Fachhochschule
Dortmund
University of Applied Sciences and Arts

Signals & Systems for Automated Driving

Semester Project

Aditya Kumar
Igor Risteski
Md. Abul Khair

Outline

- Type of sensors and associated noise
- Types of Filters
- Developed simulation environment for the project
- ADAS Function: ACC and CAS combined
- ADAS Function: Lane Merging

Types of Sensors and Noise

- **Radar: Distance, Radial Speed, Angle**

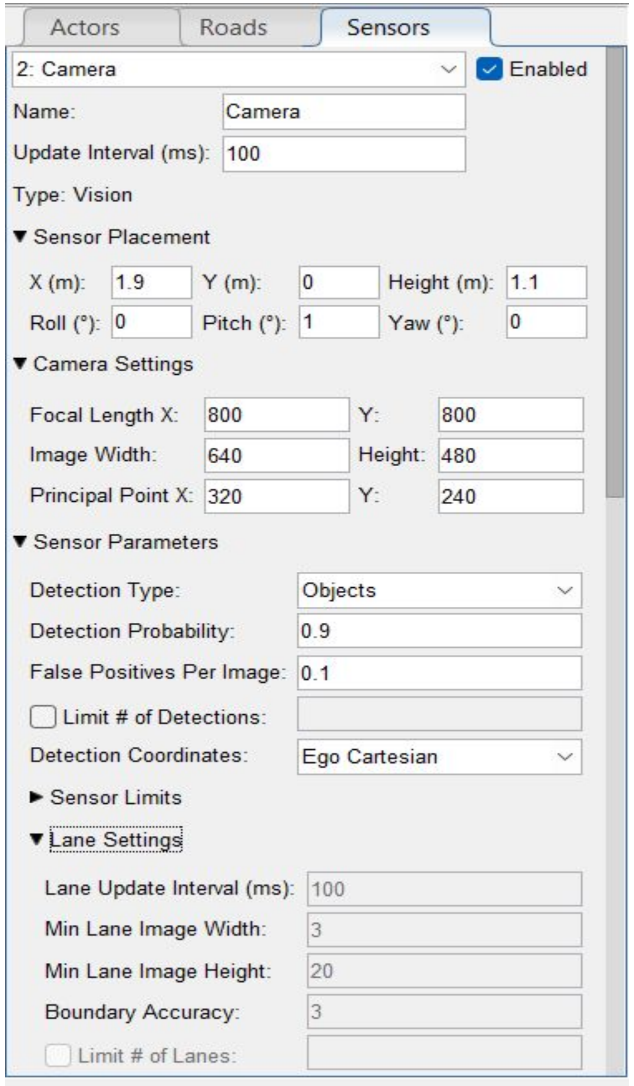
  Noise: Motion, Interference, clutter

- **Lider: Distance, Multi Angle, scanning**

  Noise: Environment(sunlight), reflectivity

- **Camera: Projection, lane tracking, Distance**

  Noise: Environment(Dark), Compression, Lens Flares



Types of filters

- Batch Expression: Computationally expensive
- Average Filter: For static signal
- Moving Average Filter: Equal weighting
- Low Pass filter: Static Weighting
- Kalman Filter: Dynamic

The Kalman Filter

- **Step 0: Initialization**
  - Initial System State ($\hat{x}_{0,0}$)
  - Initial State Variance ($p_{0,0}$)

The initialization is followed by prediction.

- **Step 1: Measurement**
  - Measured System State (z_n)
  - Measurement Variance (r_n)

- **Step 2: State Update**

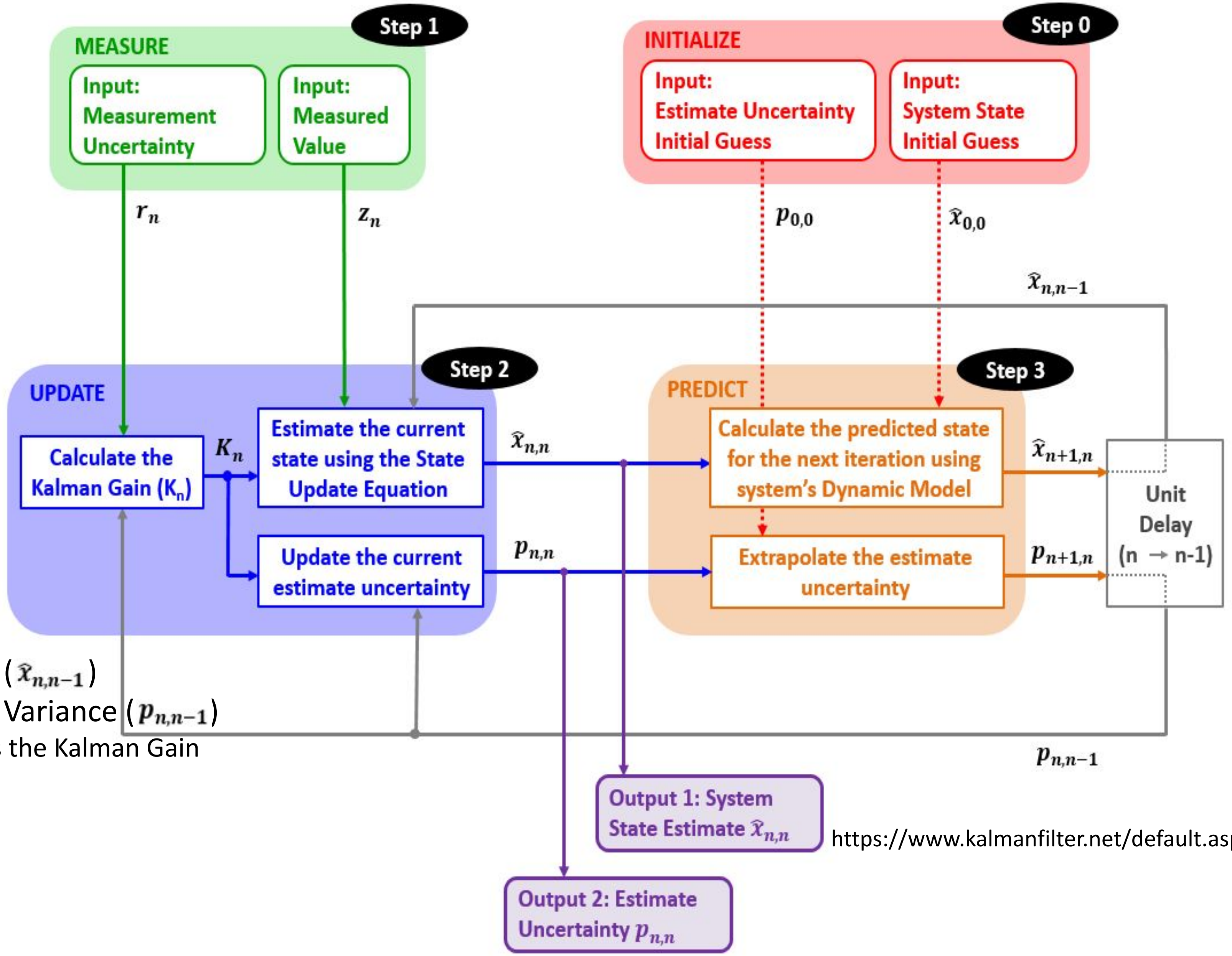
The state update process is responsible for the state estimation of the current state of the system.
The state update process inputs are:

- Measured Value (z_n)
- A Measurement Variance (r_n)
- A prior Predicted System State Estimate ($\hat{x}_{n,n-1}$)
- A prior Predicted System State Estimate Variance ($p_{n,n-1}$)

Based on the inputs, the state update process calculates the Kalman Gain and provides 2 outputs:

- Current System State Estimate ($\hat{x}_{n,n}$)
- Current State Estimate Variance ($p_{n,n}$)

These parameters are the Kalman Filter outputs.



https://www.kalmanfilter.net/default.aspx

- **Step 3: Prediction**

The prediction process anticipates the current system state estimate and its variance to the next system state based on the dynamic model of the system.
At the first iteration, the initialization is treated as the Prior State Estimate and Variance.
The Prediction outputs are used as the Prior (predicted) State Estimate and Variance in the following filter iterations.

The Kalman Filter

Kalman Gain

The Kalman Gain is a number between $0 \leq K_n \leq 1$, that defines the measurement weight and the prior estimate weight, when forming a new estimate.

State update equation:

$$\hat{x}_{n,n} = \hat{x}_{n,n-1} + K_n \left(z_n - \hat{x}_{n,n-1}\right) = \left(1 - K_n\right) \hat{x}_{n,n-1} + K_n z_n$$

where,
(K_n) is the measurement weight, and
($1 - K_n$) term is the weight of the current state estimate.

If the measurement uncertainty is high, and the estimate uncertainty is low, the Kalman Gain is close to 0. This means that significant weight is given to the estimate, and small to the uncertainty.
If the opposite is the case, then the Kalman Gain is close to 1.
If both the measurement and estimate are equally uncertain, the Kalman Gain equals 0.5.

Process Noise

There are often uncertainties in the system dynamic model. These uncertainties, are called Process Noise.
The Process Noise is denoted by the variable $\boldsymbol{\omega_n}$, and the Process Noise Variance is denoted by the letter **q**.
This provokes changes in the covariance extrapolation equation.

Without noise:

$$p_{n+1,n} = p_{n,n}$$

(For constant dynamics)

$$p^x_{n+1,n} = p^x_{n,n} + \Delta t^2 \cdot p^v_{n,n}$$
$$p^v_{n+1,n} = p^v_{n,n}$$

(For constant velocity dynamics)

With noise:

$$p_{n+1,n} = p_{n,n} + q_n$$

(For constant dynamics)

$$p^x_{n+1,n} = p^x_{n,n} + \Delta t^2 \cdot p^v_{n,n}$$
$$p^v_{n+1,n} = p^v_{n,n} + q_n$$

(For constant velocity dynamics)

The Kalman Filter

In our Matlab implementation, the Kalman filter for a given scenario is defined in the following way:

```
functions = utility_functions;
[allData, ~, ~] = Sonnenstrasse_sim();
og_trajectory = functions.get_trajectory(allData, 2);
og_measures = functions.get_aggregated_measures(allData, 2);

% Remove rows with NaN values from og_measures
og_measures = og_measures(~any(isnan(og_measures), 2), :);
```

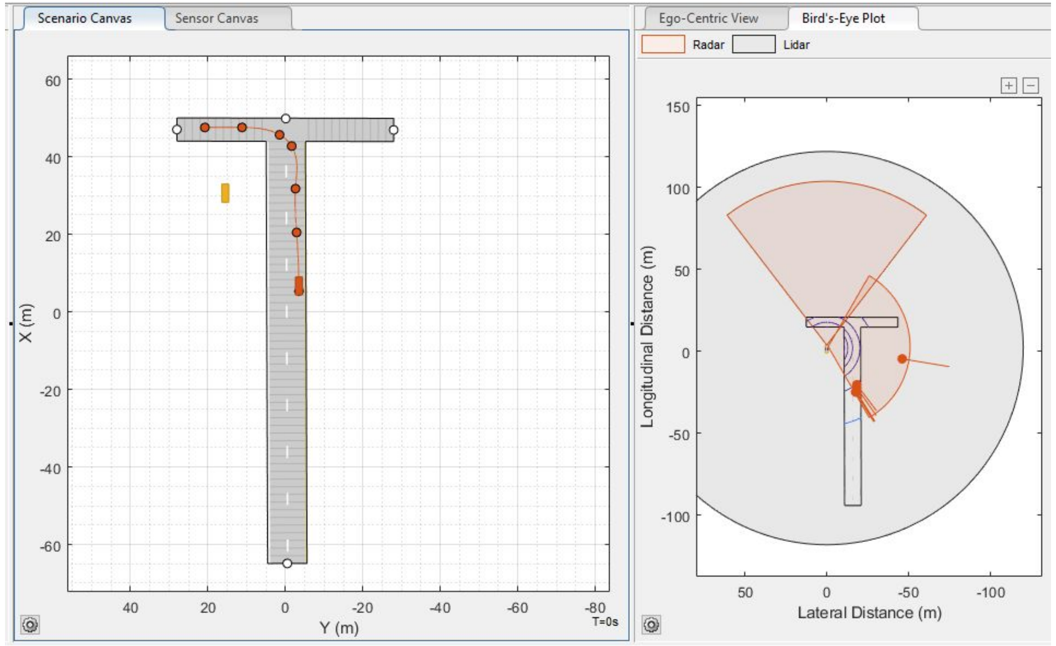
```
% Define system matrices
dt = 1; % Time step
A = [1 0 dt 0; 0 1 0 dt; 0 0 1 0; 0 0 0 1]; % State transition matrix
H = [1 0 0 0; 0 1 0 0]; % Measurement matrix
Q = eye(4); % Process noise covariance matrix
R = eye(2); % Measurement noise covariance matrix

% Initialize state estimate and covariance matrix
x_est = [trajectory(1, 1); trajectory(1, 2); 0; 0]; % Initial state estim
P = eye(4); % Initial covariance matrix

% Initialize variables to store estimated trajectory
estimated_trajectory = zeros(size(measures, 1), 2);

% Kalman filter loop
for i = 1:size(measures, 1)
    % Prediction update
    x_pred = A * x_est;
    P_pred = A * P * A' + Q;

    % Measurement update
    K = P_pred * H' * inv(H * P_pred * H' + R);
    z = measures(i, :)'; % Measurement vector
    x_est = x_pred + K * (z - H * x_pred);
    P = (eye(4) - K * H) * P_pred;

    % Store estimated trajectory
    estimated_trajectory(i, :) = [x_est(1), x_est(2)];
end
```

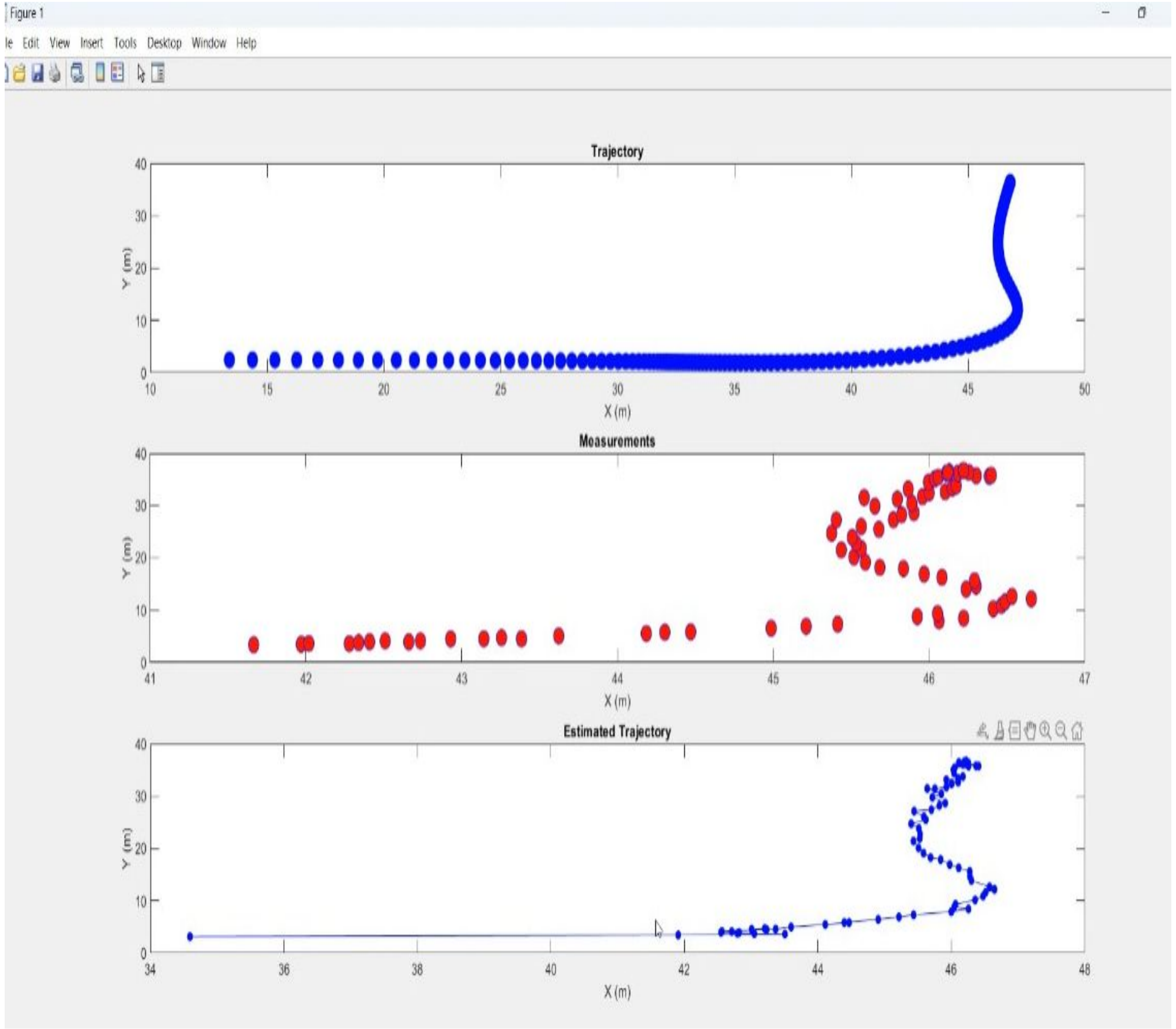
- Initially we collect data from the simulation environment using two different functions: one for the trajectories and one for measurements.

- The time step and matrices are then defined, the predicted trajectory is recorded in variables, and the Kalman filter is applied in a loop to all subsequent measurement data.

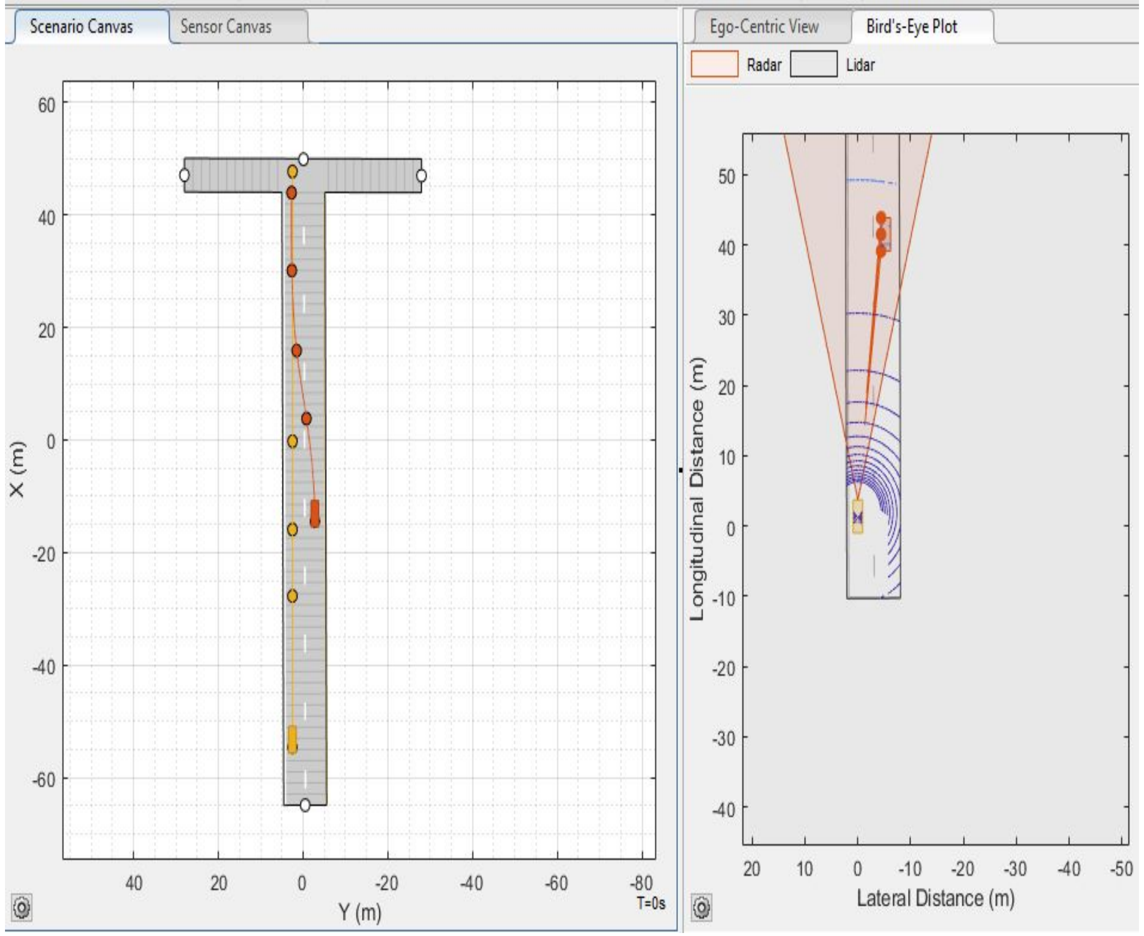
Kalman Filter for Simulated Radar

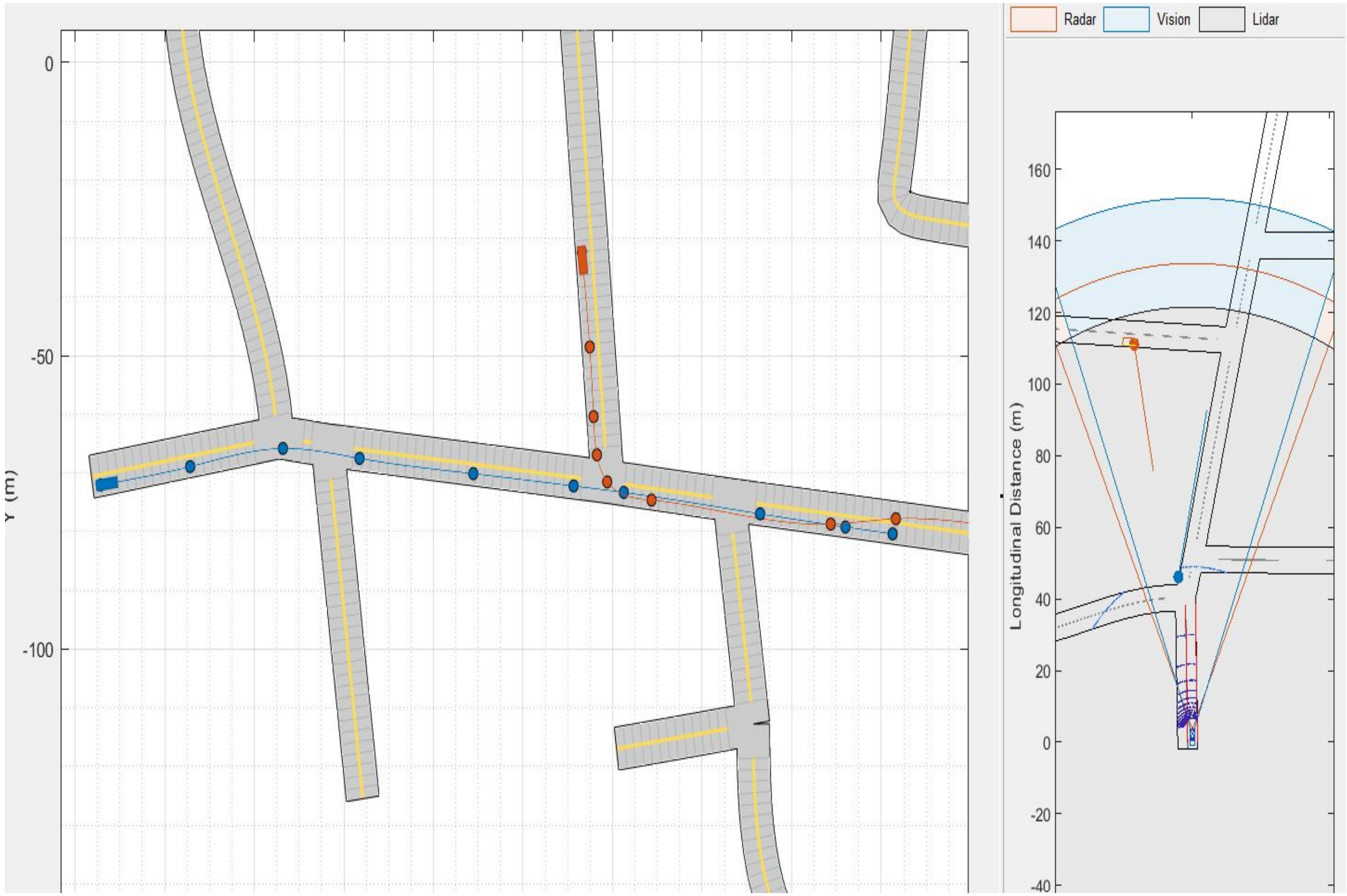
After setting up the utility functions and making the Kalman filter script, the next step is creating a scenario, in order to test it out. For this purpose, we are using the “Driving Scenario Designer” addon within Matlab. the inititial scenario consists of: creating a street and 2 vehicles, one of which is moving along the road, and the other is stationary, outside of the road, then setting up the stationary vehicle as an Ego vehicle, and putting sensors on it. This vehicle is therefore “watching” the moving vehicle, and gathers the data from the sensors, which the Kalman filter will later use and plot the trajectory.





Project Scenario Simulation

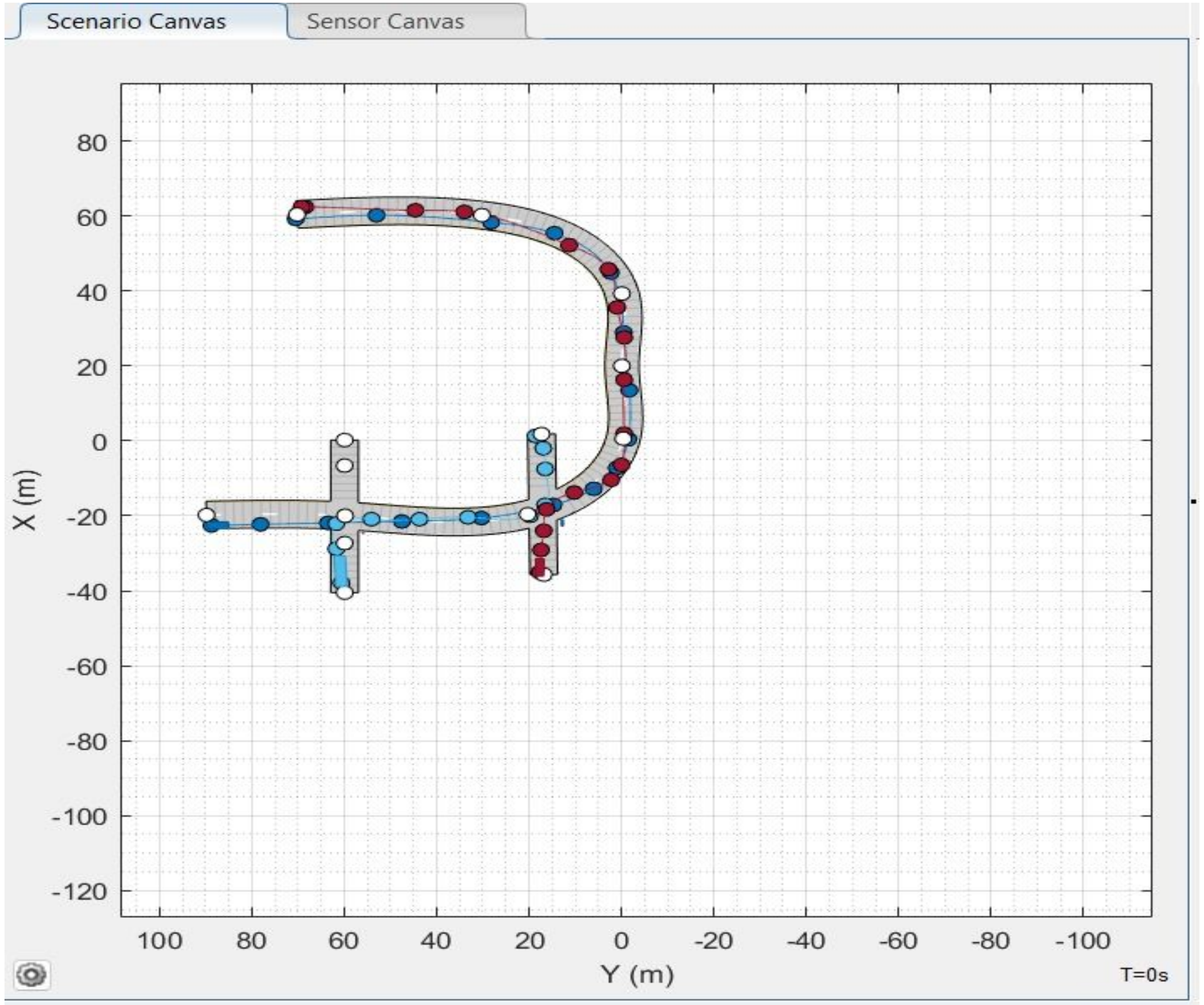


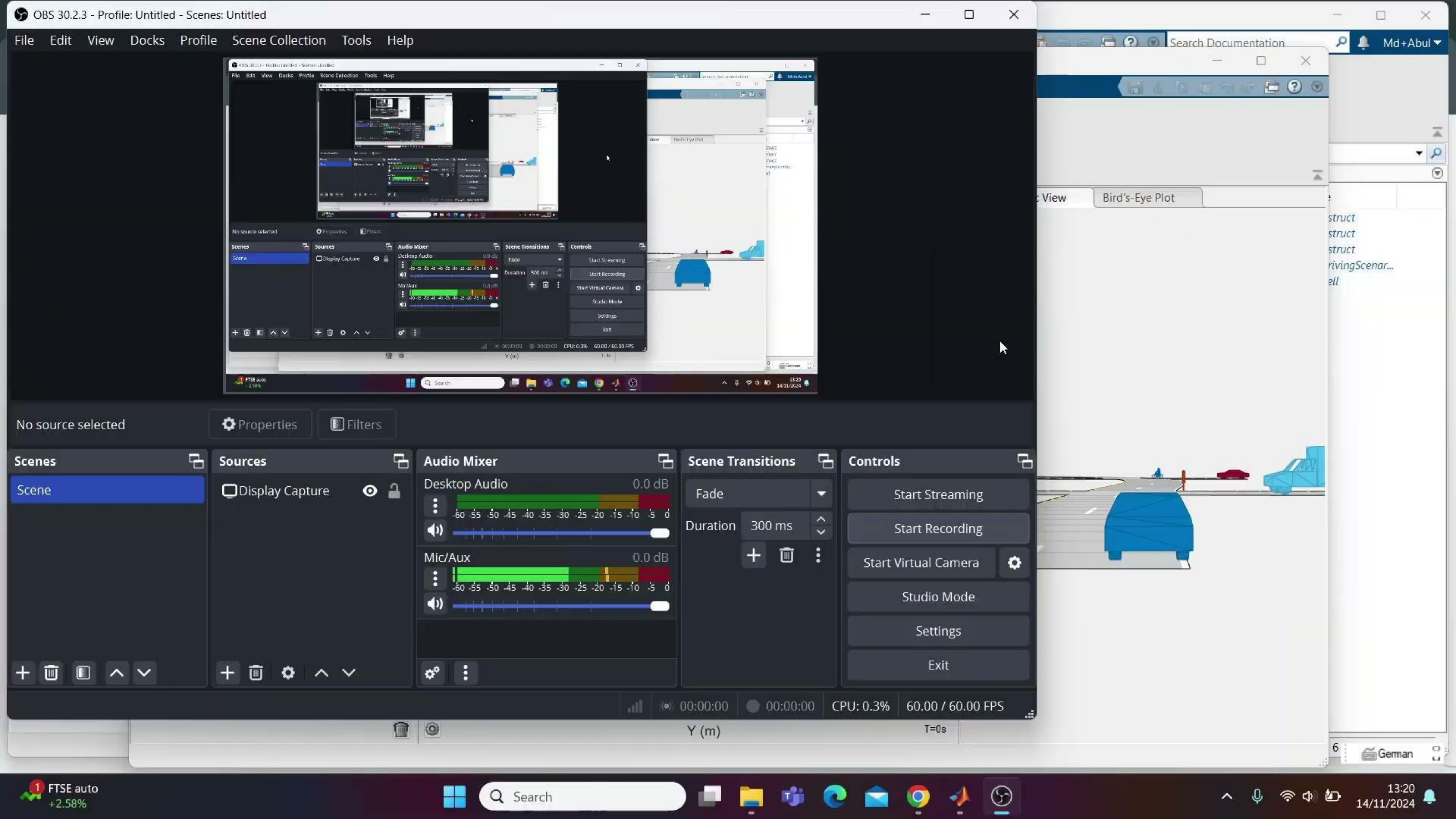


Fachhochschule Dortmund
University of Applied Sciences and Arts

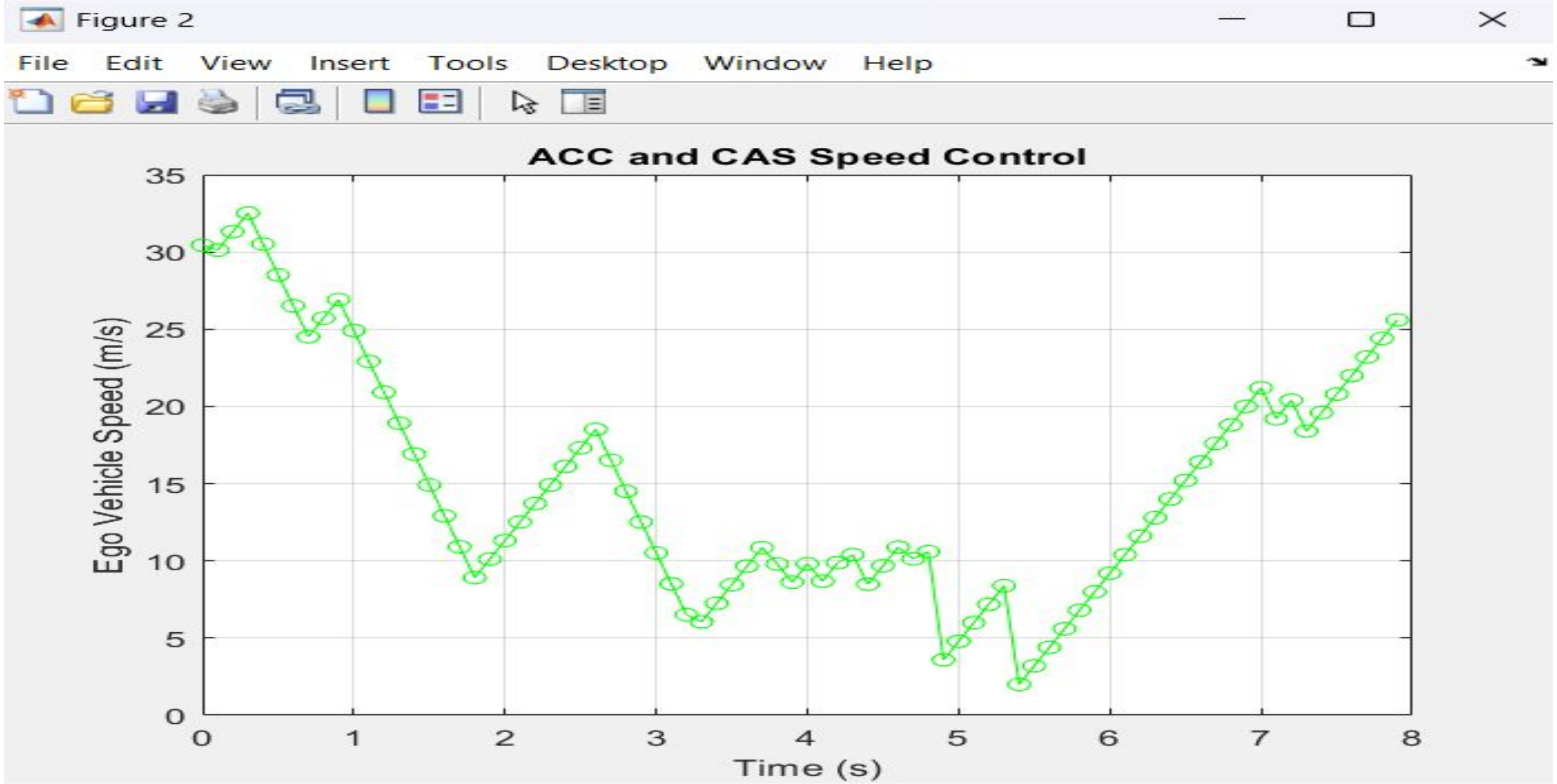
Implementing ADAS Function: ACC and CAS

While choosing which ADAS function to implement, we agreed that the simplest one would either be Lane Keeping, or Adaptive Cruise Control. We chose the ACC function combined with CAS. To implement this, we used the following scenario.

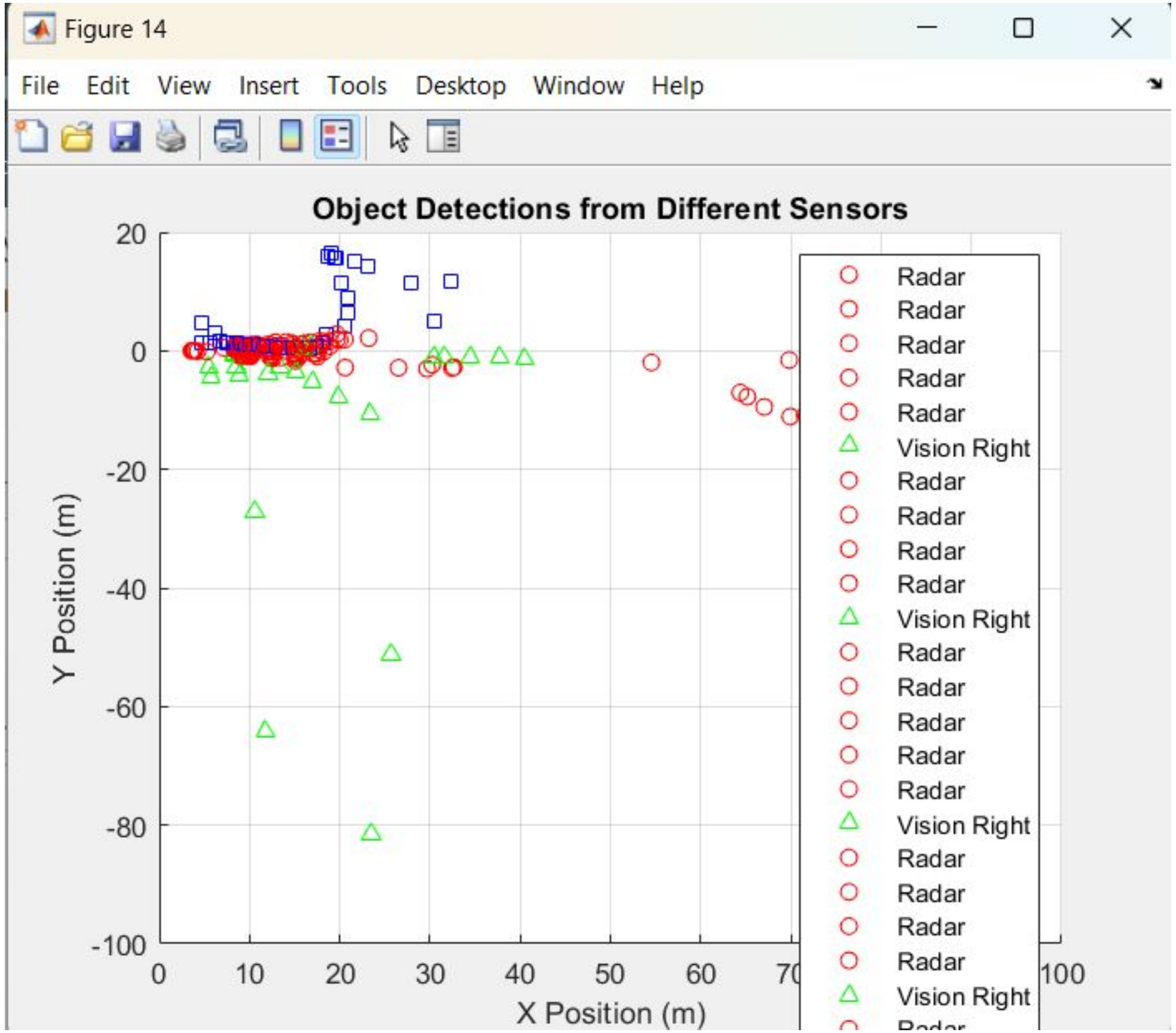


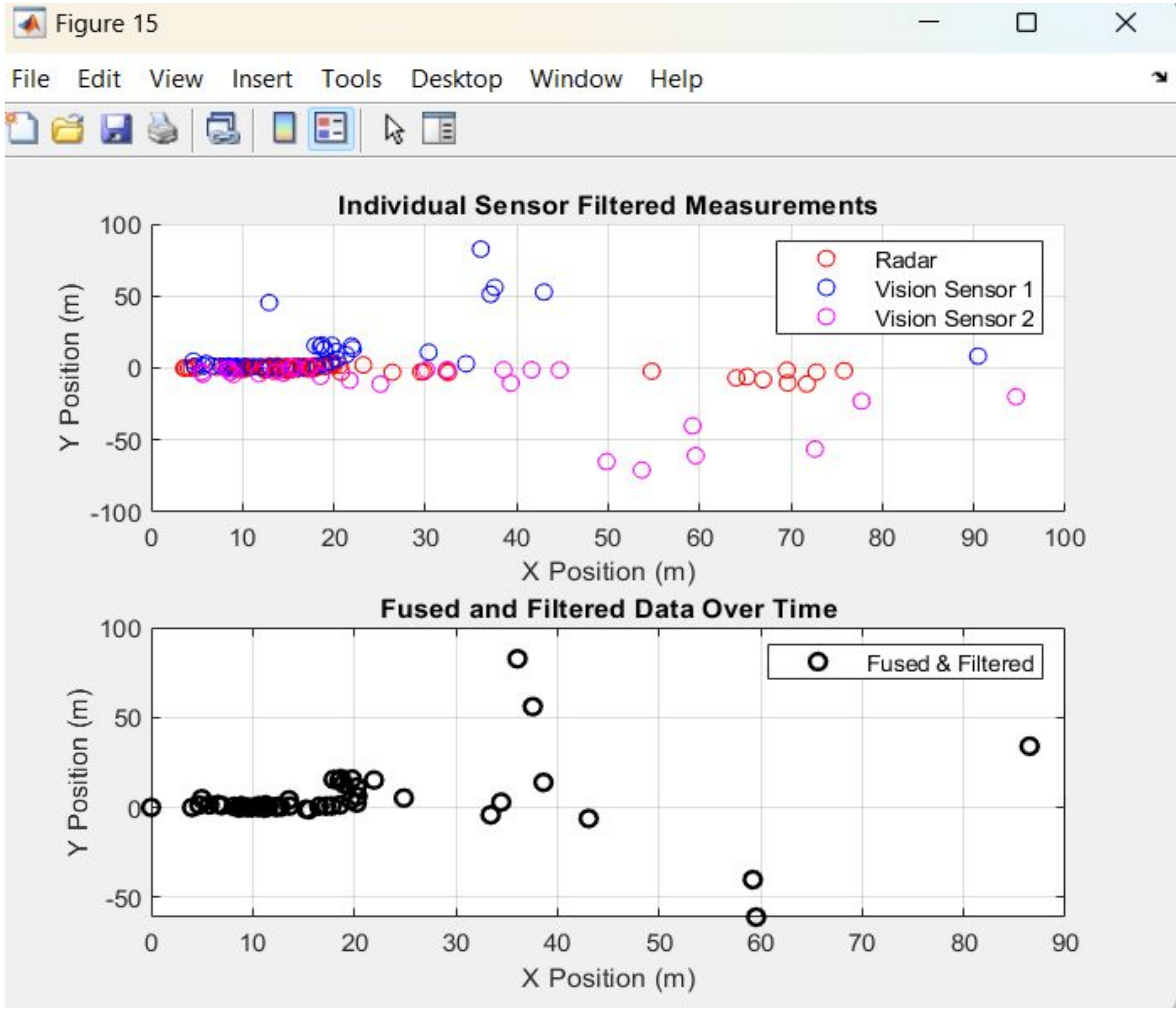

OBS 30.2.3 - Profile: Untitled - Scenes: Untitled
File
Edit
View
Docks
Profile
Scene Collection
Tools
Help
No source selected
Properties
Filters
Scenes
Scene
Sources
Display Capture
Audio Mixer
Desktop Audio
0.0 dB
Mic/Aux
0.0 dB
-60 -55 -50 -45 -40 -35 -30 -25 -20 -15 -10 -5 0
Scene Transitions
Fade
Duration
300 ms
Controls
Start Streaming
Start Recording
Start Virtual Camera
Studio Mode
Settings
Exit
00:00:00
00:00:00
CPU: 0.3%
60.00 / 60.00 FPS
Search Documentation
Md+Abul
View
Bird's-Eye Plot
struct
struct
struct
rivingScenar...
ell
Y (m)
T=0s
German
FTSE auto
+2.58%
Search
13:20
14/11/2024

Implementing ADAS Function: ACC and CAS



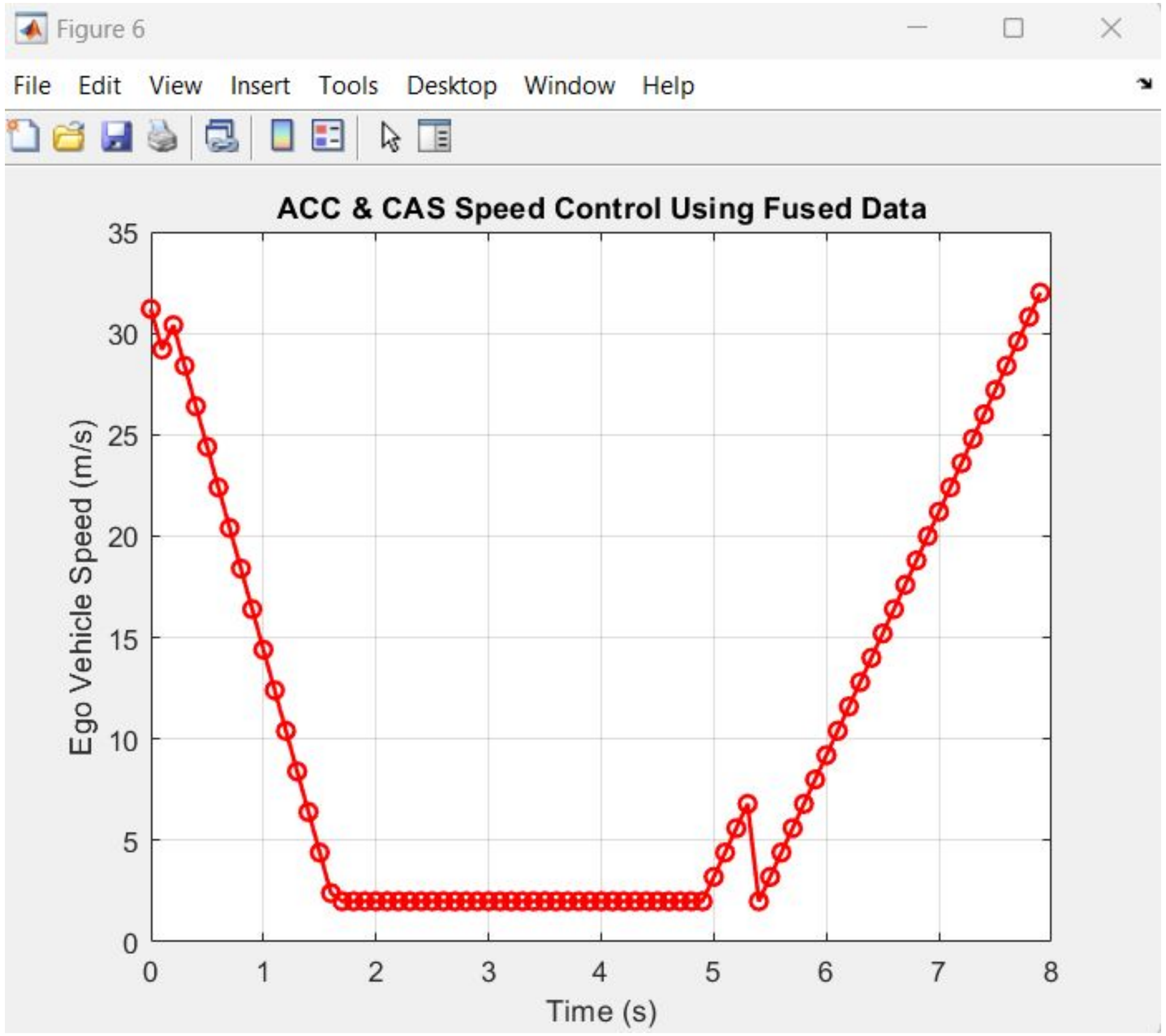
This figure shows the velocity changes over time
for raw data

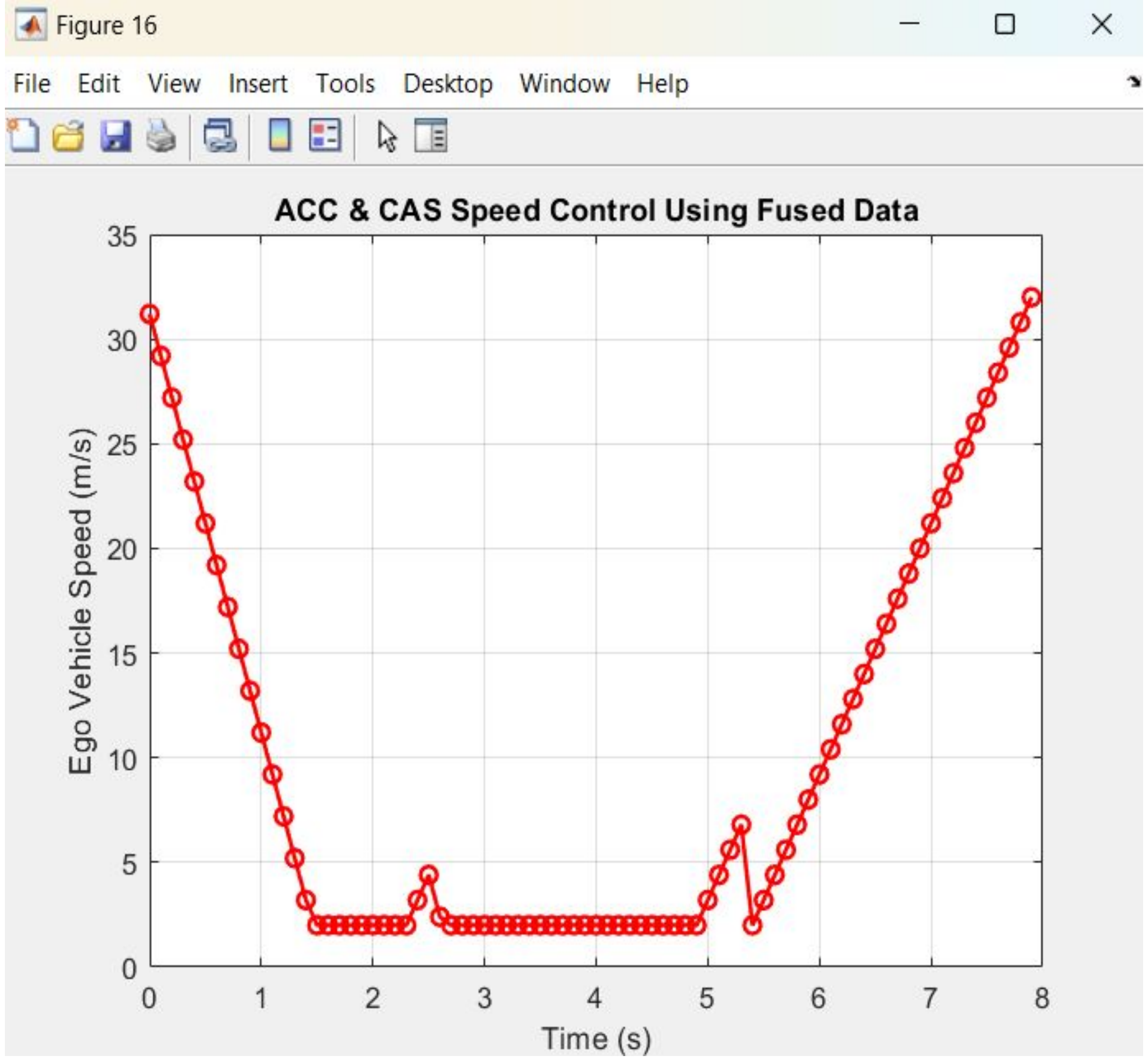




Before Filtering and Fusion

After Filtering and Fusion

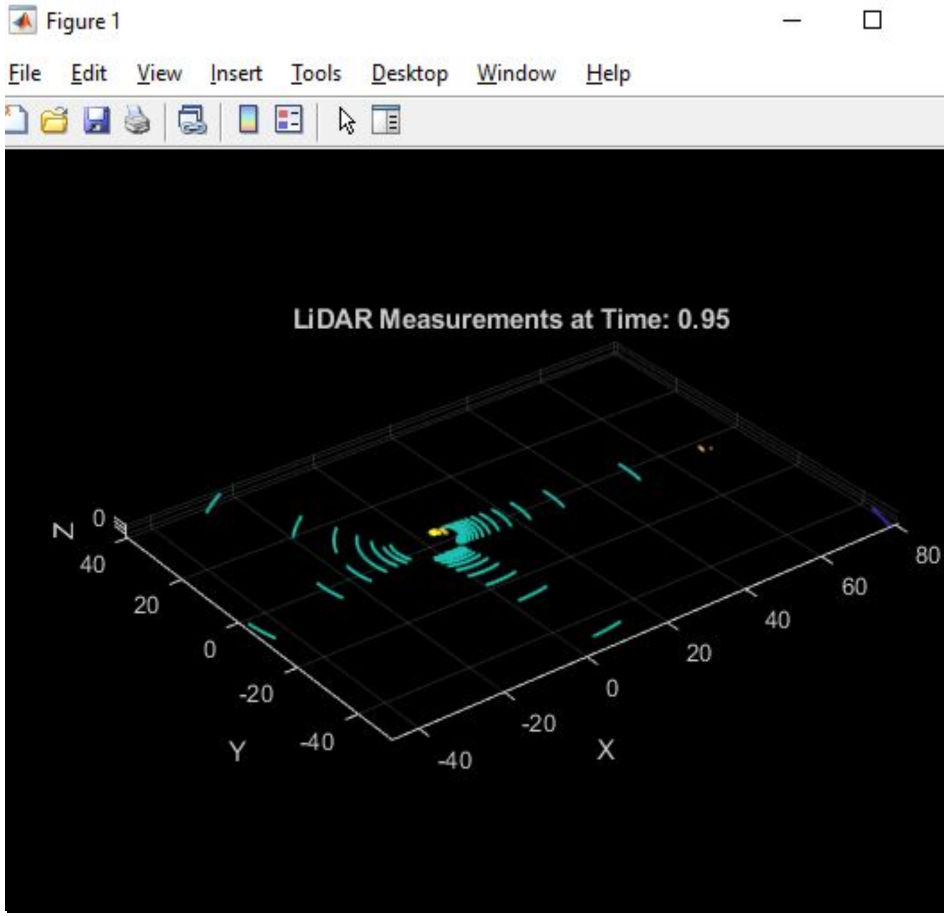


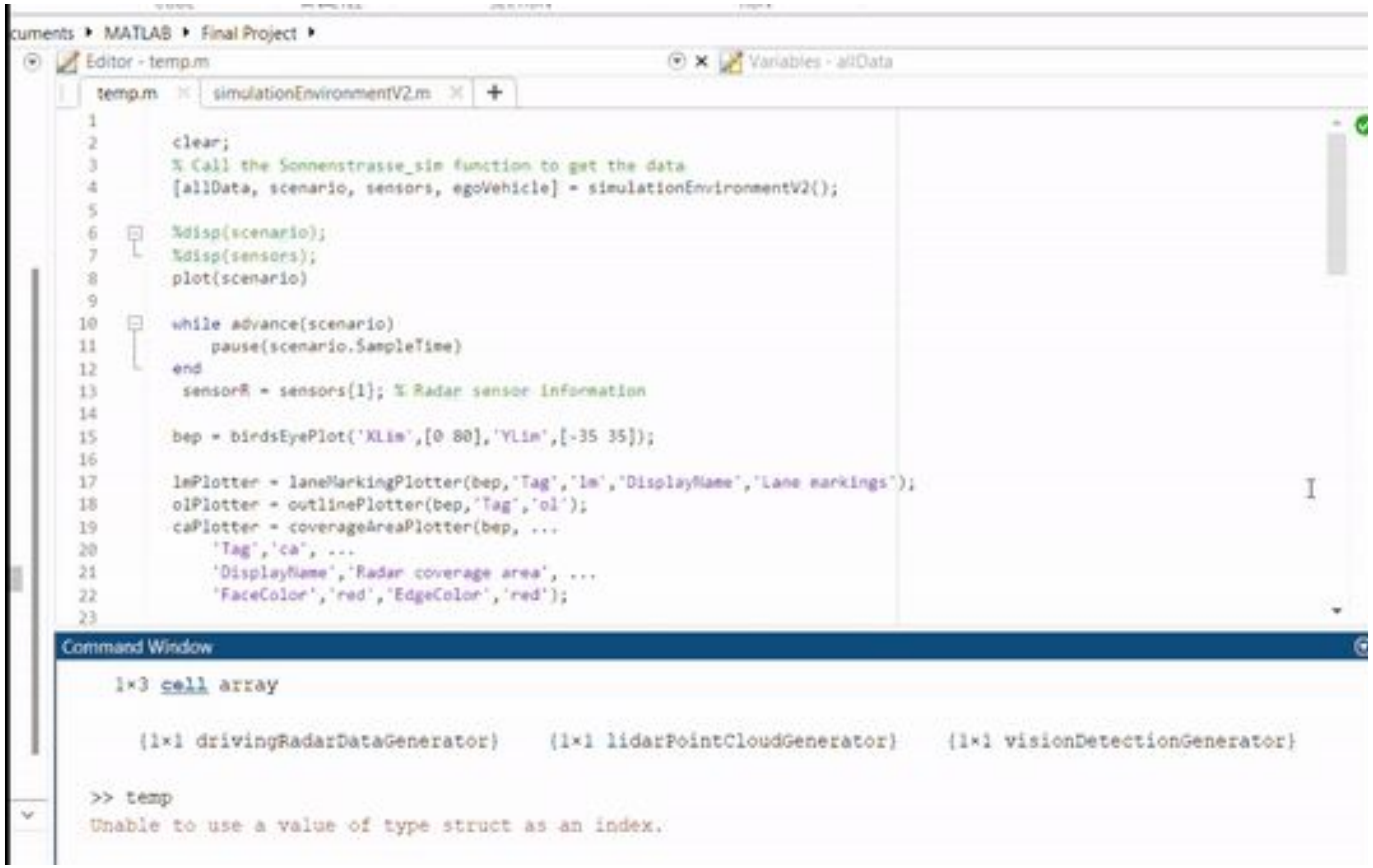


Static Weighted Average Fusion

Adaptive Weighted Average Fusion

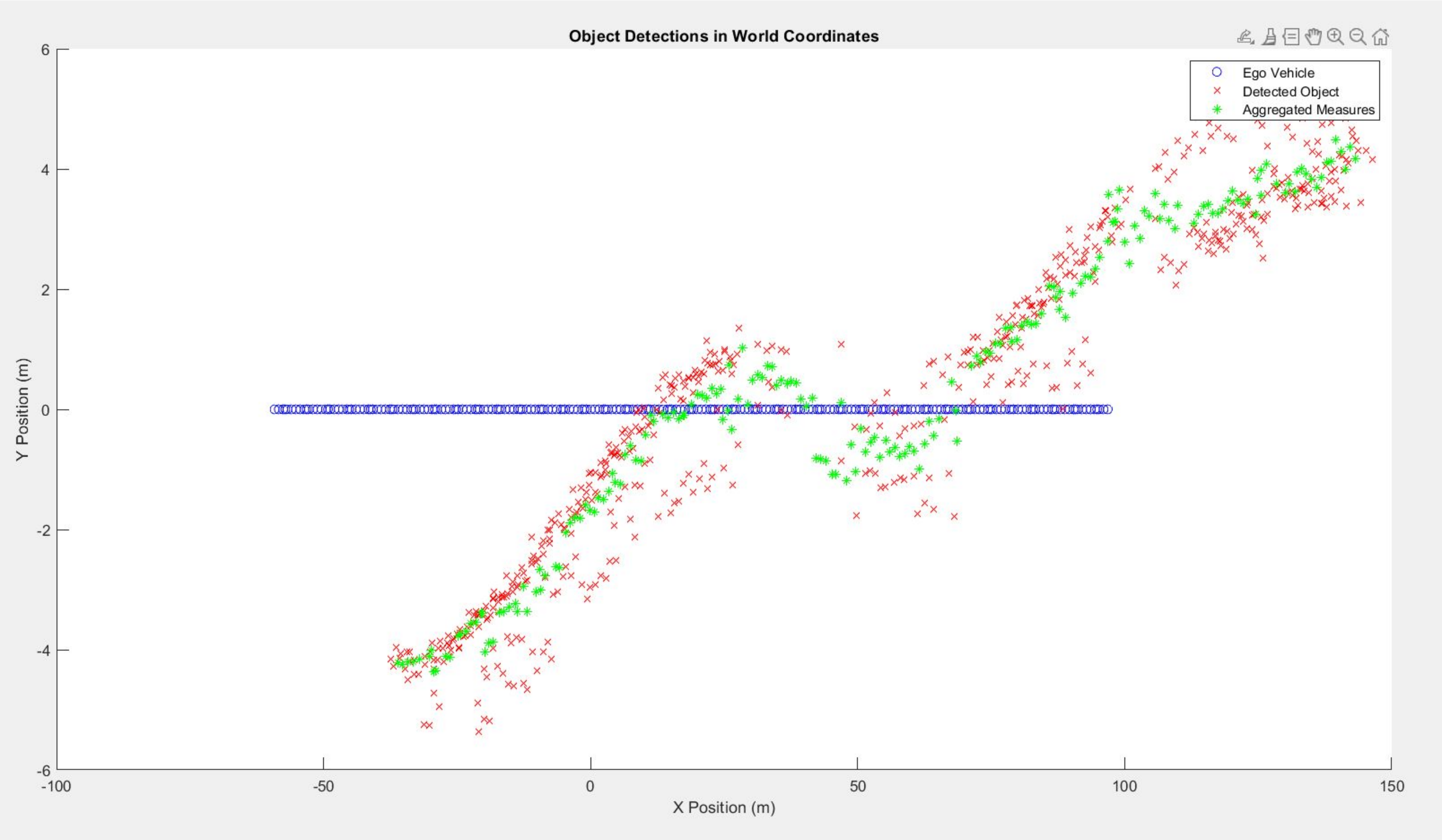
ADAS Implementation: Lane Merging



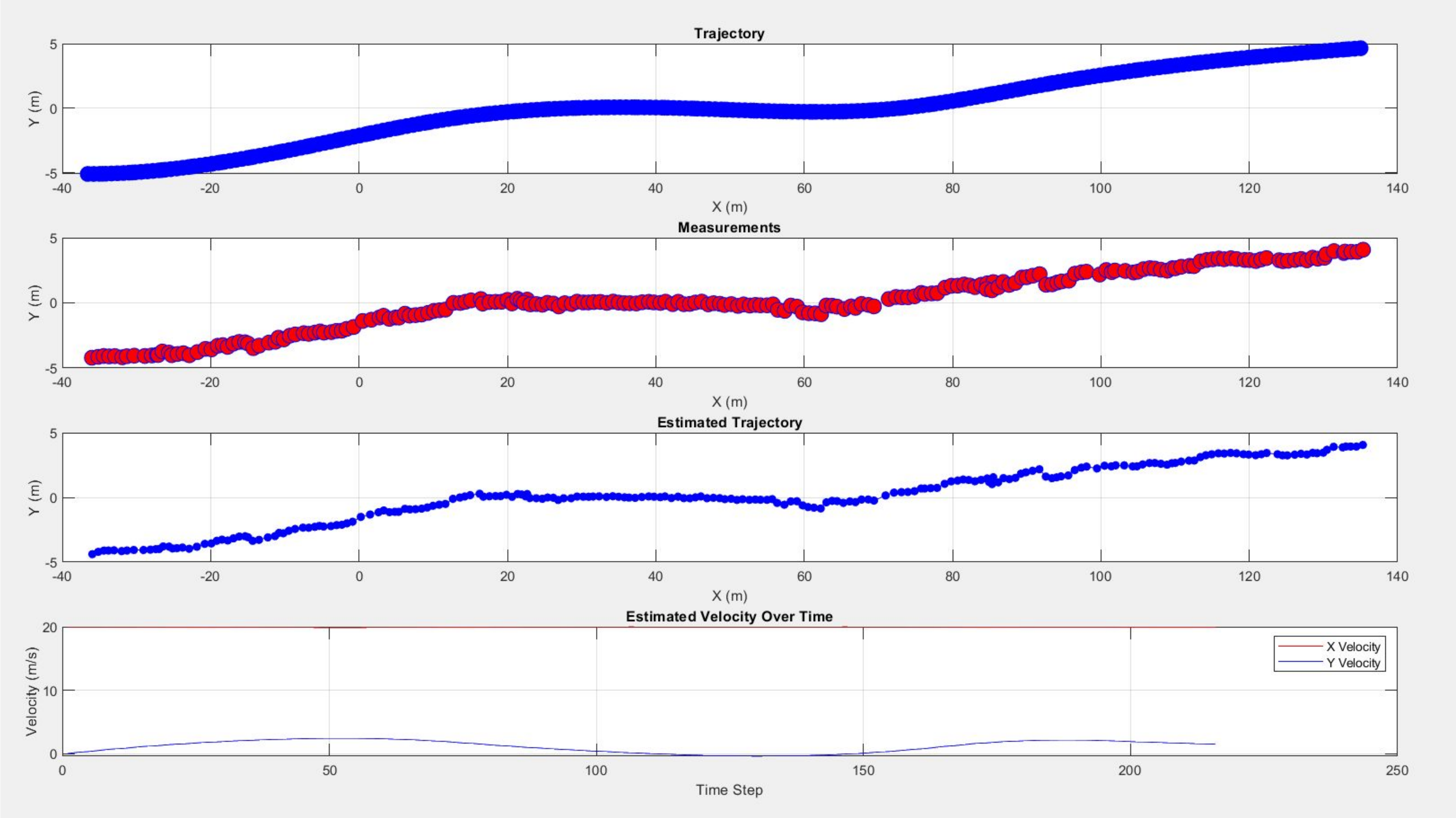


Fachhochschule Dortmund
University of Applied Sciences and Arts

ADAS Implementation: Lane Merging

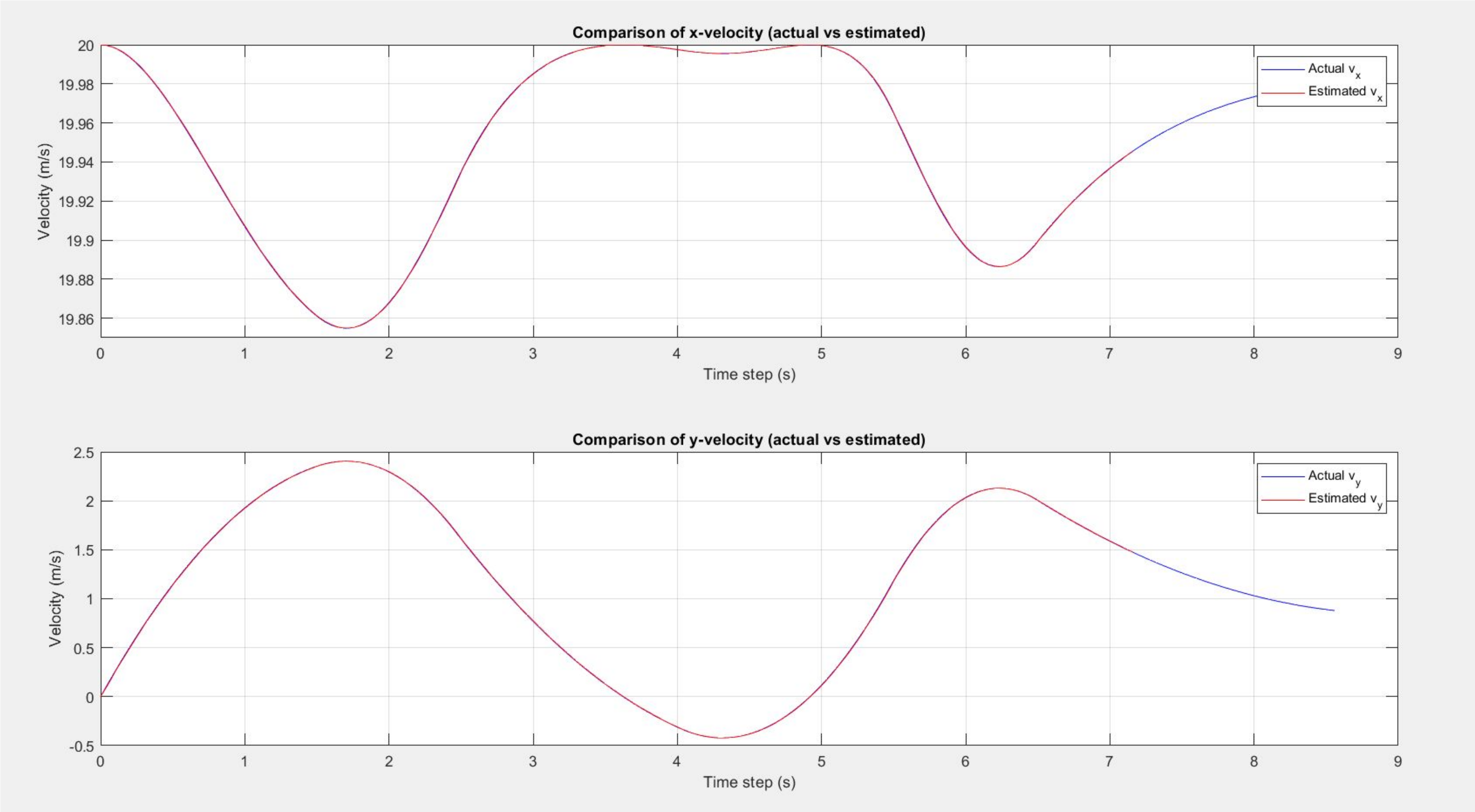


ADAS Implementation: Lane Merging



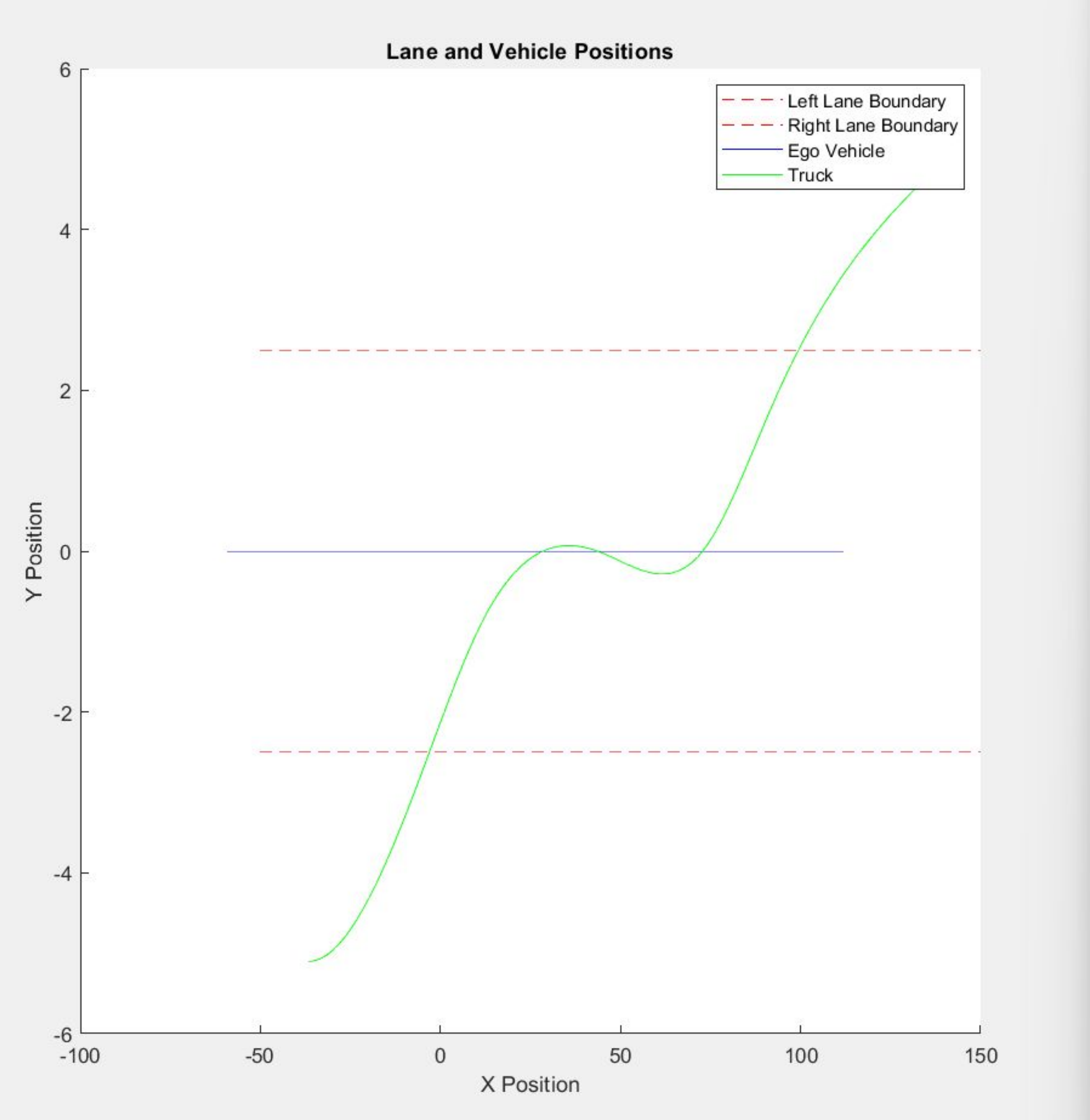
Fachhochschule Dortmund
University of Applied Sciences and Arts

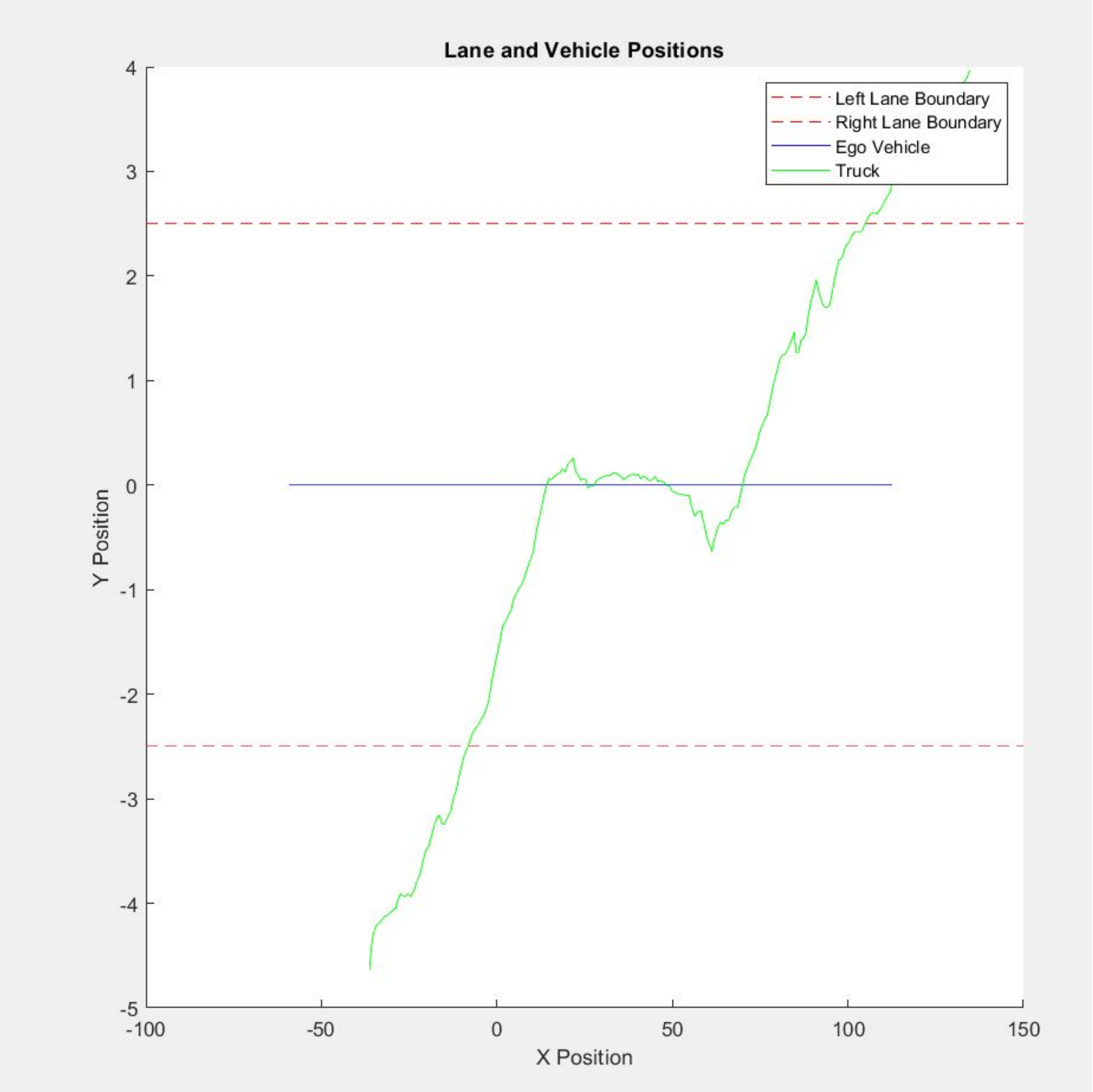
ADAS Implementation: Lane Merging



Fachhochschule Dortmund
University of Applied Sciences and Arts

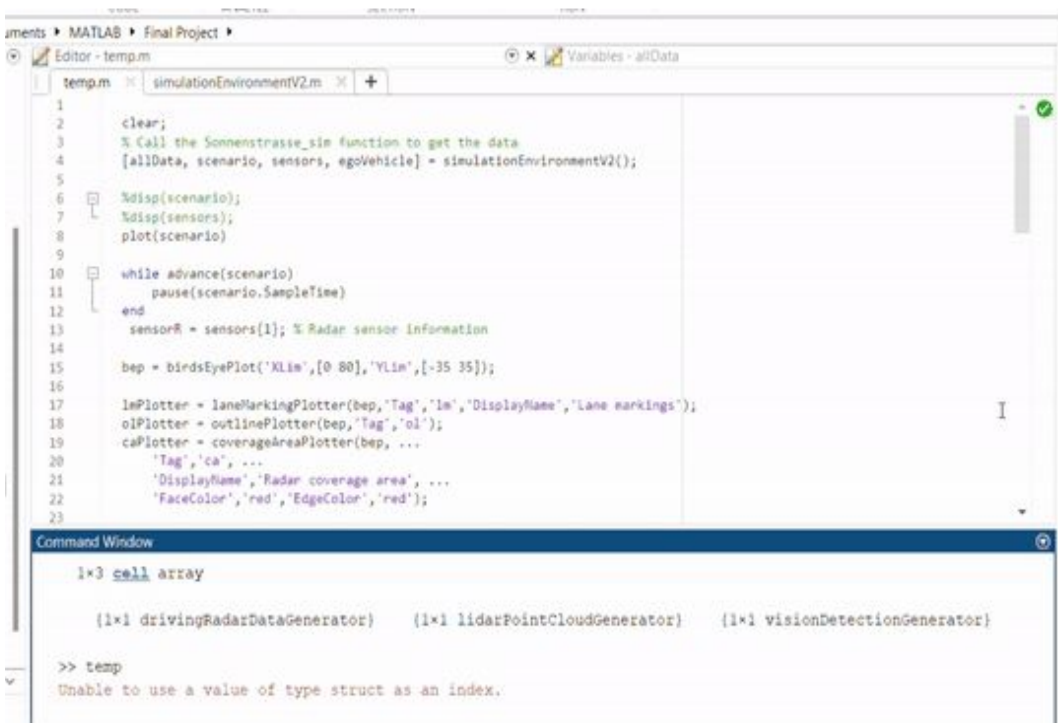
ADAS Implementation: Lane Merging





Fachhochschule Dortmund
University of Applied Sciences and Arts

ADAS Implementation: Lane Merging




```
Estimated trajectory size: 216 x 2
Truck identified in the right lane at time = 0
Warning: Truck entered the same lane as ego vehicle at time = 1.28 (Entered from right)
Warning: Truck left the ego lane to the left lane at time = 5.96
Truck identified in the left lane at time = 5.96
Estimated trajectory does not cover all time steps. Adjusting plot to show available data.
>> ADAS_LaneWarning
Truck identified in the right lane at time = 0
Warning: Truck entered the same lane as ego vehicle at time = 1.64 (Entered from right)
Warning: Truck left the ego lane to the right at time = 6.5
Truck identified in the left lane at time = 6.5
>>
```

Fachhochschule Dortmund

University of Applied Sciences and Arts

Thank you for your attention !