Configuration Manual

MSc Research Project
Cyber Security

Taher Ahmed
Student ID: 23186950

School of Computing
National College of Ireland

Supervisor: Mr. Raza Ul Mustafa

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Taher Ahmed
Student ID:	23186950
Programme:	MSc in Cyber Security
Year:	2024-25
Module:	MSc Research Project
Lecturer:	Mr. Raza Ul Mustafa
Submission Due Date:	12th December 2024
Project Title:	Detection of replay attacks in Autonomous vehicle LTV systems using Dynamic Watermarking, Kalman Filter and Mahalanobis Distance
Word Count:	1035

Signature: Taher Ahmed

Date: 12th December 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Taher Ahmed
Student ID: 23186950

1 Introduction

The research project aims to propose a robust framework in detecting replay attacks in autonomous vehicles Linear Time Varying (LTV) systems using dynamic watermarking, Kalman Filter and Mahalanobis Distance. In addition to this paper, the configuration manual provides a clear insight on setting up a simulation environment on how the CAN bus model has been developed, replay attacks are triggered, detection using dynamic watermarking with Kalman Filter and Mahalanobis Distance, and the configuration set up for evaluation metrics.

2 System configuration

2.1 System hardware configuration

- Processor: Intel core I7
- Operating System: Windows 11
- Storage: 500 GB HDD
- RAM: 32 GB

2.2 Software versions

- Python: 3.10.8
- Python-can: 4.4.2
- VS code: 1.95.1
- Matplotlib: 3.9.2
- Pandas: 2.2.3
- Filterpy: 1.4.5
- Scikit-learn: 1.5.2

3 Development environment set up

3.1 Install VS code

The entire simulation environment is developed using IDE VS code. The latest version can be downloaded from here https://code.visualstudio.com/download

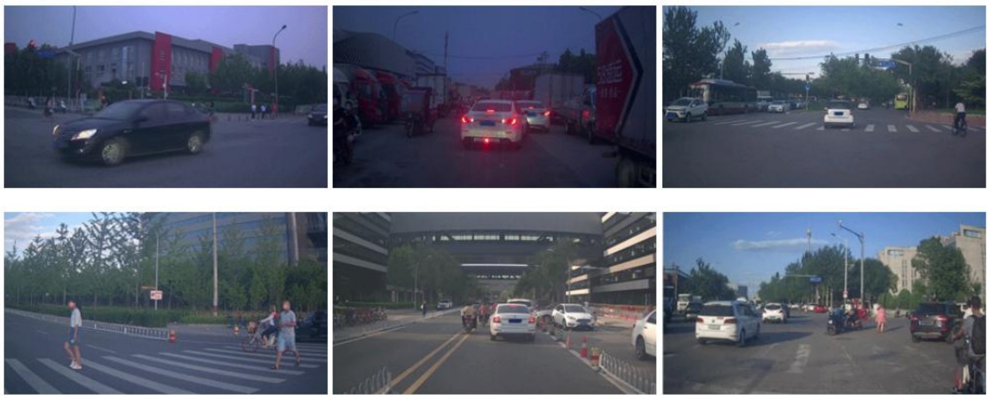
3.2 Install Python

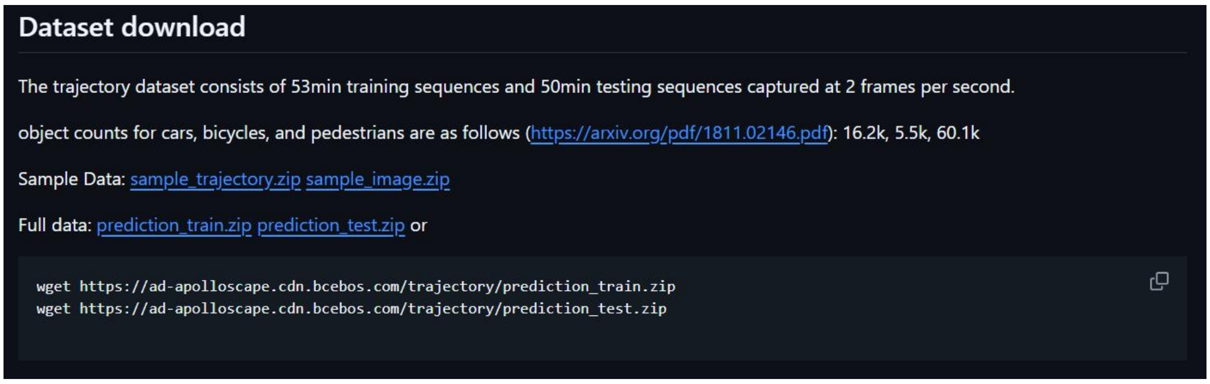
The python has been selected as a programming language to set up the simulation environment due to its high availability of libraries dedicated for a CAN bus model. The latest version can be downloaded from here https://www.python.org/downloads/

4 Dataset configuration

4.1 Apollo Scape dataset set up

Apollo Scape dataset provides a large scale trajectory dataset of urban streets consisting of camera based images, LiDAR scanned point clouds and manually annotated trajectories along with traffic flows containing vehicles, riders and pedestrians. This overall sensor data of more than 400,000 data points are used as message transfer between sensor nodes to ECU or Actuator nodes in the simulation environment to handle the dynamics of the vehicle maneuver based on detected objects around the vehicle to avoid any collision.





Dataset download

The trajectory dataset consists of 53min training sequences and 50min testing sequences captured at 2 frames per second.

object counts for cars, bicycles, and pedestrians are as follows (https://arxiv.org/pdf/1811.02146.pdf): 16.2k, 5.5k, 60.1k

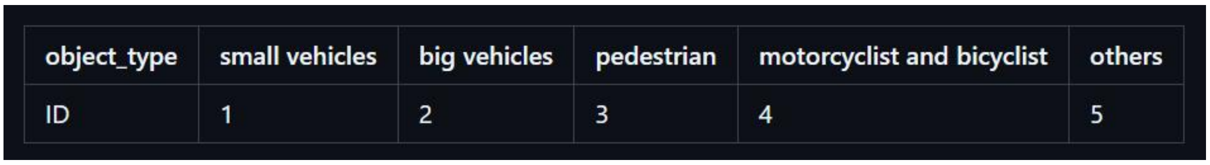
Sample Data: sample_trajectory.zip sample_image.zip

Full data: prediction_train.zip prediction_test.zip or

```
wget https://ad-apolloscape.cdn.bcebos.com/trajectory/prediction_train.zip
wget https://ad-apolloscape.cdn.bcebos.com/trajectory/prediction_test.zip
```

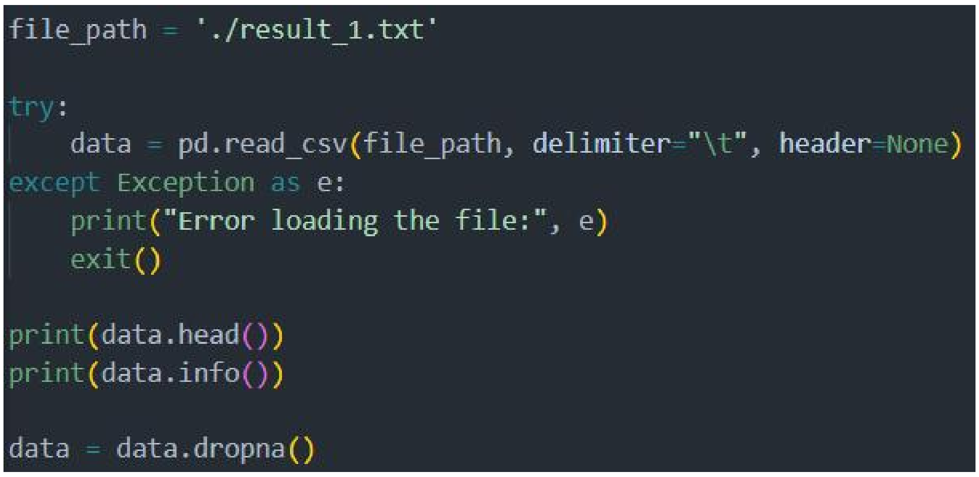
4.2 Dataset pre-processing

The dataset provides overall details of five different object types such as, small vehicles, big vehicles, pedestrian, motorcyclist and bicyclist, and other variants. This data structure is categorized with separate columns as, frame id, object id, object type, position x, position y, position z, object length, object width, object height and heading.



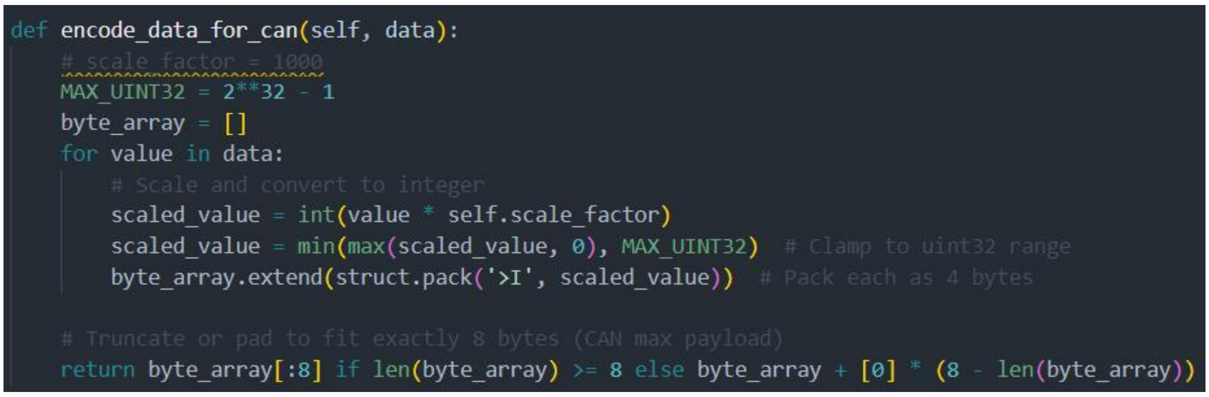
object_type	small vehicles	big vehicles	pedestrian	motorcyclist and bicyclist	others
ID	1	2	3	4	5

Considering the amount of dataset, the first step of pre-processing involved inspecting the raw data to ensure data with proper delimiter, removing null values or inappropriate values from the dataset. This ensures that the dataset has valid and appropriate data to maintain the reliability and accuracy for simulation of messages.



```python
file_path = './result_1.txt'

try:
    data = pd.read_csv(file_path, delimiter="\t", header=None)
except Exception as e:
    print("Error loading the file:", e)
    exit()

print(data.head())
print(data.info())

data = data.dropna()
```

In order to transmit the data through CAN messages, the range value exceeding 0 to 255 bytes were encoded to split each values and transmit across multiple bytes. This technique is known as Byte Splitting which allows the larger values into sequence of bytes and later be decoded to reassemble to fetch the actual data.



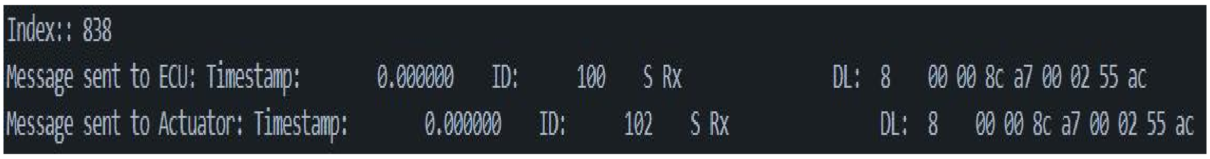
```python
def encode_data_for_can(self, data):
    # scale_factor = 1000
    MAX_UINT32 = 2**32 - 1
    byte_array = []
    for value in data:
        # Scale and convert to integer
        scaled_value = int(value * self.scale_factor)
        scaled_value = min(max(scaled_value, 0), MAX_UINT32)  # Clamp to uint32 range
        byte_array.extend(struct.pack('>I', scaled_value))  # Pack each as 4 bytes

    # Truncate or pad to fit exactly 8 bytes (CAN max payload)
    return byte_array[:8] if len(byte_array) >= 8 else byte_array + [0] * (8 - len(byte_array))
```

5 Simulation environment set up

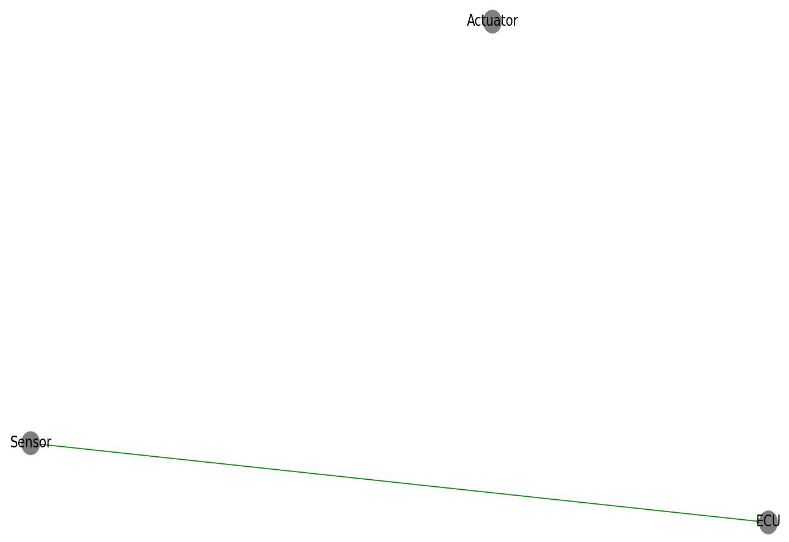
5.1 Preparation of CAN bus model

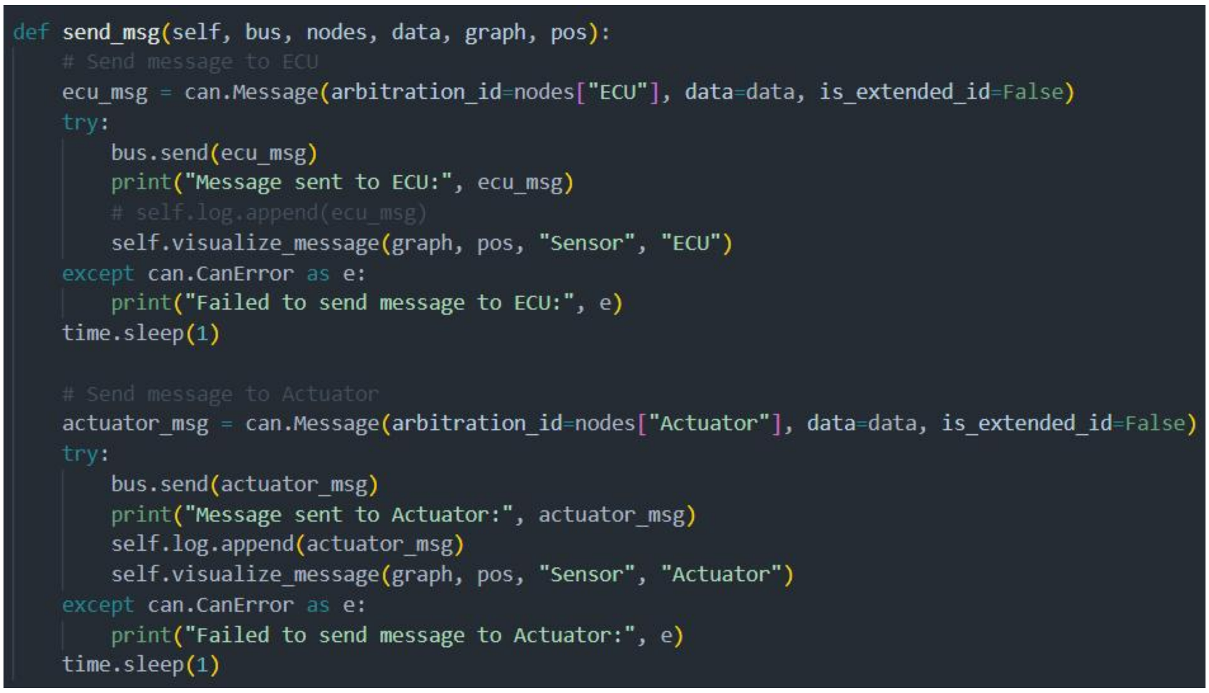
Considering the key components of LTV system in an autonomous vehicles, three nodes, ECU, Actuators and Sensors within a CAN bus are created to send and receive messages between each units. Python provides a dedicated library which acts as a CAN bus model and by default handles its characteristics and limitations. The latest version can be downloaded from here https://python-can.readthedocs.io/en/stable/

The log of a particular message transfer between sensor node to an actuator node in a CAN bus is shown below.



```
Index:: 838
Message sent to ECU: Timestamp:       0.000000    ID:      100    S Rx             DL:  8    00 00 8c a7 00 02 55 ac
Message sent to Actuator: Timestamp:       0.000000    ID:      102    S Rx             DL:  8    00 00 8c a7 00 02 55 ac
```

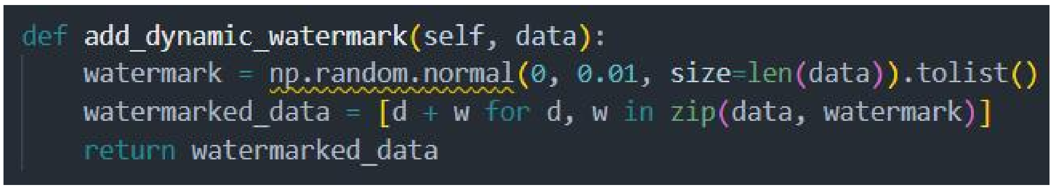




```python
def send_msg(self, bus, nodes, data, graph, pos):
    # Send message to ECU
    ecu_msg = can.Message(arbitration_id=nodes["ECU"], data=data, is_extended_id=False)
    try:
        bus.send(ecu_msg)
        print("Message sent to ECU:", ecu_msg)
        # self.log.append(ecu_msg)
        self.visualize_message(graph, pos, "Sensor", "ECU")
    except can.CanError as e:
        print("Failed to send message to ECU:", e)
    time.sleep(1)

    # Send message to Actuator
    actuator_msg = can.Message(arbitration_id=nodes["Actuator"], data=data, is_extended_id=False)
    try:
        bus.send(actuator_msg)
        print("Message sent to Actuator:", actuator_msg)
        self.log.append(actuator_msg)
        self.visualize_message(graph, pos, "Sensor", "Actuator")
    except can.CanError as e:
        print("Failed to send message to Actuator:", e)
    time.sleep(1)
```

5.2 Apply dynamic watermarking

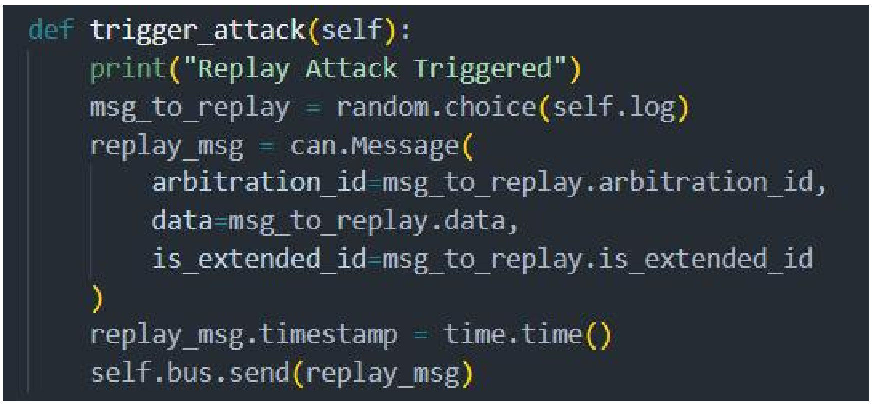
Every messages passed through the CAN bus model are handled by adding dynamic watermarking. The following python function is used for this integration.



```python
def add_dynamic_watermark(self, data):
    watermark = np.random.normal(0, 0.01, size=len(data)).tolist()
    watermarked_data = [d + w for d, w in zip(data, watermark)]
    return watermarked_data
```

5.3 Trigger replay attack

In order to simulate a replay attack within the message transmission between sensor and actuator nodes, the previous sensor messages sent into CAN bus are captured and these messages are replayed with a varying watermark at different time intervals.

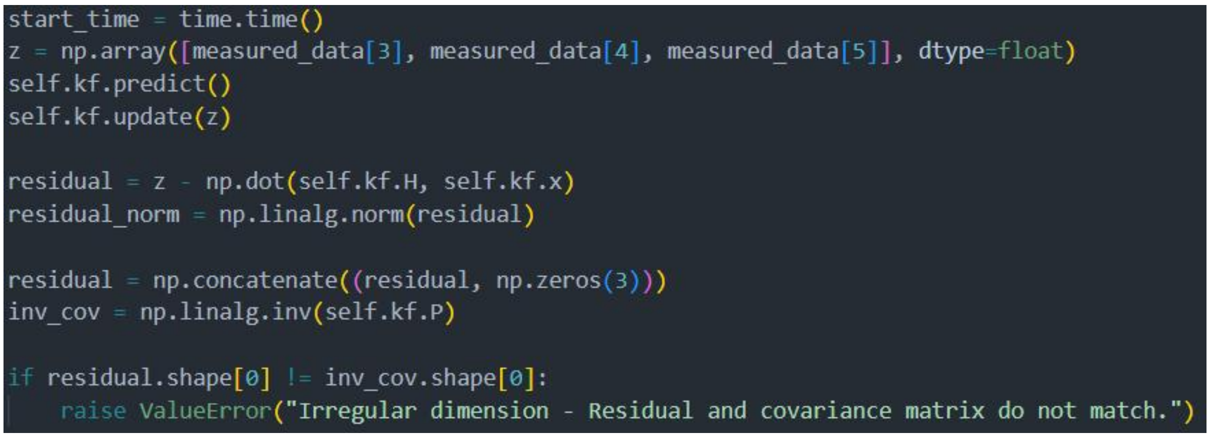


```python
def trigger_attack(self):
    print("Replay Attack Triggered")
    msg_to_replay = random.choice(self.log)
    replay_msg = can.Message(
        arbitration_id=msg_to_replay.arbitration_id,
        data=msg_to_replay.data,
        is_extended_id=msg_to_replay.is_extended_id
    )
    replay_msg.timestamp = time.time()
    self.bus.send(replay_msg)
```

5.4 Detect replay attack

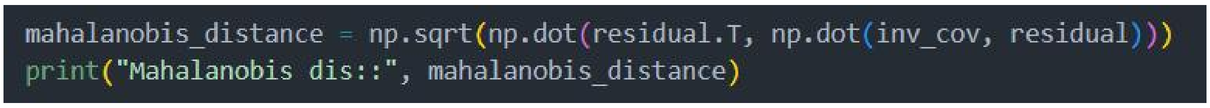
The Kalman filter using Python is derived with 'filter py' library provided by Python which handles the algorithm. The latest version can be downloaded from here https://filterpy.readthedocs.io/en/latest/

This library provides a detailed documentation on how to apply Kalman Filter into our code. Based on this documentation, the Kalman filter is applied to detect any potential replay attack.



```python
start_time = time.time()
z = np.array([measured_data[3], measured_data[4], measured_data[5]], dtype=float)
self.kf.predict()
self.kf.update(z)

residual = z - np.dot(self.kf.H, self.kf.x)
residual_norm = np.linalg.norm(residual)

residual = np.concatenate((residual, np.zeros(3)))
inv_cov = np.linalg.inv(self.kf.P)

if residual.shape[0] != inv_cov.shape[0]:
    raise ValueError("Irregular dimension - Residual and covariance matrix do not match.")
```

Furthermore, the Mahalanobis Distance mathematical equation is derived into python code in order to calculate its distance for each messages.

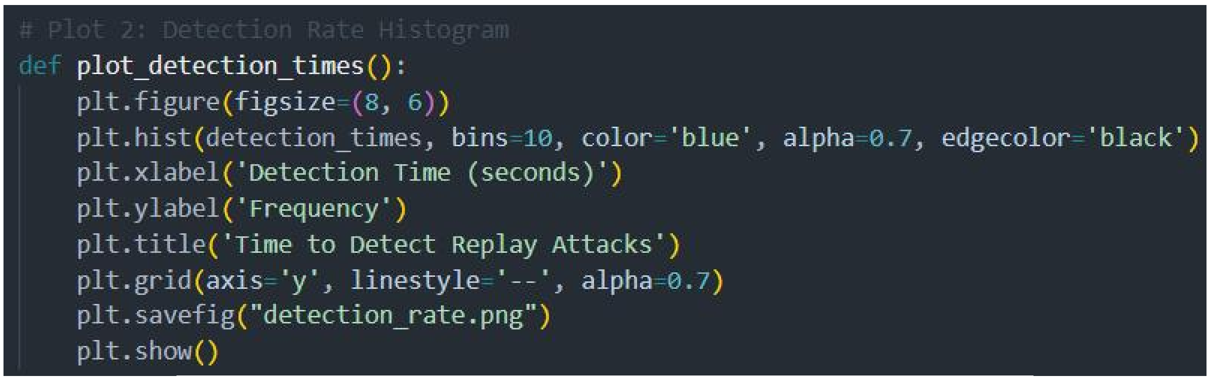


```python
mahalanobis_distance = np.sqrt(np.dot(residual.T, np.dot(inv_cov, residual)))
print("Mahalanobis dis::", mahalanobis_distance)
```

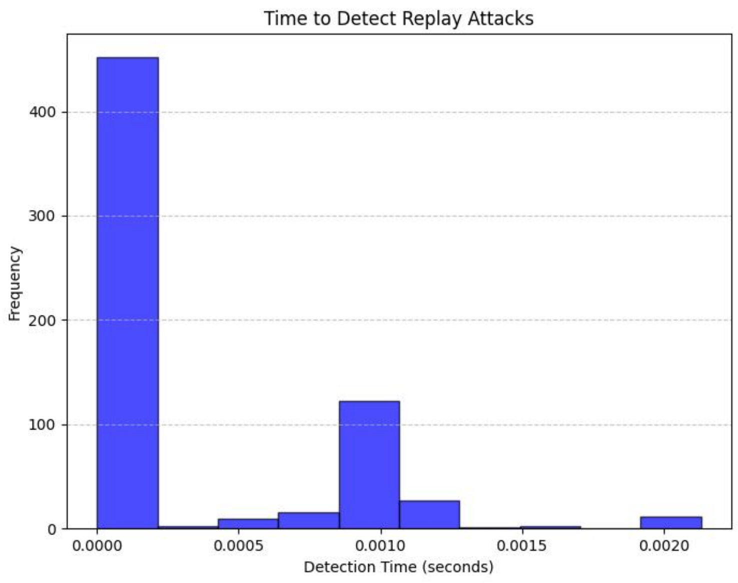
6 Results

6.1 Detection rate

The detection rate is derived based on the ratio of number of anomalies detected with the total number of anomalies present. The approach has resulted with significant results in detecting a replay attack between the range of 0.00 to 0.20 seconds which proves the technique detects a potential anomaly in a short span of time. The detection rate of replay attack is obtained by accumulating the time taken to detect each attacks and the graph was plot using 'matplotlib' library.

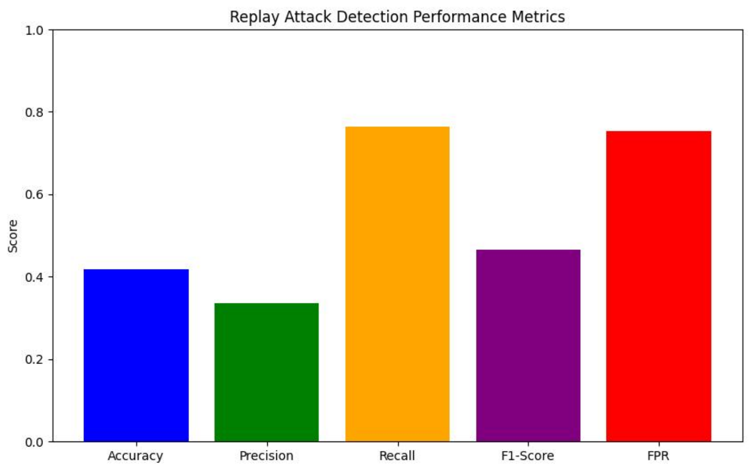


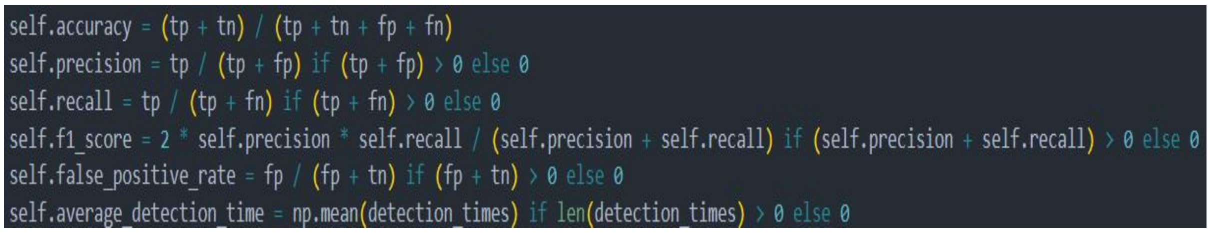
```
# Plot 2: Detection Rate Histogram
def plot_detection_times():
    plt.figure(figsize=(8, 6))
    plt.hist(detection_times, bins=10, color='blue', alpha=0.7, edgecolor='black')
    plt.xlabel('Detection Time (seconds)')
    plt.ylabel('Frequency')
    plt.title('Time to Detect Replay Attacks')
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.savefig("detection_rate.png")
    plt.show()
```



6.2 Overall performance

This overall performance with combination of Kalman filter and Mahalanobis distance has resulted with higher true positive rate of 77%, which significantly improves handling multidimensional states of an AV. With 'matplotlib' library and using the mathematical equations for each metrics, the overall performance is derived as shown below.

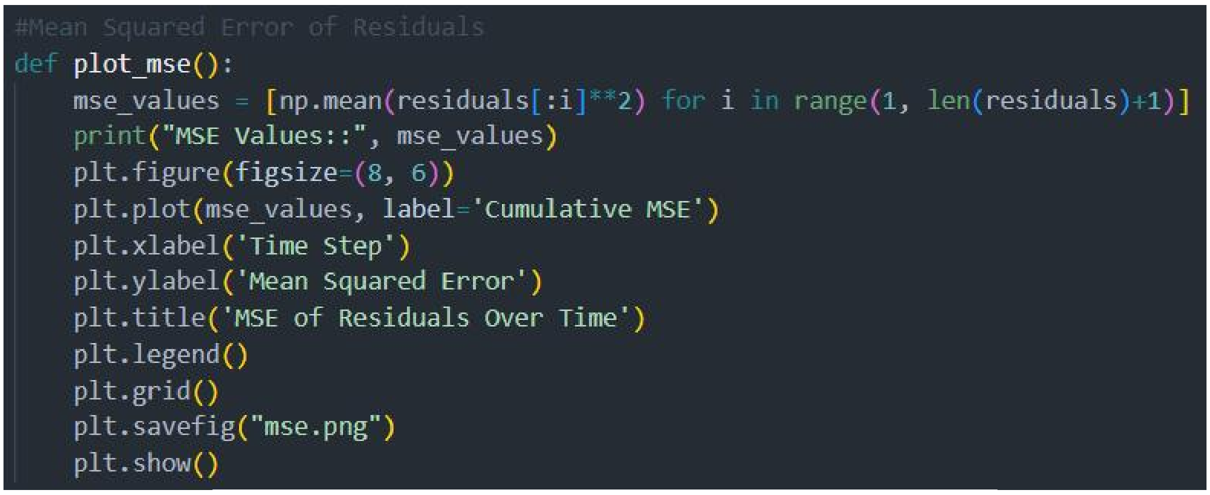




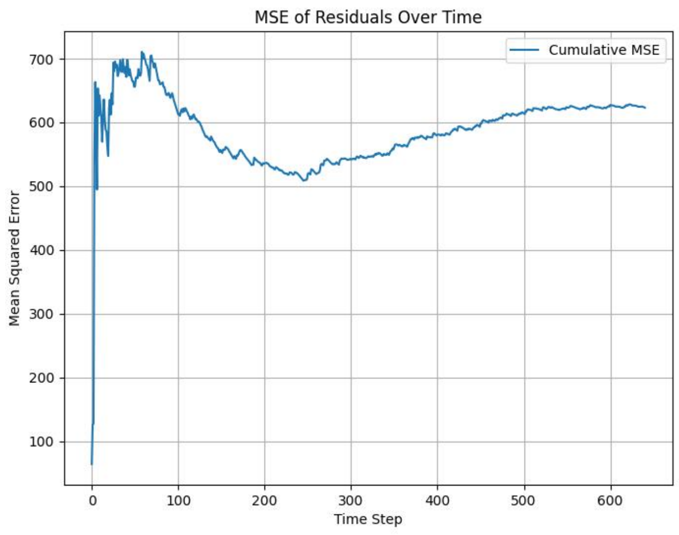
```
self.accuracy = (tp + tn) / (tp + tn + fp + fn)
self.precision = tp / (tp + fp) if (tp + fp) > 0 else 0
self.recall = tp / (tp + fn) if (tp + fn) > 0 else 0
self.f1_score = 2 * self.precision * self.recall / (self.precision + self.recall) if (self.precision + self.recall) > 0 else 0
self.false_positive_rate = fp / (fp + tn) if (fp + tn) > 0 else 0
self.average_detection_time = np.mean(detection_times) if len(detection_times) > 0 else 0
```

6.3 Mean Squared Errors (MSE) of residuals

The MSE of residuals is evaluated to differentiate between the Kalman Filter predictions from the observed values. With ‘matplotlib’ library and using the mathematical equations to calculate the mean values of each residuals, the MSE for residuals are derived and plotted as shown below.

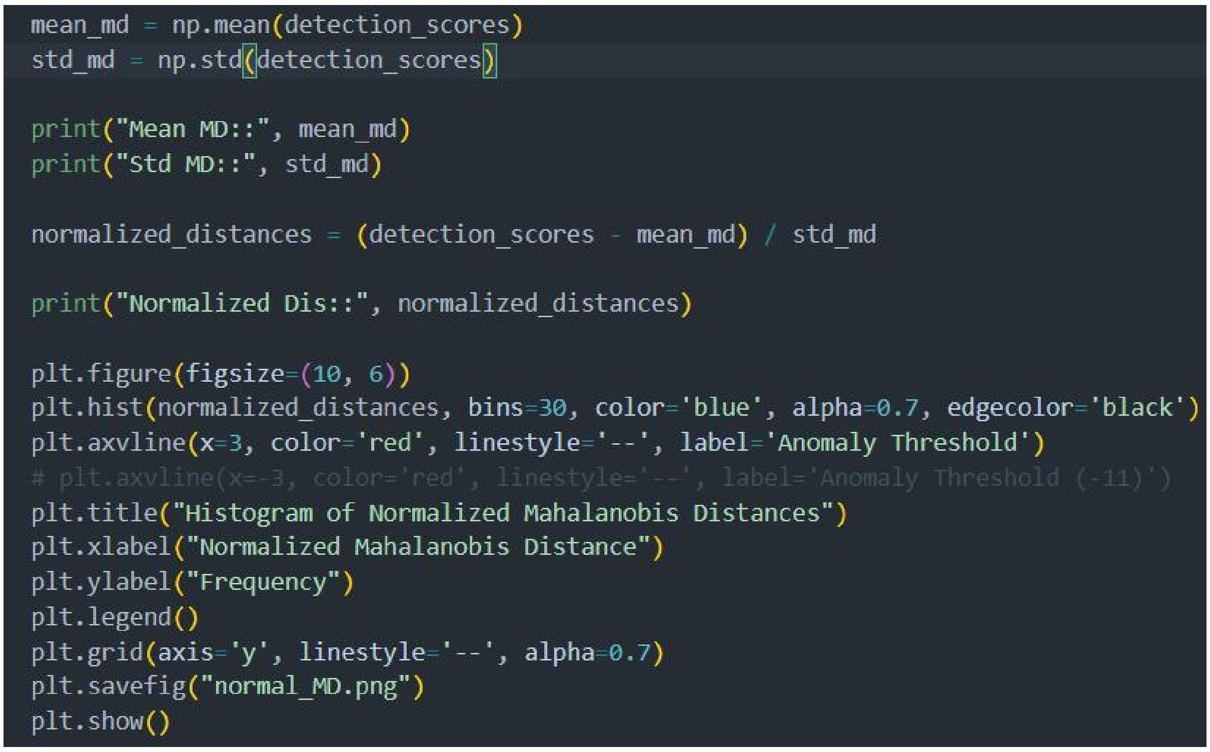


```
#Mean Squared Error of Residuals
def plot_mse():
    mse_values = [np.mean(residuals[:i]**2) for i in range(1, len(residuals)+1)]
    print("MSE Values::", mse_values)
    plt.figure(figsize=(8, 6))
    plt.plot(mse_values, label='Cumulative MSE')
    plt.xlabel('Time Step')
    plt.ylabel('Mean Squared Error')
    plt.title('MSE of Residuals Over Time')
    plt.legend()
    plt.grid()
    plt.savefig("mse.png")
    plt.show()
```

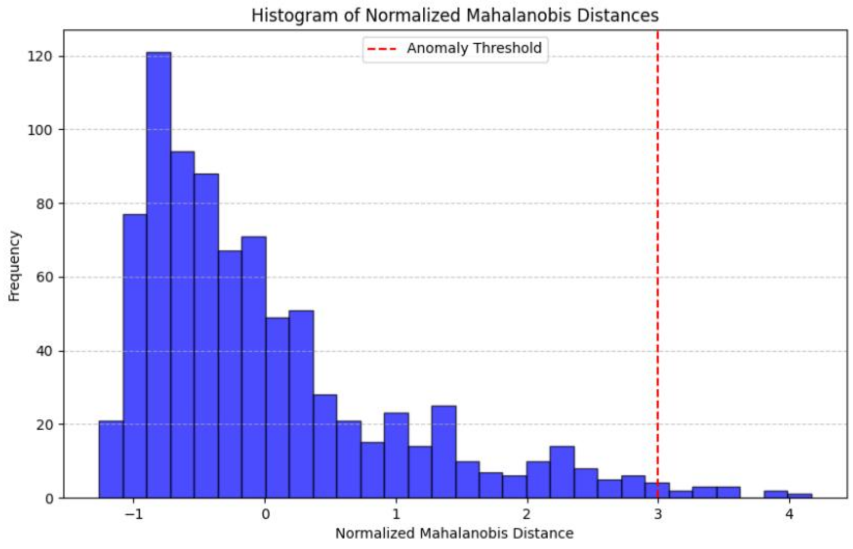


6.4 Normalized Mahalanobis Distance distribution

The normalized Mahalanobis distance is derived based on calculating the mean and standard deviations of the overall detection scores. The following images shows how the metrics are plotted using ‘matplotlib’ library and its results.

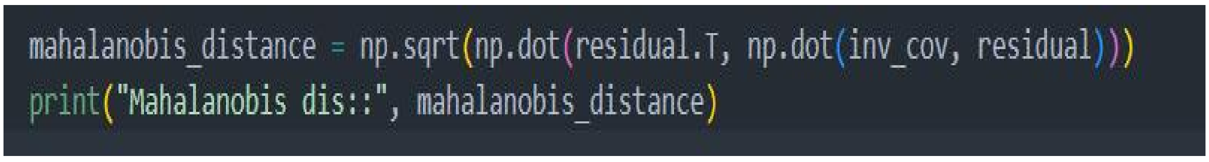


```python
mean_md = np.mean(detection_scores)
std_md = np.std(detection_scores)

print("Mean MD::", mean_md)
print("Std MD::", std_md)

normalized_distances = (detection_scores - mean_md) / std_md

print("Normalized Dis::", normalized_distances)

plt.figure(figsize=(10, 6))
plt.hist(normalized_distances, bins=30, color='blue', alpha=0.7, edgecolor='black')
plt.axvline(x=3, color='red', linestyle='--', label='Anomaly Threshold')
# plt.axvline(x=-3, color='red', linestyle='--', label='Anomaly Threshold (-11)')
plt.title("Histogram of Normalized Mahalanobis Distances")
plt.xlabel("Normalized Mahalanobis Distance")
plt.ylabel("Frequency")
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.savefig("normal_MD.png")
plt.show()
```

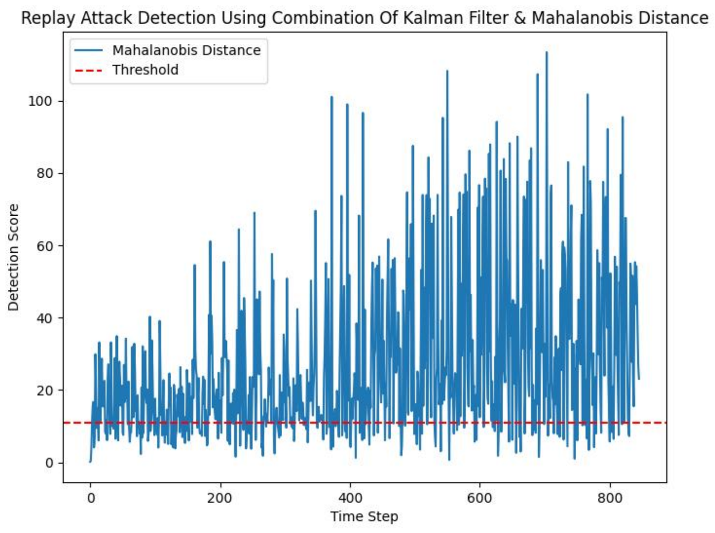


6.5 Detection score

The detection score of the overall Mahalanobis distance is calculated based on accumulated detection scores of every messages transferred into the CAN bus model. The approach for calculating the detection score for each message and its following outcome is shown below.

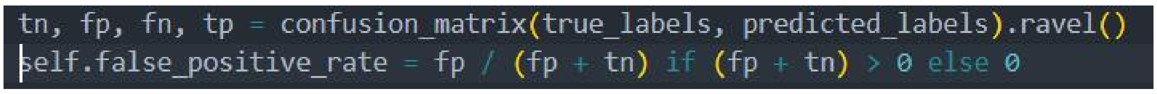


```python
mahalanobis_distance = np.sqrt(np.dot(residual.T, np.dot(inv_cov, residual)))
print("Mahalanobis dis::", mahalanobis_distance)
```

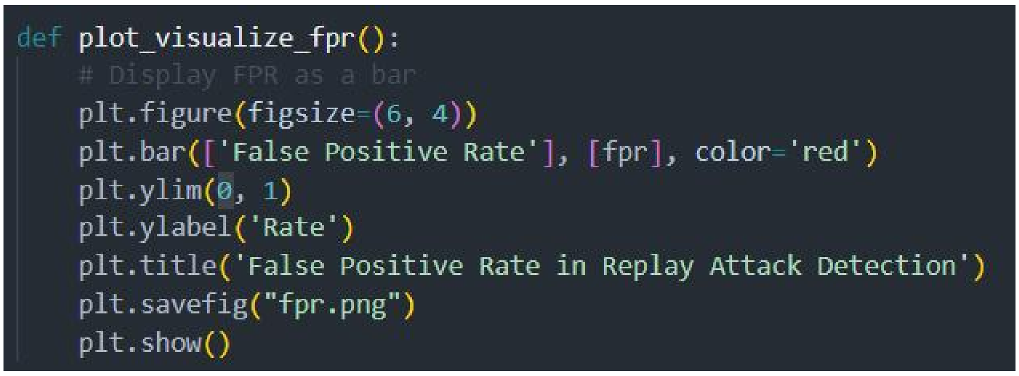


6.6 False Positive Rate in noisy environment

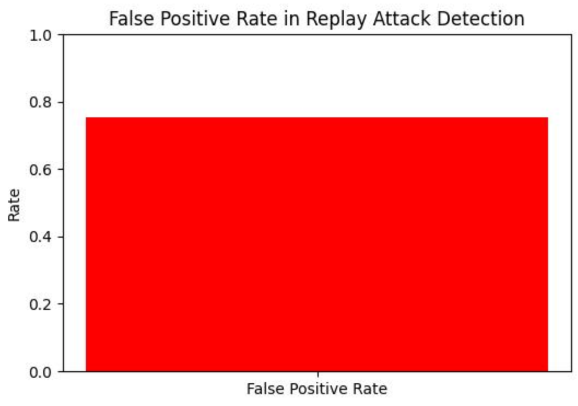
The false positive rates is derived based on the false positive rates and true negative rates resulted from the confusion matrix. This final false positive rates are plotted using 'matplotlib' library.



```
tn, fp, fn, tp = confusion_matrix(true_labels, predicted_labels).ravel()
self.false_positive_rate = fp / (fp + tn) if (fp + tn) > 0 else 0
```



```
def plot_visualize_fpr():
    # Display FPR as a bar
    plt.figure(figsize=(6, 4))
    plt.bar(['False Positive Rate'], [fpr], color='red')
    plt.ylim(0, 1)
    plt.ylabel('Rate')
    plt.title('False Positive Rate in Replay Attack Detection')
    plt.savefig("fpr.png")
    plt.show()
```



References

ApolloScapeAuto (2018). dataset-api/trajectory_prediction at master · ApolloScapeAuto/dataset-api. [online] GitHub. Available at: https://github.com/ApolloScapeAuto/dataset-api/tree/master/trajectory_prediction [Accessed 11 Dec. 2024].

Can, P. (n.d.). python-can 4.3.1 documentation. [online] python-can.readthedocs.io. Available at: https://python-can.readthedocs.io/en/stable/.

Python (2019). Download Python. [online] Python.org. Available at: https://www.python.org/downloads/.

Python, F. (n.d.). FilterPy — FilterPy 1.4.4 documentation. [online] filterpy.readthedocs.io. Available at: https://filterpy.readthedocs.io/en/latest/.

Visual Studio Code (2016). Visual Studio Code. [online] Visualstudio.com. Available at: https://code.visualstudio.com/Download.