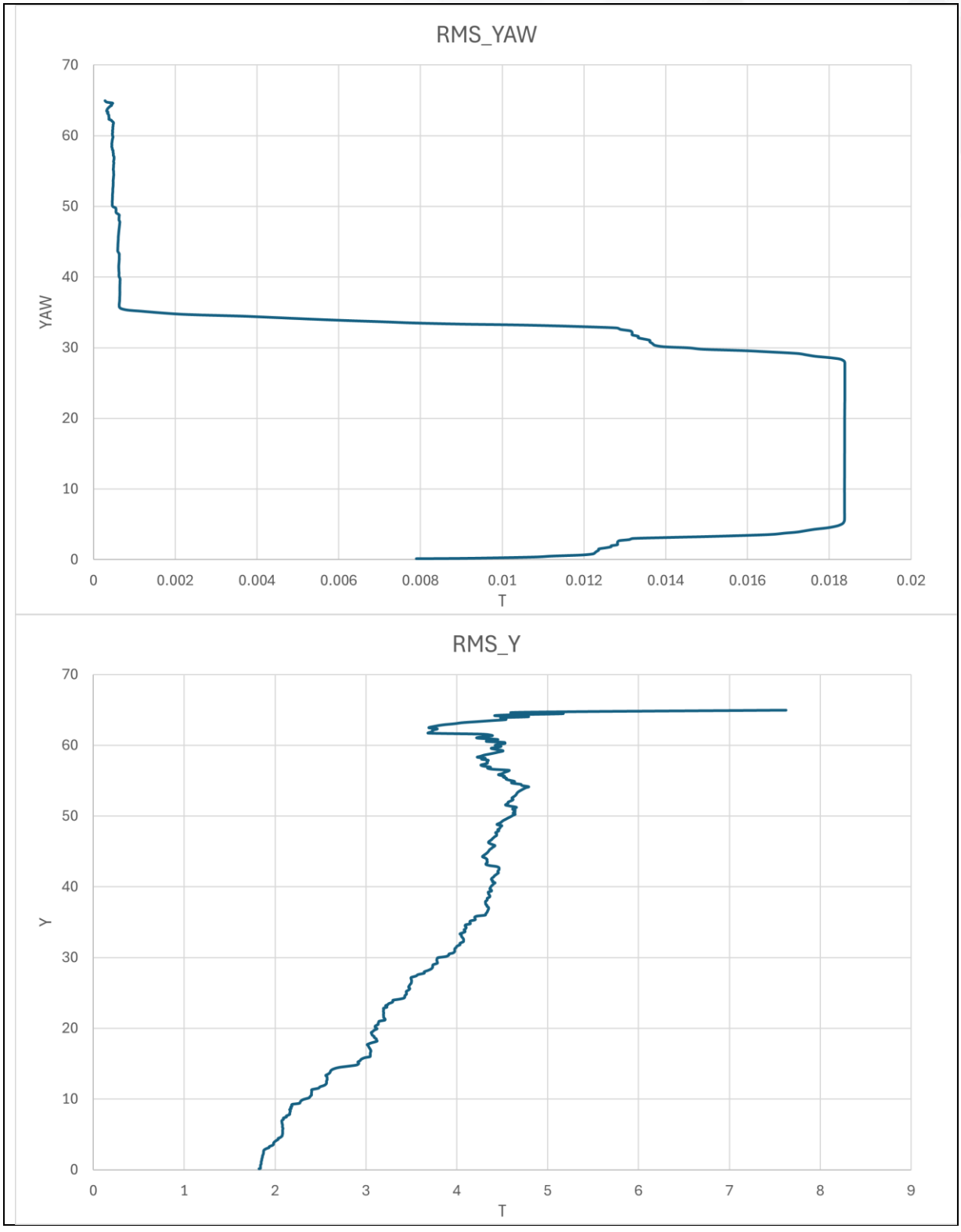

**5- V VS T**

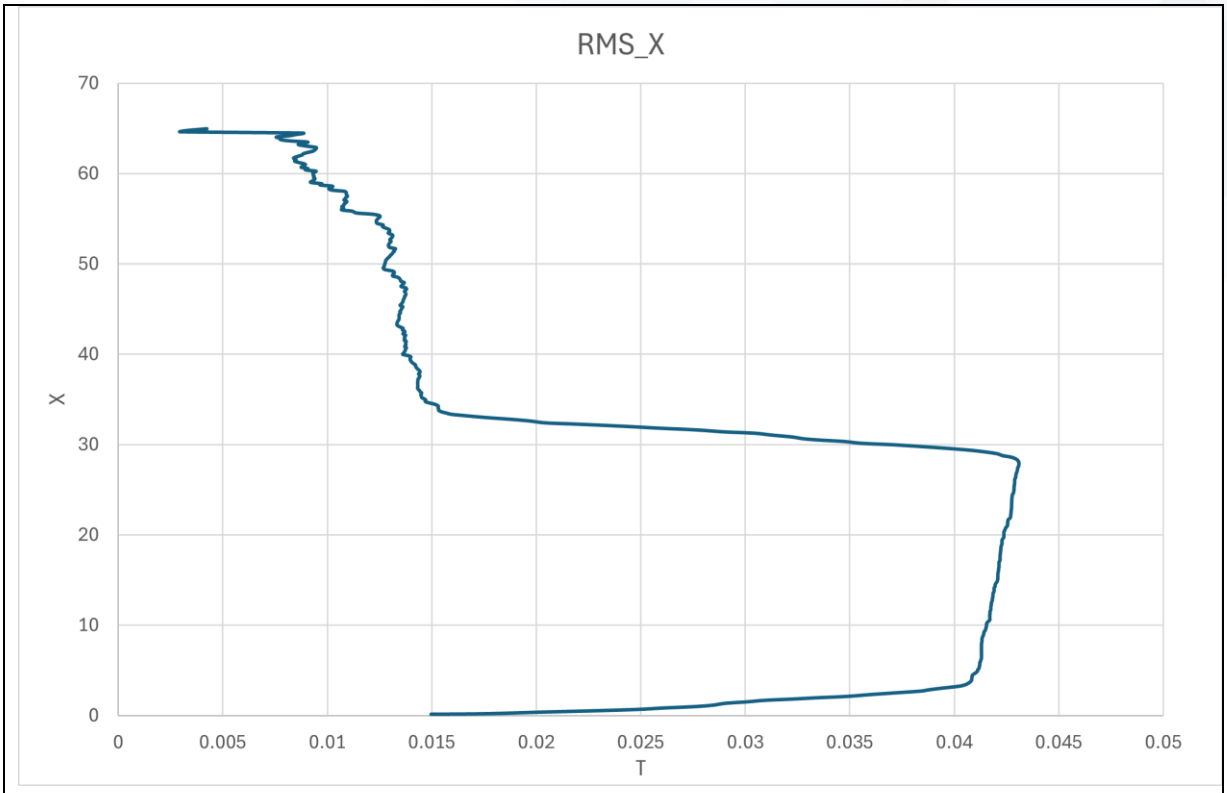



**6- A VS T**



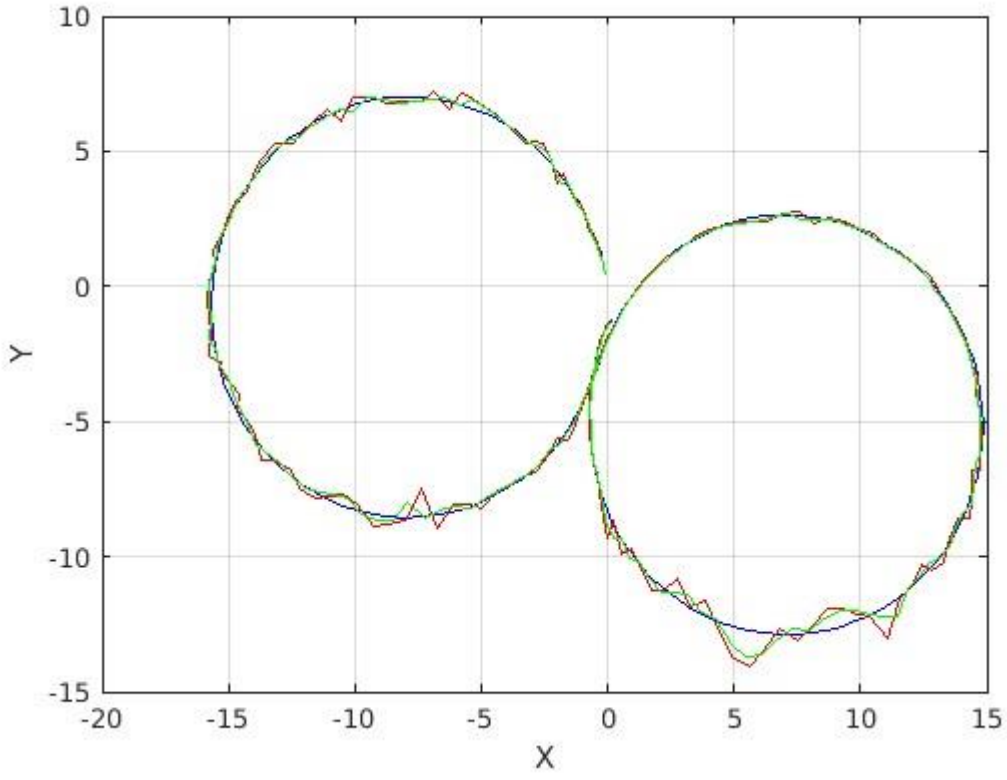

**7- RMS VS T**


RMS_YAW
YAW
T
0
10
20
30
40
50
60
70
0
0.002
0.004
0.006
0.008
0.01
0.012
0.014
0.016
0.018
0.02
RMS_Y
Y
T
0
10
20
30
40
50
60
70
0
1
2
3
4
5
6
7
8
9

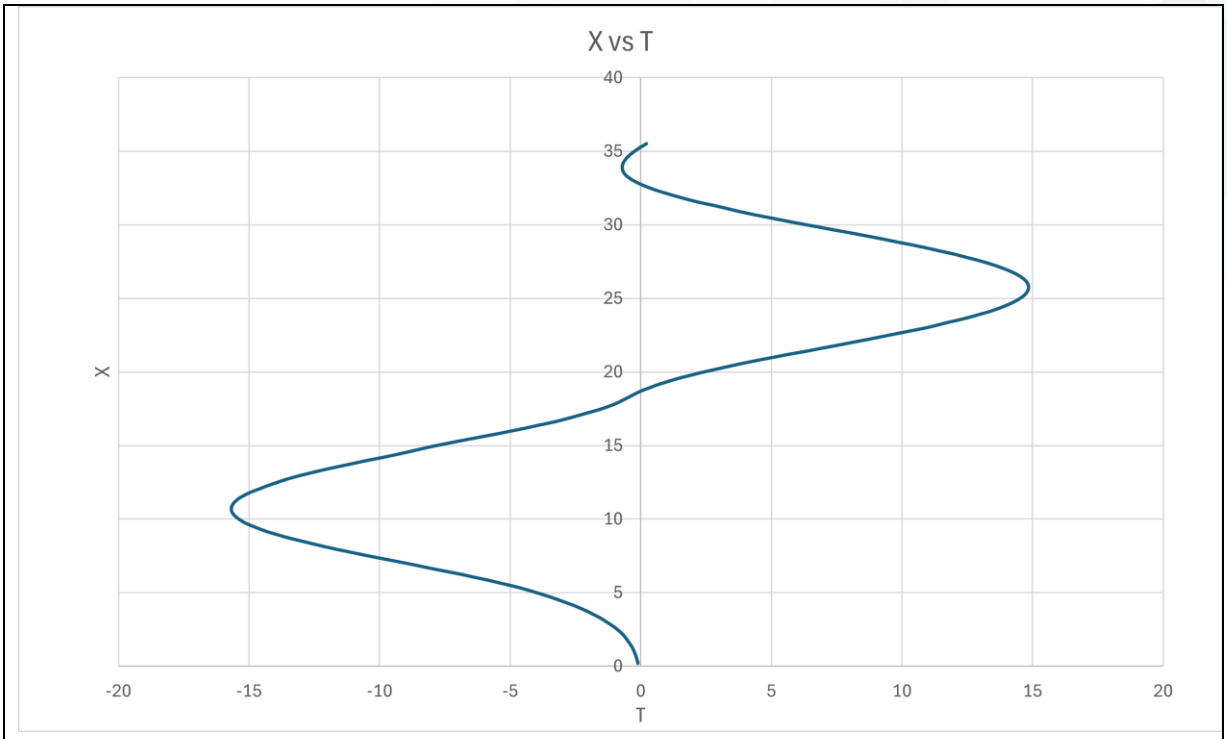



**Path4**

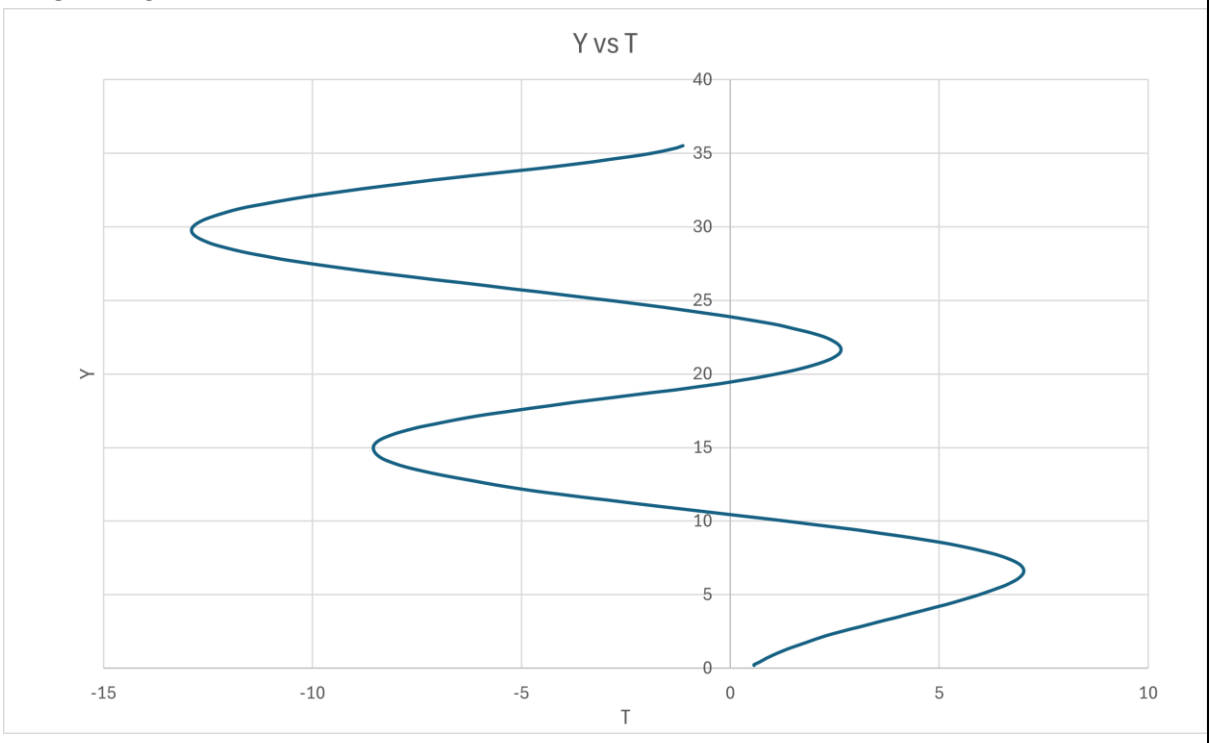
**1- X VS Y**



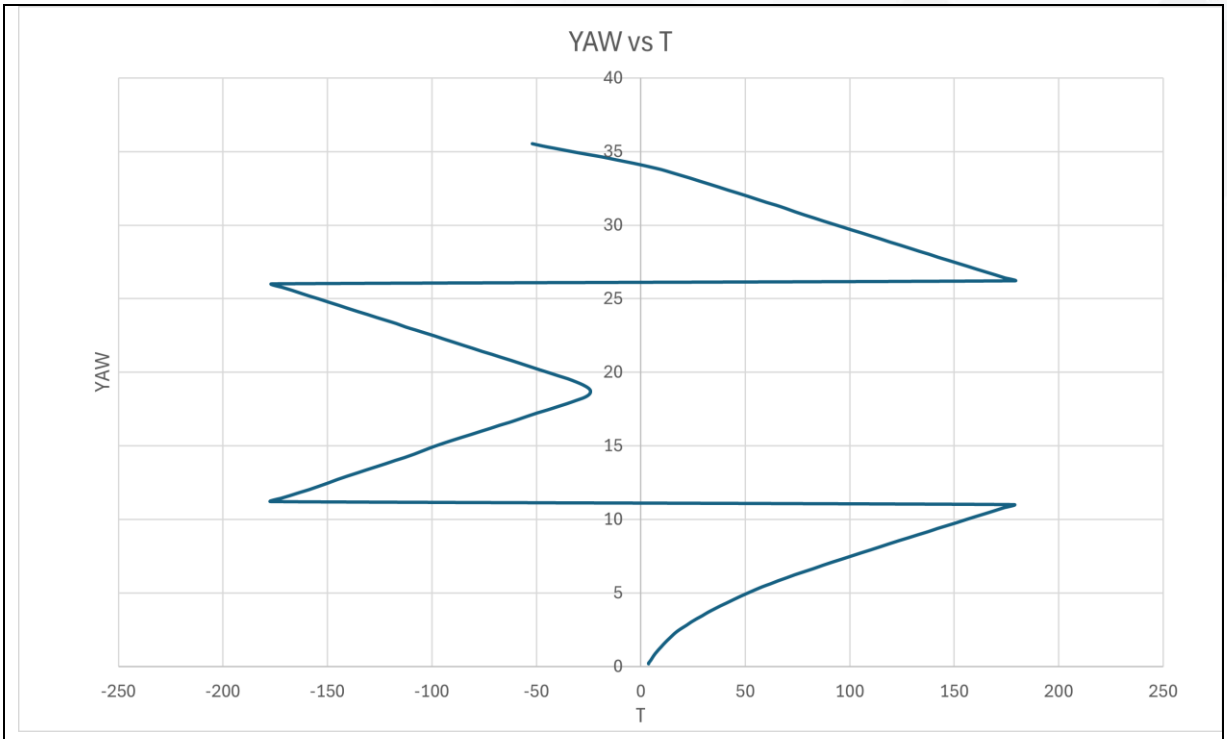

**2- X VS T**



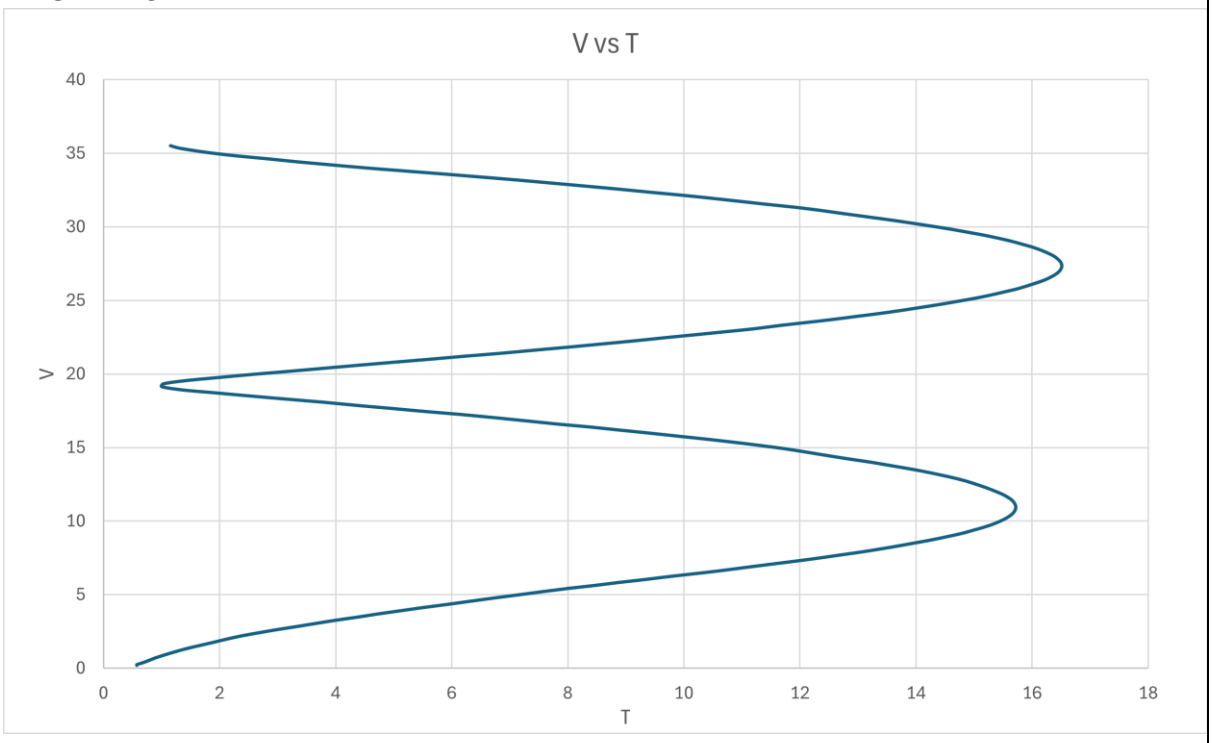

**3- Y VS T**



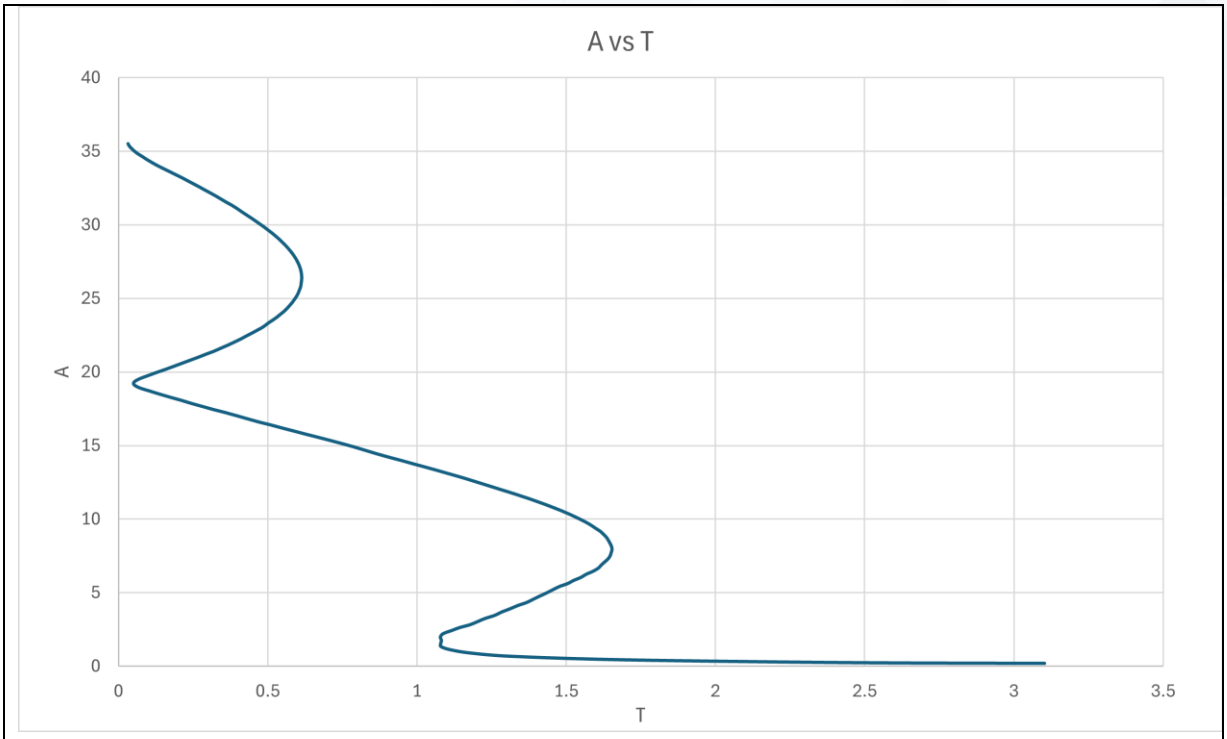

**4- YAW VS T**


YAW vs T
YAW
T
40
35
30
25
20
15
10
5
0
-250
-200
-150
-100
-50
0
50
100
150
200
250

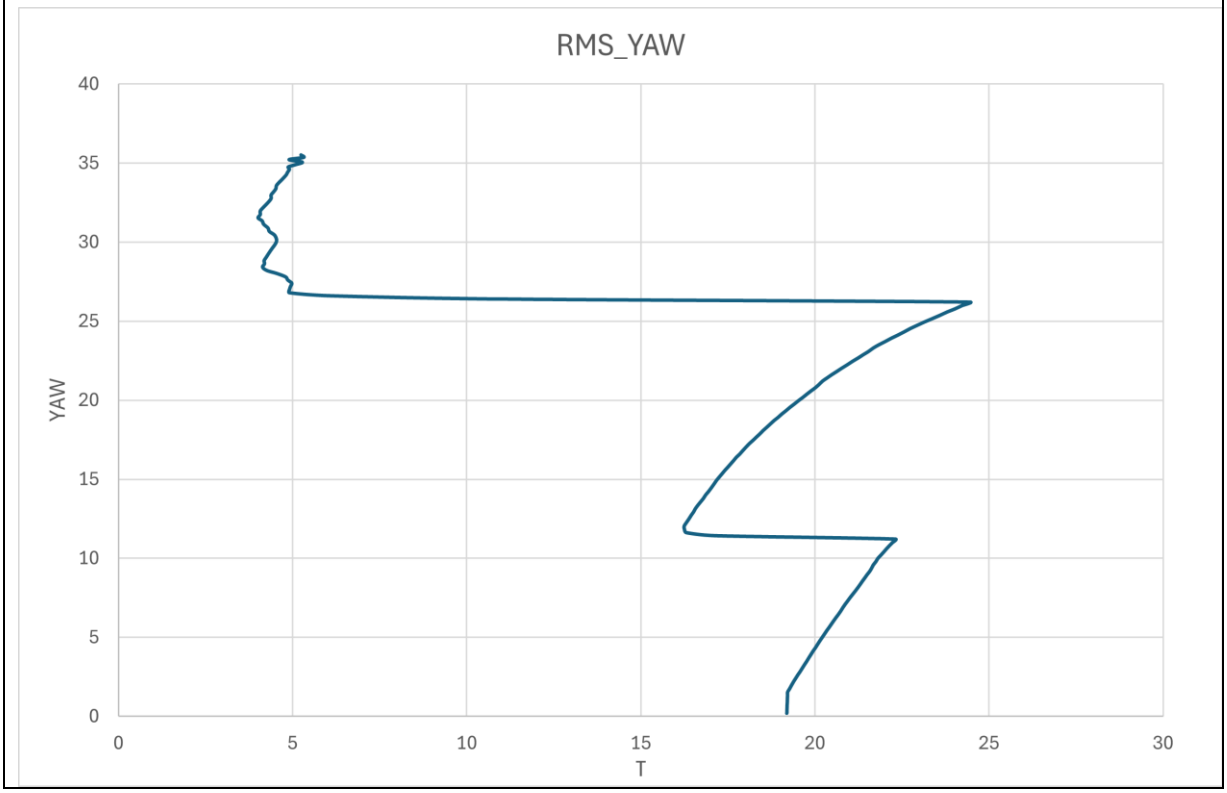
**5- V VS T**

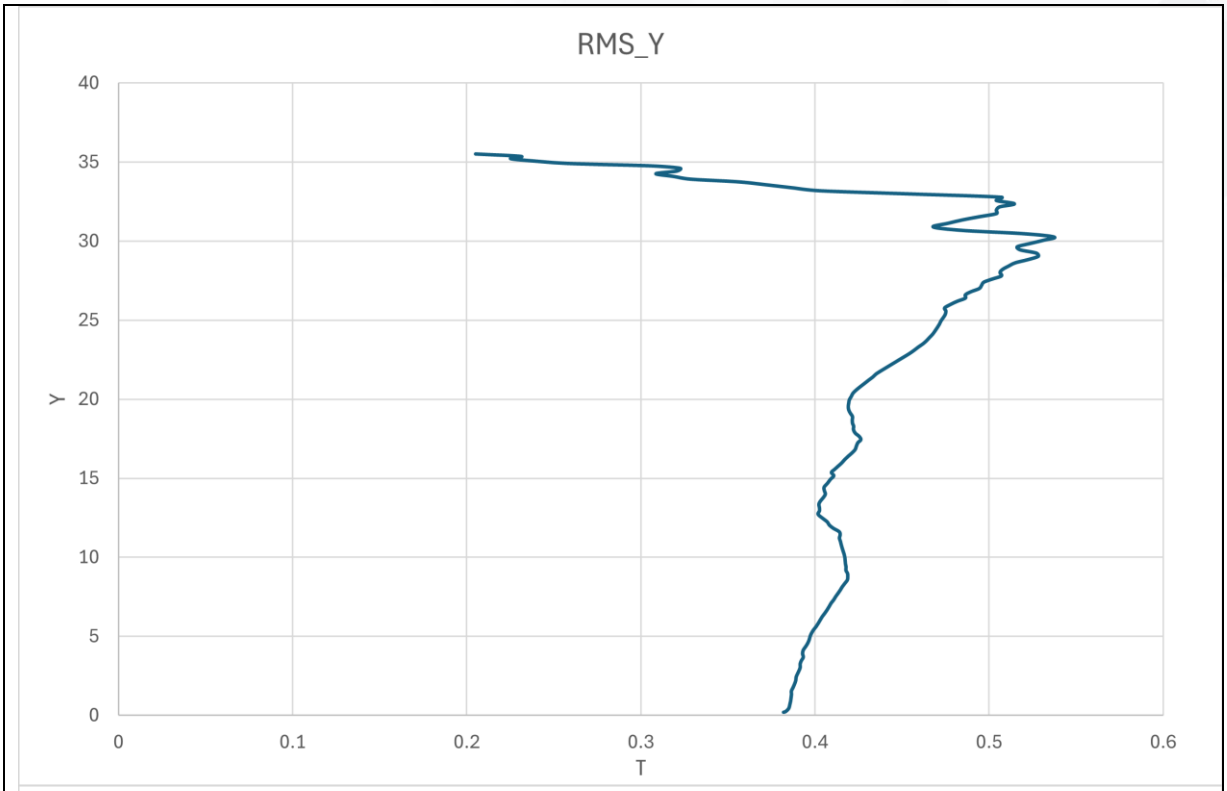

V vs T
V
T
40
35
30
25
20
15
10
5
0
0
2
4
6
8
10
12
14
16
18

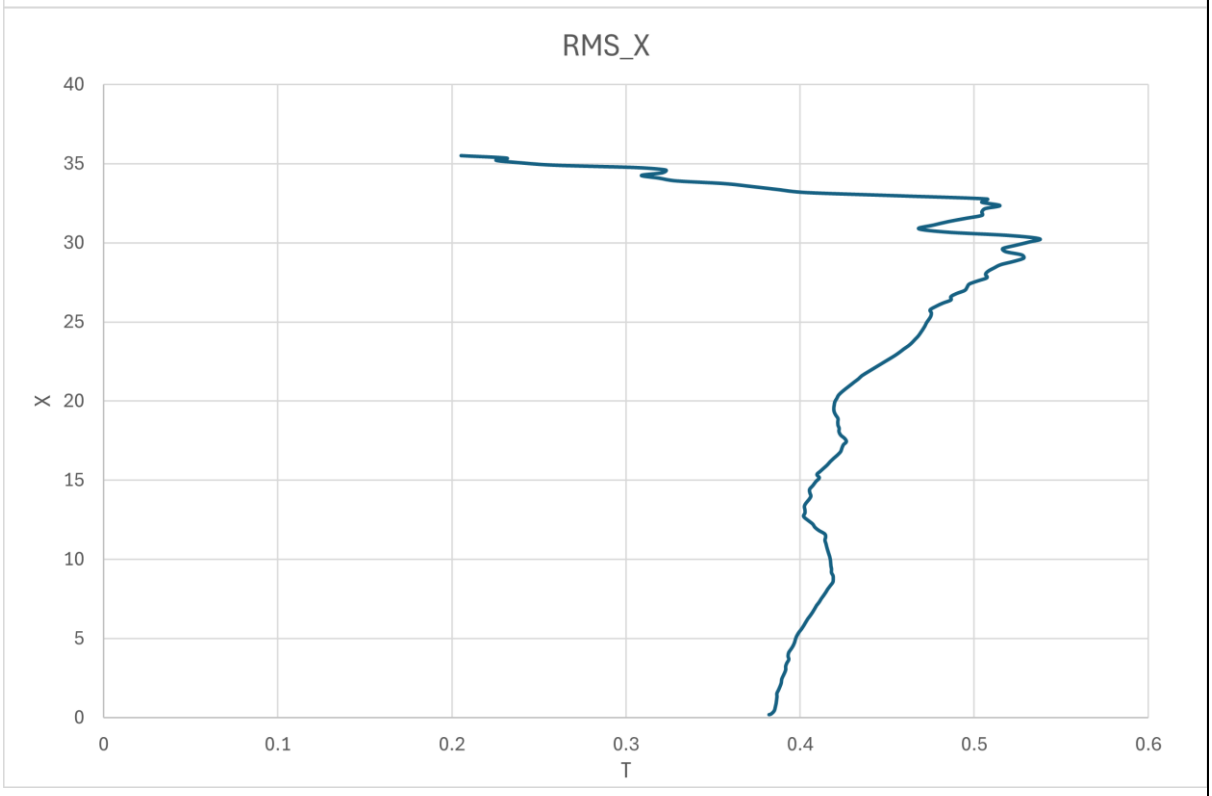
**6- A VS T**


A vs T
A
T
0
5
10
15
20
25
30
35
40
0
0.5
1
1.5
2
2.5
3
3.5

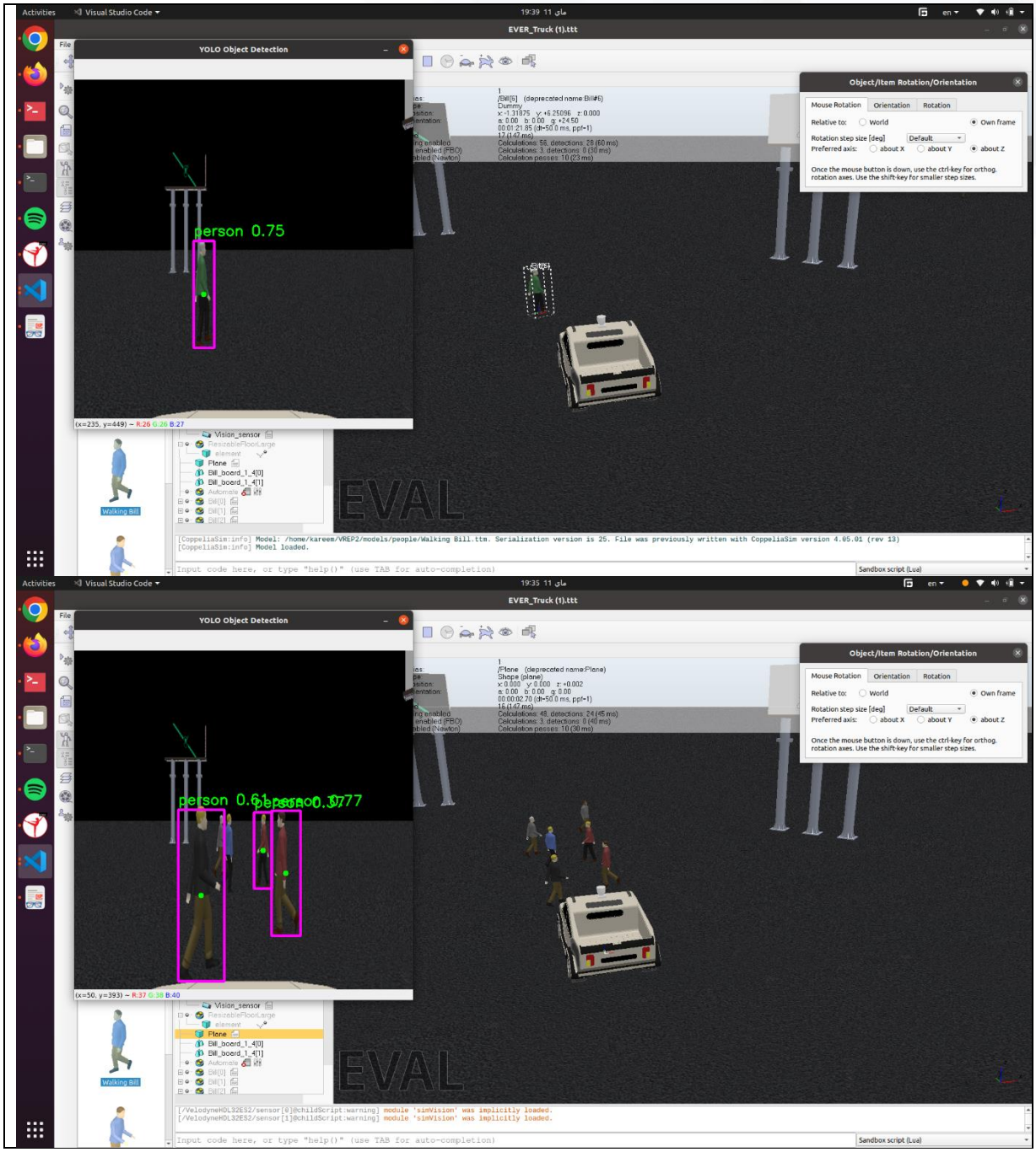
**7- RMS VS T**

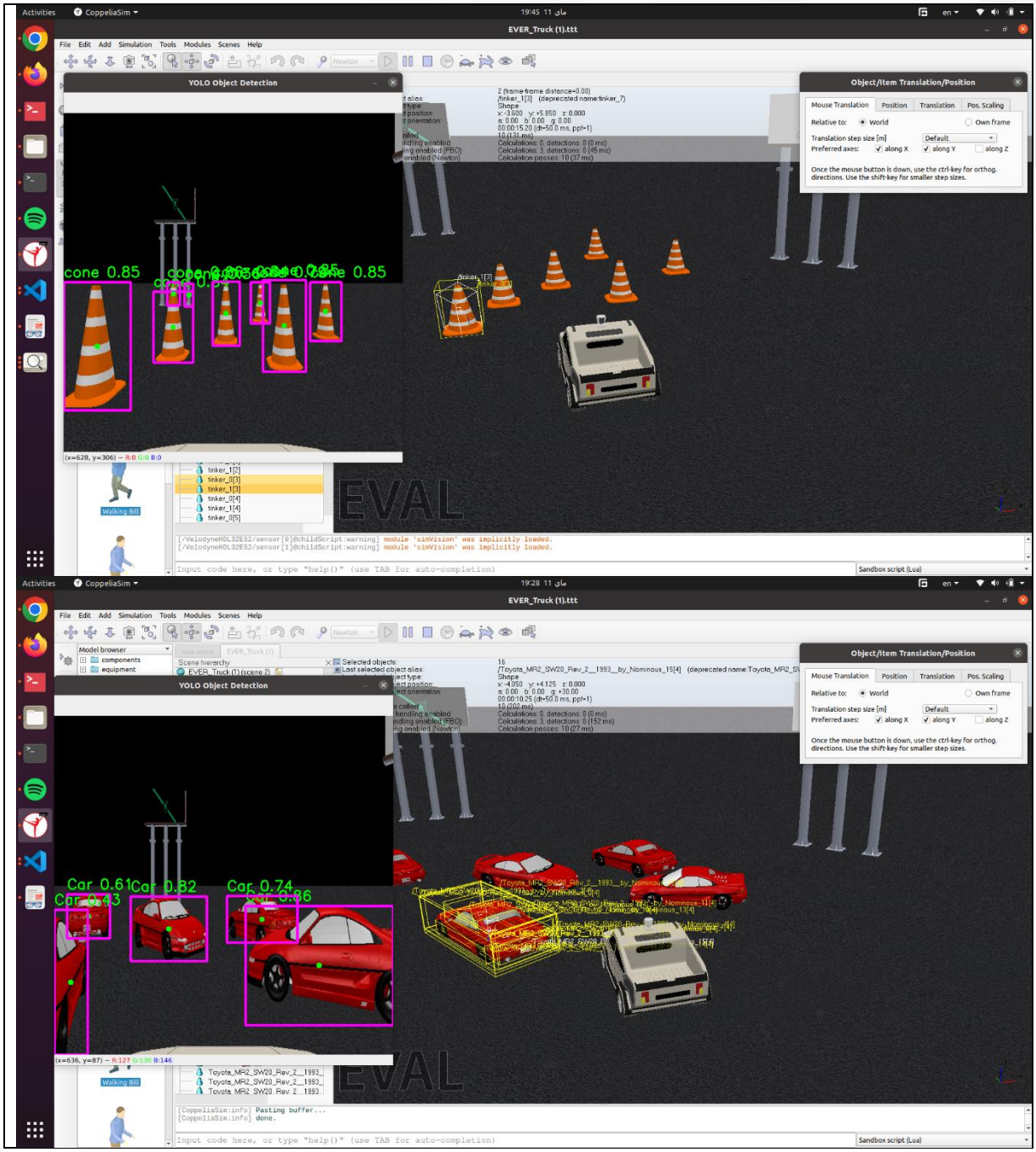

RMS_YAW
YAW
T
0
5
10
15
20
25
30
35
40
0
5
10
15
20
25
30

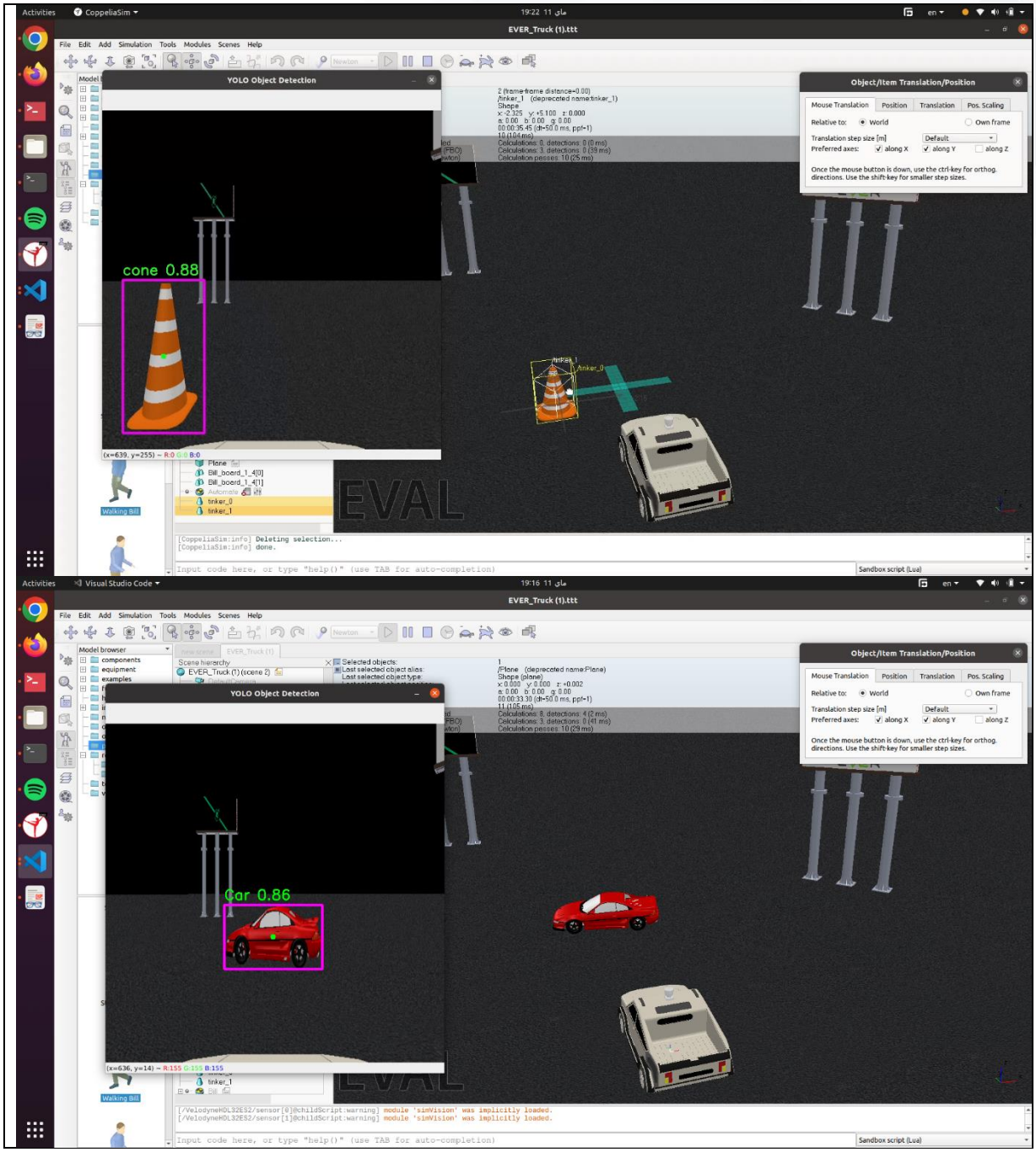

RMS_Y
40
35
30
25
20
15
10
5
0
Y
0
0.1
0.2
0.3
0.4
0.5
0.6
T

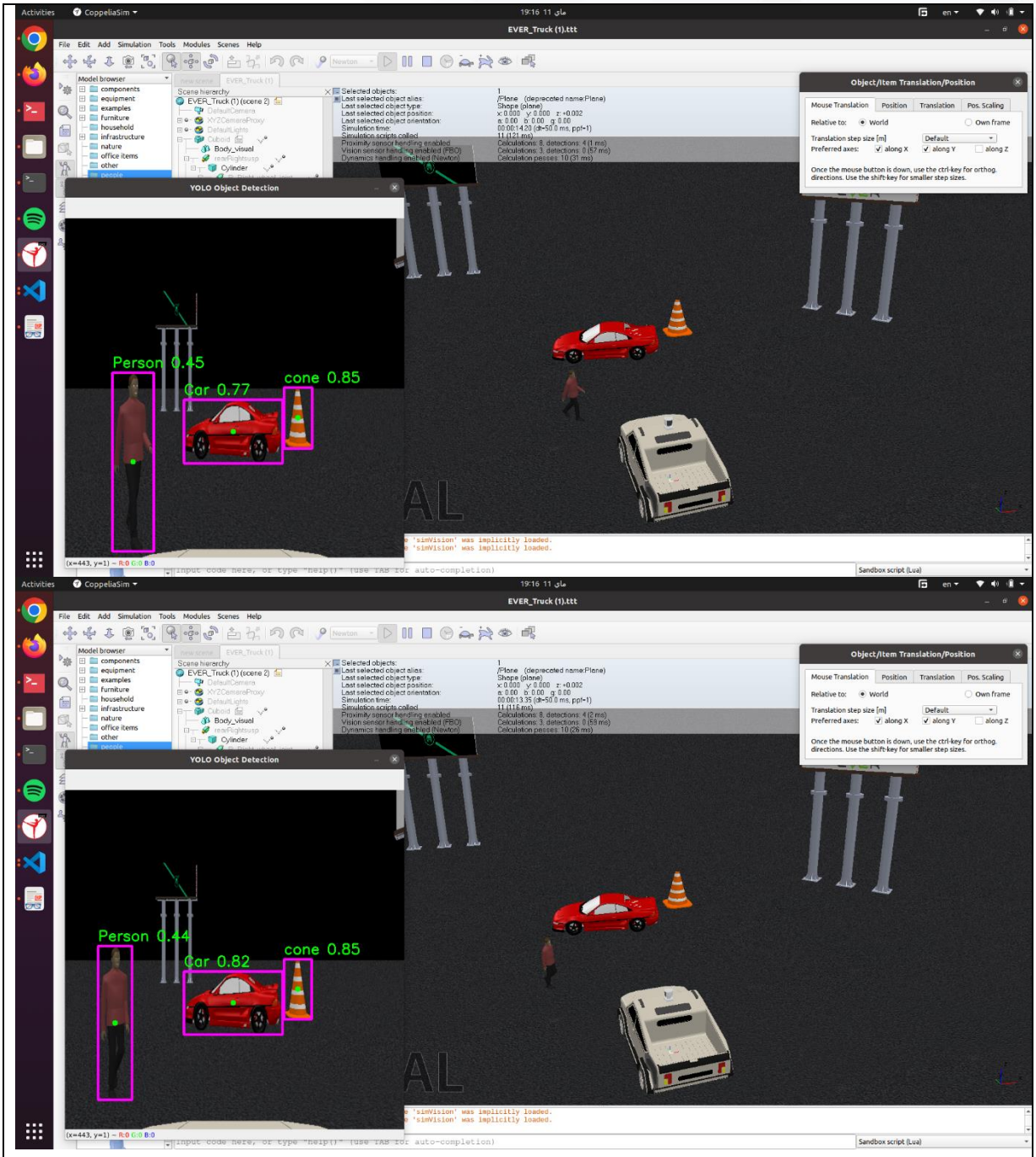

RMS_X
40
35
30
25
20
15
10
5
0
X
0
0.1
0.2
0.3
0.4
0.5
0.6
T

**Part2:**


Activities
Visual Studio Code
EVER_Truck (1).ttt
YOLO Object Detection
person 0.75
Object/Item Rotation/Orientation
Mouse Rotation
Orientation
Rotation
Relative to:
World
Own frame
Rotation step size [deg]
Default
Preferred axis:
about X
about Y
about Z
Once the mouse button is down, use the ctrl-key for orthog. rotation axes. Use the shift-key for smaller step sizes.
EVAL
Walking Bill
Input code here, or type "help()" (use TAB for auto-completion)
Sandbox script (Lua)
Activities
Visual Studio Code
EVER_Truck (1).ttt
YOLO Object Detection
person 0.61
person 0.77
Object/Item Rotation/Orientation
Mouse Rotation
Orientation
Rotation
Relative to:
World
Own frame
Rotation step size [deg]
Default
Preferred axis:
about X
about Y
about Z
Once the mouse button is down, use the ctrl-key for orthog. rotation axes. Use the shift-key for smaller step sizes.
EVAL
Walking Bill
Input code here, or type "help()" (use TAB for auto-completion)
Sandbox script (Lua)


EVER_Truck (1).ttt
YOLO Object Detection
Object/Item Translation/Position
cone 0.85
cone 0.85
EVER_Truck (1).ttt
YOLO Object Detection
Object/Item Translation/Position
Car 0.61
Car 0.82
Car 0.74
Car 0.86
Car 0.43


YOLO Object Detection
cone 0.88
Object/Item Translation/Position
EVAL
YOLO Object Detection
Car 0.86
Object/Item Translation/Position
EVAL




## The Written Code

**Part 1:**

**MATLAB Codes:**

1- We implemented a matlab code to try it's useful tools for a gaussian noise and Kalman filter and connected the matlab with ROS to subscribe the needed data, make the required transformations, add a gaussian noise to it, apply the Kalman filter, and publish the filtered data to control the car.

```
clear all;
clc;

rosshutdown
rosinit

[startpub, msg4] = rospublisher("/startSimulation", "std_msgs/Bool");
msg4.Data = 1;
send(startpub, msg4);

[odomPub1, msg1] = rospublisher("/odom1", "std_msgs/Float32MultiArray");
[odomPub2, msg2] = rospublisher("/odom2", "std_msgs/Float32MultiArray");
[odomPub3, msg3] = rospublisher("/odom3", "std_msgs/Float32MultiArray");


A = [ 1 0 0;
      0 1 0;
      0 0 1];

B = [0 1 1;
     0 1 1;
     1 0 0;];

H = [1;
     1;
     1];          % Observation matrix

Q = 0.629*eye(3);     % Process noise covariance to be applied to the whole matrix

R = 0.9*eye(3);           % Measurement noise covariance  old 0.1

x = [0;
     0;
     0];         % Initial state estimate for x, y directions and yaw.

P = eye(3);         % Initial error covariance to be large number


% quat to euler

function yaw_deg = quat2yaw(quat)

    q_x = quat(1);
    q_y = quat(2);
    q_z = quat(3);
    q_w = quat(4);

```

```
function yaw_deg = quat2yaw(quat)

    q_x = quat(1);
    q_y = quat(2);
    q_z = quat(3);
    q_w = quat(4);

    yaw_rad = atan2(2*(q_w*q_z + q_x*q_y), 1 - 2*(q_y^2 + q_z^2));
    yaw_deg = rad2deg(yaw_rad);
end


while 1

    odomSub = rossubscriber("/odom","DataFormat","struct");
    odomMsg = receive(odomSub,10);
    pose = odomMsg.Pose.Pose;
    vel_x = odomMsg.Twist.Twist.Linear.X;
    vel_y = odomMsg.Twist.Twist.Linear.Y;

    xposd = pose.Position.X;
    yposd = pose.Position.Y;
    orientation_x = pose.Orientation.X;
    orientation_y = pose.Orientation.Y;
    orientation_z = pose.Orientation.Z;
    orientation_w = pose.Orientation.W;

    Yaw = quat2yaw([orientation_x, orientation_y, orientation_z, orientation_w]);

    Data_X = xposd;
    Data_Y = yposd;
    Data_yaw = Yaw;

    xPointsM= awgn(Data_X, 43.428,"measured");  % old 0.0313 with 43.428 dB
    yPointsM= awgn(Data_Y, 25.9,"measured");
    yawPointsM= awgn(Data_yaw, 25.9,"measured");   %old 0.075 with 25.9 dB

    Data_XN = xPointsM;
    Data_YN = yPointsM;
    Data_yawN = yawPointsM;

    % Predictionstep
    x = A.*x ;                      % x = A.*x + B.*U;
    P = A.*P.*A' + Q;               % Predicted error covariance


    % Update step
    K = P.*H'/((H.*P).*H' + R);          % Kalman gain

```

```
    Data_XN = xPointsM;
    Data_YN = yPointsM;
    Data_yawN = yawPointsM;

    % Predictionstep
    x = A.*x ;                      % x = A.*x + B.*U;
    P = A.*P.*A' + Q;               % Predicted error covariance


    % Update step
    K = P.*H'/((H.*P).*H' + R);          % Kalman gain

    % K = eye(2).*H'/(H.*eye(2).*H' + R);  % Kalman gain (using identity matrix for P)

    x = x + K.*([Data_XN;Data_YN;Data_yawN] - H.*x); % Updated state estimate
    P = P - K.*(H.*P);           % Updated error covariance  (use TIMES (.) for elementwise multiplication.)

    filtered_xPoints = x(1);    % Store filtered x
    filtered_yPoints = x(2,2);    % Store filtered y
    filtered_yawPoints = x(3,3);    % Store filtered yaw


    k1 = yposd
    k2 = yPointsM
    k3_1 = filtered_xPoints;
    k3_2 = filtered_yPoints
    k3_3 = filtered_yawPoints;

    msg1.Data = [k3_1, k3_2, k3_3];
    msg2.Data = [xposd, yposd, Yaw, xPointsM, yPointsM, yawPointsM, filtered_xPoints, filtered_yPoints, filtered_yawPoints];

    send(odomPub1, msg1);  %publishing for path
    send(odomPub2, msg2);  %publishing for CSV file

    pause(0.1);

end
```

2- We also used a matlab code to generate a plots from the CSV file Data

for closed loop paths.
it's clean the data from any NaN values and generates a plot for what we need.

Open

closed_loop_pathplot.m
~/Downloads/Milestone2

```
clear all;

filename = '/home/eslam/catkin_workspace/src/t2/scripts/Dimensions2_4.csv';
data = readtable(filename);

columnData_X = data.positions_x;
columnData_X_clean = columnData_X(~isnan(columnData_X));

columnData_Y = data.positions_y;
columnData_Y_clean = columnData_Y(~isnan(columnData_Y));

columnData_YAW = data.yaw;
columnData_YAW_clean = columnData_YAW(~isnan(columnData_YAW));

columnData_X_noised = data.Noisy_x;
columnData_X_noised_clean = columnData_X_noised(~isnan(columnData_X_noised));

columnData_Y_noised = data.Noisy_y;
columnData_Y_noised_clean = columnData_Y_noised(~isnan(columnData_Y_noised));

columnData_YAW_noised = data.Noisy_yaw;
columnData_YAW_noised_clean = columnData_YAW_noised(~isnan(columnData_YAW_noised));


columnData_X_filtered = data.filtered_x;
columnData_X_filtered_clean = columnData_X_filtered(~isnan(columnData_X_filtered));

columnData_Y_filtered = data.filtered_y;
columnData_Y_filtered_clean = columnData_Y_filtered(~isnan(columnData_Y_filtered));

columnData_YAW_filtered = data.filtered_yaw;
columnData_YAW_filtered_clean = columnData_YAW_filtered(~isnan(columnData_YAW_filtered));

columnData_time = data.Time_Sec;
columnData_time_clean = columnData_time(~isnan(columnData_time));

%true value plot
plot(columnData_X_clean,columnData_Y_clean , '-b');
xlabel('X');
ylabel('Y');
grid on;
hold on;

%noise plot
plot(columnData_X_noised_clean,columnData_Y_noised_clean, '-r');
hold on;

%filter plot
plot(columnData_X_filtered_clean,columnData_Y_filtered_clean, '-g');
hold on;
```

3- We used the same Idea for the previous code to generate the open

loop data plots and get the new CSV file for the data noise.

```
filename = '/home/eslam/catkin_workspace/src/t2/scripts/line_with_turn.xlsx';
newFilename = '/home/eslam/catkin_workspace/src/t2/scripts/NoisyOdom3.csv';

header = {'Noisy_X', 'Noisy_Y', 'Time'};

% Read the data from the CSV file
data = readtable(filename);
Noise_data = readtable(newFilename);

columnData_X = data.positions_X;
columnData_X_clean = columnData_X(~isnan(columnData_X));

columnData_Y = data.positions_y;
columnData_Y_clean = columnData_Y(~isnan(columnData_Y));

%columnData_yaw = data.Yaw;

columnData_time = data.Time_Sec;
columnData_time_clean = columnData_time(~isnan(columnData_time));


columnData_NX = Noise_data.Noisy_X;
columnData_NY = Noise_data.Noisy_Y;
%columnData_NYaw = Noise_data.Noisy_Yaw;

xPointsM= awgn(columnData_X_clean, 43.428, "measured");     % old = 0.0313
yPointsM= awgn(columnData_Y_clean, 46.3, "measured");       % old = 46.3
%yawPointsM= awgn(columnData_yaw,0.075,"measured");

% acual data plotting
plot(columnData_X_clean,columnData_Y_clean , '-b');
xlabel('X');
ylabel('Y');
grid on;
hold on;

% noise data plotting
plot(xPointsM,yPointsM , '-r');
hold on;

dataWithHeader = [header; num2cell(xPointsM), num2cell(yPointsM), num2cell(columnData_time_clean)];

writecell(dataWithHeader, newFilename);
```

**1- Path1**

```python
#!/usr/bin/env python
import rospy
from std_msgs.msg import Float64, Bool
from nav_msgs.msg import Odometry

class CarController:
    def __init__(self):
        rospy.init_node('car_controller', anonymous=True)
        self.pub_SteeringAngle = rospy.Publisher('SteeringAngle', Float64, queue_size=10)
        self.pub_cmd_vel = rospy.Publisher('cmd_vel', Float64, queue_size=10)
        self.pub_brakes = rospy.Publisher('brakes', Float64, queue_size=10)
        self.odom_sub = rospy.Subscriber('odom', Odometry, self.odom_callback)
        self.brake_value =1.6*pow(10, -47.2)
        self.Kp = 0.79
        self.Ki = 0.005
        self.Kd = 5
        self.set_point = 75.0
        self.error = 0.0
        self.integral = 0.0
        self.derivative = 0.0
        self.last_error = 0.0
        self.output = 0.0
        self.last_time = rospy.get_time()

        self.current_distance = 0.0

    def start_simulation(self):
        """Publishes a command to start the simulation."""
        simulation_pub = rospy.Publisher('/startSimulation', Bool, queue_size=1)
        while simulation_pub.get_num_connections() == 0:
            rospy.sleep(0.1)
        simulation_pub.publish(True)
        rospy.loginfo("Simulation started by node")

    def compute_pid(self):
        """Calculate PID value for given reference feedback."""
        current_time = rospy.get_time()
        dt = current_time - self.last_time if self.last_time != 0 else 0.1
        self.error = self.set_point - self.current_distance
        self.integral += self.error * dt
```

```
        self.derivative = (self.error - self.last_error) / dt if dt > 0 else 0

        self.output = self.Kp * self.error + self.Ki * self.integral + self.Kd * self.derivative
        self.last_error = self.error
        self.last_time = current_time
        return self.output

    def odom_callback(self, msg):
        """Updates the current distance based on odometry data."""
        self.current_distance = msg.pose.pose.position.y

    def run(self):
        """Main method to control the car using PID."""
        self.start_simulation()
        rate = rospy.Rate(10)  # 10 Hz
        while not rospy.is_shutdown():
            velocity = self.compute_pid()
            print(velocity,"    ",self.current_distance)
            if velocity > 1.0:
                velocity = 1.0
                self.pub_brakes.publish(0)
            elif velocity < 0.0:
                velocity = 0.0
                self.pub_brakes.publish(self.brake_value)

            self.pub_cmd_vel.publish(velocity)
            print(velocity)
            self.pub_SteeringAngle.publish(0.0)  # Assuming straight line movement

            if self.current_distance >= self.set_point:
                rospy.loginfo("Target distance reached. Stopping.")
                self.pub_cmd_vel.publish(0.0)
                self.pub_brakes.publish(self.brake_value)
                print("done")
                break
            rate.sleep()

if __name__ == '__main__':
    try:
        controller = CarController()
        controller.run()
    except rospy.ROSInterruptException:
        rospy.loginfo("Node was interrupted")
```

## 2- Path2

```python
#!/usr/bin/env python

import rospy
from std_msgs.msg import String, Float64, Int16, Bool, Int32, Float32MultiArray
from nav_msgs.msg import Odometry
from sensor_msgs.msg import Imu
import time
import math

start = time.time()
# d = 37.714   2*pi*r          d = 0.5*a*t^2

class MyNode:
    def __init__(self):
        rospy.init_node('path2_node', anonymous=True)
        self.pub = rospy.Publisher('SteeringAngle', Float64, queue_size=10)
        self.pub2 = rospy.Publisher('cmd_vel', Float64, queue_size=10)
        self.pub3 = rospy.Publisher('brakes', Float64, queue_size=10)
        self.pub4 = rospy.Publisher('startSimulation', Bool, queue_size=10)
        self.pub5 = rospy.Publisher('pauseSimulation', Bool, queue_size=10)


        self.error_sum = 0
        self.last_error = 0

        self.Kp = 0.1
        self.Ki = 0
        self.Kd = 8
        self.desired_radius = 6
        self.error = 0
        self.integral = 0
        self.derivative = 0
        self.last_error = 0
        self.desired_steering_angle = 18.5
        self.last_time = rospy.get_time()

    def compute_pid(self):

        current_time = rospy.get_time()
```

```
        dt = current_time - self.last_time if self.last_time != 0 else 0.1
        self.error = self.desired_radius - C
        self.integral += self.error * dt
        self.derivative = (self.error - self.last_error) / dt if dt > 0 else 0

        self.steering_angle = self.desired_steering_angle + self.Kp * self.error + self.Ki * self.integral + self.Kd * self.derivative
        self.last_error = self.error
        self.last_time = current_time

        return self.steering_angle

    def odom_callback(self, point):

        global xpose, ypose, yaw,C,state

        xpose = point.data[0]
        ypose = point.data[1]
        yaw = point.data[2]


        C = math.sqrt((xpose+6)**2 + ypose**2)    #desired = 6
        state = xpose**2 + ypose**2

        rospy.loginfo("x position =  {}".format(xpose))
        rospy.loginfo("y position =  {}".format(ypose))
        rospy.loginfo("yaw =  {}".format(yaw))

        rospy.loginfo("c =  {}".format(state))

        new_steering_angle = self.compute_pid()

        self.pub.publish(new_steering_angle)

        if state < 150 and ypose<-4 and ypose>-5:               #if x > and y < -1 and y>-2:
            gaspedal = 0
            self.pub2.publish(gaspedal)
            brakepedal= (time.time() - start) * 1.37* pow(10, -48)
            self.pub3.publish(brakepedal)
            rospy.signal_shutdown("Condition met")

    def od_subscriber(self):
        rospy.Subscriber("/odom1", Float32MultiArray, self.odom_callback)

    def run(self):
        rate = rospy.Rate(1)  # 1 Hz

        while not rospy.is_shutdown():

            steering_angle = 18.5  # Change this value as needed
            gaspedal = 0.23
            brakepedal = 0
            startsim = True

            #rospy.loginfo("Publishing: {}".format(data_to_publish))
            self.pub4.publish(Bool(startsim))
            self.pub.publish(steering_angle)
            self.pub2.publish(gaspedal)
            self.pub3.publish(brakepedal)
            my_node.od_subscriber()

            rate.sleep()


if __name__ == '__main__':
    my_node = MyNode()
    try:
        my_node.run()
    except rospy.ROSInterruptException:
        pass
```

### 3- Path3

```python
#!/usr/bin/env python
import rospy
from std_msgs.msg import Float64, Bool, Float32MultiArray
from nav_msgs.msg import Odometry
import tf
class CarController:
    def __init__(self):
        rospy.init_node('car_controller', anonymous=True)
        self.pub_SteeringAngle = rospy.Publisher('SteeringAngle', Float64, queue_size=10)
        self.pub_cmd_vel = rospy.Publisher('cmd_vel', Float64, queue_size=10)
        self.pub_brakes = rospy.Publisher('brakes', Float64, queue_size=10)
        self.odom_sub = rospy.Subscriber('/odom1', Float32MultiArray, self.odom_callback)
        self.brake_value =1.6*pow(10, -47.2)
        self.Kp = 0.79
        self.Ki = 0.005
        self.Kd = 5
        self.set_point = 75.0
        self.error = 0.0
        self.integral = 0.0
        self.derivative = 0.0
        self.last_error = 0.0
        self.output = 0.0
        self.steer_flag =0
        self.last_time = rospy.get_time()

        self.current_distance = 0.0

    def start_simulation(self):
        """Publishes a command to start the simulation."""
        simulation_pub = rospy.Publisher('/startSimulation', Bool, queue_size=1)
        while simulation_pub.get_num_connections() == 0:
            rospy.sleep(0.1)
        simulation_pub.publish(True)
        rospy.loginfo("Simulation started by node")

    def compute_pid(self,setpoint):
        """Calculate PID value for given reference feedback."""
        if (self.steer_flag != 1):
            self.integral = 0.0
            self.last_error = 0.0
        current_time = rospy.get_time()
        dt = current_time - self.last_time if self.last_time != 0 else 0.1
        self.error = setpoint - self.current_distance
        self.integral += self.error * dt
        self.derivative = (self.error - self.last_error) / dt if dt > 0 else 0

        self.output = self.Kp * self.error + self.Ki * self.integral + self.Kd * self.derivative
        self.last_error = self.error
        self.last_time = current_time
```

```
50          self.steer_flag = 1
51          return self.output
52      def compute_steer_pid(self,setpoint):
53          if (self.steer_flag != 1):
54              self.integral = 0.0
55              self.last_error = 0.0
56          current_time = rospy.get_time()
57          dt = current_time - self.last_time if self.last_time != 0 else 0.1
58          self.error = abs(setpoint - self.current_lateral_position)
59          self.integral += self.error * dt
60          self.derivative = (self.error - self.last_error) / dt if dt > 0 else 0
61
62          self.output = self.Kp * self.error + self.Ki * self.integral + self.Kd * self.derivative
63          self.last_error = self.error
64          self.last_time = current_time
65          self.steer_flag =1
66          return self.output
67
68      def odom_callback(self, arr):
69          """Updates the current distance based on odometry data."""
70          self.current_distance = arr.data[1]
71          self.current_lateral_position = arr.data[0]
72          self.current_yaw = arr.data[2]
73
74      def run(self,setpoint):
75          """Main method to control the car using PID."""
76          self.start_simulation()
77          rate = rospy.Rate(10)  # 10 Hz
78          while not rospy.is_shutdown():
79              velocity = self.compute_pid(setpoint)
80              print(velocity,"    ",self.current_distance)
81              if velocity > 1.0:
82                  velocity = 1.0
83                  self.pub_brakes.publish(0)
84              elif velocity < 0.0:
85                  velocity = 0.0
86                  self.pub_brakes.publish(self.brake_value)
87
88              self.pub_cmd_vel.publish(velocity)
89              print(velocity)
90              self.pub_SteeringAngle.publish(0.0)  # Assuming straight line movement
91
92              if self.current_distance >= setpoint:
93                  rospy.loginfo("Target distance reached. Stopping.")
94                  self.pub_cmd_vel.publish(0.0)
95                  self.pub_brakes.publish(self.brake_value)
96                  print("done")
97                  break
98              rate.sleep()
```

```
 99      def steer(self,setpoint,angle):
100          rate = rospy.Rate(10)  # 10 Hz
101          while not rospy.is_shutdown():
102
103              velocity = self.compute_steer_pid(setpoint-3)
104              print(velocity,"   ",self.current_lateral_position)
105              if velocity > 1.0:
106                  velocity = 1.0
107                  self.pub_brakes.publish(0)
108              elif velocity < 0.0:
109                  velocity = 0.0
110                  self.pub_brakes.publish(self.brake_value)
111              self.pub_SteeringAngle.publish(angle-self.current_yaw*57.32)
112              print(angle-self.current_yaw)
113              self.pub_cmd_vel.publish(velocity)
114              print(velocity)
115
116
117              if self.current_lateral_position <= setpoint:
118                  rospy.loginfo("Target distance reached. Stopping.")
119                  self.pub_cmd_vel.publish(0.0)
120                  self.pub_brakes.publish(self.brake_value)
121                  print("done")
122                  break
123              rate.sleep()
124              if self.current_distance > 85 :
125                  break
126
127
128  if __name__ == '__main__':
129      try:
130          controller = CarController()
131          controller.run(75)
132          controller.steer_flag = 0
133          controller.steer(-3.5/2,90)
134          controller.steer_flag = 0
135          controller.steer(-10,0)
136          controller.steer_flag = 0
137          controller.run(153.5)
138      except rospy.ROSInterruptException:
139          rospy.loginfo("Node was interrupted")
```

**4- Path4**

```python
#!/usr/bin/env python
import rospy
from std_msgs.msg import Float64, Bool, Float32MultiArray
from nav_msgs.msg import Odometry
import tf
class CarController:
    def __init__(self):
        rospy.init_node('car_controller', anonymous=True)
        self.pub_SteeringAngle = rospy.Publisher('SteeringAngle', Float64, queue_size=10)
        self.pub_cmd_vel = rospy.Publisher('cmd_vel', Float64, queue_size=10)
        self.pub_brakes = rospy.Publisher('brakes', Float64, queue_size=10)
        self.odom_sub = rospy.Subscriber("/odom1", Float32MultiArray, self.odom_callback)
        self.brake_value =1.6*pow(10, -47.2)
        self.Kp = 0.79
        self.Ki = 0.005
        self.Kd = 5
        self.set_point = 75.0
        self.error = 0.0
        self.integral = 0.0
        self.derivative = 0.0
        self.last_error = 0.0
        self.output = 0.0
        self.steer_flag =0
        self.last_time = rospy.get_time()

        self.current_distance = 0.0

    def start_simulation(self):
        """Publishes a command to start the simulation."""
        simulation_pub = rospy.Publisher('/startSimulation', Bool, queue_size=1)
        while simulation_pub.get_num_connections() == 0:
            rospy.sleep(0.1)
        simulation_pub.publish(True)
        rospy.loginfo("Simulation started by node")

    def compute_pid(self,setpoint):
        """Calculate PID value for given reference feedback."""
        if (self.steer_flag != 1):
            self.integral = 0.0
            self.last_error = 0.0
```

```
        current_time = rospy.get_time()
        dt = current_time - self.last_time if self.last_time != 0 else 0.1
        self.error = setpoint - self.current_distance
        self.integral += self.error * dt
        self.derivative = (self.error - self.last_error) / dt if dt > 0 else 0

        self.output = self.Kp * self.error + self.Ki * self.integral + self.Kd * self.derivative
        self.last_error = self.error
        self.last_time = current_time
        self.steer_flag = 1
        return self.output
    def compute_steer_pid(self,setpoint):
        if (self.steer_flag != 1):
            self.integral = 0.0
            self.last_error = 0.0
        current_time = rospy.get_time()
        dt = current_time - self.last_time if self.last_time != 0 else 0.1
        self.error = abs(setpoint - self.current_lateral_position)
        self.integral += self.error * dt
        self.derivative = (self.error - self.last_error) / dt if dt > 0 else 0

        self.output = self.Kp * self.error + self.Ki * self.integral + self.Kd * self.derivative
        self.last_error = self.error
        self.last_time = current_time
        self.steer_flag =1
        return self.output

    def odom_callback(self, arr):
        """Updates the current distance based on odometry data."""
        self.current_distance = arr.data[1]
        self.current_lateral_position = arr.data[0]
        self.current_yaw = arr.data[2]

    def run(self,setpoint):
        """Main method to control the car using PID."""
        self.start_simulation()
        rate = rospy.Rate(10)  # 10 Hz
        while not rospy.is_shutdown():
            velocity = self.compute_pid(setpoint)
            print(velocity,"    ",self.current_distance)
```

```
            if velocity > 1.0:
                velocity = 1.0
                self.pub_brakes.publish(0)
            elif velocity < 0.0:
                velocity = 0.0
                self.pub_brakes.publish(self.brake_value)

            self.pub_cmd_vel.publish(velocity)
            print(velocity)
            self.pub_SteeringAngle.publish(0.0)  # Assuming straight line movement

            if self.current_distance >= setpoint:
                rospy.loginfo("Target distance reached. Stopping.")
                self.pub_cmd_vel.publish(0.0)
                self.pub_brakes.publish(self.brake_value)
                print("done")
                break
            rate.sleep()
    def steer(self):
        rate = rospy.Rate(10)  # 10 Hz
        self.start_simulation()
        flag =0
        while not rospy.is_shutdown():
            self.pub_SteeringAngle.publish(16.5)
            self.pub_cmd_vel.publish(0.2)
            print(1)
            if (self.current_lateral_position<-5):
                flag =1
            if (self.current_lateral_position < 0 and self.current_lateral_position >-1 and flag ==1 ):
                flag =0
                break
        while not rospy.is_shutdown():
            self.pub_SteeringAngle.publish(-16.5)
            self.pub_cmd_vel.publish(0.2)
            print(2)
            if (self.current_lateral_position>5):
                flag =1
            if (self.current_lateral_position < -0.5 and flag ==1 ):
                self.pub_cmd_vel.publish(0)
                self.pub_brakes.publish(self.brake_value)
```

```
                break


if __name__ == '__main__':
    try:
        controller = CarController()
        controller.steer()

    except rospy.ROSInterruptException:
        rospy.loginfo("Node was interrupted")
```

**Part 2:**

YOLO Detection:

```
import rospy
from sensor_msgs.msg import Image as ROSImage
from cv_bridge import CvBridge
import cv2
import torch
from ultralytics import YOLO
import numpy as np
from threading import Thread

# Initialize CvBridge
bridge = CvBridge()

# Set device to GPU if available
device = 'cuda' if torch.cuda.is_available() else 'cpu'

# Load the YOLO models and specify the device
model = YOLO("/home/kareem/catkin_ws/src/t2/scripts/yolov8n-oiv7.pt").to(device)
model1 = YOLO("/home/kareem/catkin_ws/src/t3/scripts/Cone.pt").to(device)
model2 = YOLO("/home/kareem/catkin_ws/src/t3/scripts/yolov8n.pt").to(device)
def get_centroid(x_min, y_min, x_max, y_max):
    center_x = (x_min + x_max) // 2
    center_y = (y_min + y_max) // 2
    return center_x, center_y

def detect_objects(img):
    # Resize image
    resized_img = cv2.resize(img, (640, 640))

    # Convert the image to the correct format and device
    img_tensor = torch.from_numpy(resized_img).permute(2, 0, 1).unsqueeze(0).float().to(device)
    img_tensor /= 255.0

    results = model.predict(img_tensor)
    results1 = model1.predict(img_tensor)
    results2 = model2.predict(img_tensor)
    for r in results:
        for box in r.boxes:
            label_text = model.names[int(box.cls)]  # Get the class name directly from the model's names
            confidence = box.conf.item()  # Extract the scalar confidence value
            if label_text in ["Car"]:
                x1, y1, x2, y2 = map(int, box.xyxy[0])
                # Format the label to include confidence score
                center_x, center_y = get_centroid(x1, y1, x2, y2)
                label_with_conf = f"{label_text} {confidence:.2f}"
                # Draw the bounding box and label on the resized_img directly
                cv2.rectangle(resized_img, (x1, y1), (x2, y2), (255, 0, 255), 3)
                cv2.putText(resized_img, label_with_conf, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)
                cv2.circle(resized_img, (center_x, center_y), 5, (0, 255, 0), -1)

```

```
    #Initiate second model detection
    for r in results1:
        for box in r.boxes:
            label_text = model1.names[int(box.cls)]
            confidence = box.conf.item()
            x1, y1, x2, y2 = map(int, box.xyxy[0])

            center_x, center_y = get_centroid(x1, y1, x2, y2)
            label_with_conf = f"cone {confidence:.2f}"

            cv2.rectangle(resized_img, (x1, y1), (x2, y2), (255, 0, 255), 3)
            cv2.putText(resized_img, label_with_conf, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)
            cv2.circle(resized_img, (center_x, center_y), 5, (0, 255, 0), -1)
    #Initiate third model detection
    for r in results2:
        for box in r.boxes:
            label_text = model2.names[int(box.cls)]
            confidence = box.conf.item()
            if label_text in ["person"]:
                x1, y1, x2, y2 = map(int, box.xyxy[0])

                center_x, center_y = get_centroid(x1, y1, x2, y2)
                label_with_conf = f"person {confidence:.2f}"

                cv2.rectangle(resized_img, (x1, y1), (x2, y2), (255, 0, 255), 3)
                cv2.putText(resized_img, label_with_conf, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)
                cv2.circle(resized_img, (center_x, center_y), 5, (0, 255, 0), -1)
    return resized_img

def image_callback(msg):
    try:
        # Convert ROS Image message to OpenCV image
        cv_image = bridge.imgmsg_to_cv2(msg, desired_encoding="passthrough")
        cv_image = cv2.cvtColor(cv_image, cv2.COLOR_RGB2BGR)
        # Detect objects in the image
        detected_image = detect_objects(cv_image)

        # Display the image with detections
        cv2.imshow("YOLO Object Detection", detected_image)
        cv2.waitKey(1)

    except Exception as e:
        rospy.logerr(e)

def main():
    # Initialize ROS node
    rospy.init_node('yolo_image_detection_node')

    # Subscribe to the image topic
    image_subscriber = rospy.Subscriber('image', ROSImage, image_callback)

    # Spin
    rospy.spin()

if __name__ == '__main__':
    main()
```

The models that we used in the next drive:

https://drive.google.com/drive/folders/1O3j3UatkVDyrIq9mdwpL3UICw0olV9pD?usp=sharing

Submit a clear screenshot of your code.

**Conclusion**

**Part 1:**
In this part we suffered to understand Kalman filter and its parameter and how to apply it with car
We learn how to Use MATLAB to publish and subscribe in ROS and deal with MathWorks documentation and use Kalman filter in the MATLAB and tunning it to our parameter.
Using PID and tunning it and overcome a slipping that appear in our tunning.

**Part 2:**
The challenge was to try to train on our machine and this took around 5 days to do that.
Second to understand the image processing algorithm and how to use its function in the right position.
Change between models required to convert labels file and codes to match it with coppilsim.