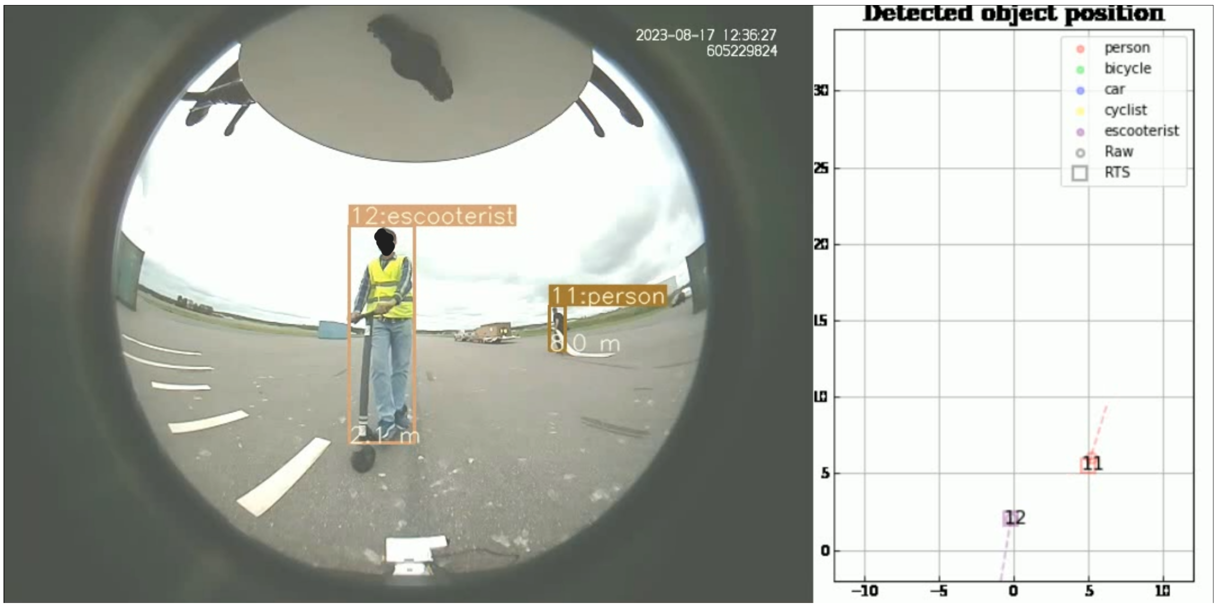
Using Machine Learning to Estimate Road-user Kinematics from Video Data

Luhan Fang, luhan@student.chalmers.se
Oskar Malm, omalm@student.chalmers.se
Yahui Wu, yahuiw@student.chalmers.se
Tianshuo Xiao, tianshuo@student.chalmers.se
Minxiang Zhao, minxiang@student.chalmers.se

December 2023



Abstract

Each year, there are over one million people who sustain fatal injuries in traffic-related crashes, with vulnerable road users, often abbreviated as VRUs, being involved in more than half of the crashes. In the context of road safety, VRUs are mainly pedestrians, cyclists, motorcyclists, and e-scooterists. To mitigate the crashes, it is essential to understand the causation mechanisms. Naturalistic data has been recognised as a good tool to understand the trafficant's behaviour and address safety concerns within the field of traffic safety research. Traditionally, critical events are identified from naturalistic data using the kinematic information from the sensors onboard the vehicles. Video footage for the trip corresponding to the critical events is then used to validate and annotate the events. While this is a reliable method when it comes to identification of crashes, near-crashes may not display any anomalies in the sensor readings thereby going unidentified. These near-crash events would be visible in the video footage, but the manual identification of these by watching videos is not feasible because the amount of videos is too large for the human eyes. Therefore, developing tools that can identify and estimate the position of different road users using the video footage is essential and will enable automation of process of identifying critical events. This report describes such models and also delves into the application of machine learning to allow identify the severity of imminent critical interactions among road users in the future.

This project investigates how models can be developed to estimate and predict the position and kinematics of various road users from video data from a camera mounted on an e-scooter. The initial generation of bounding boxes and categories for road users utilized You Only Look Once (YOLOv7) algorithms. The detection for cyclists was achieved by a simple rule-based model calculating the overlap area between the pedestrian and bicycle detected by YOLOv7. The e-scooterists detection model was implemented by combining YOLOv7 and MobileNetV2 models. Different machine learning models were trained to estimate distance for the four different road users: pedestrians, cyclists, e-scooterists, and cars separately using LiDAR and GPS data as the position ground truth. The input for these models was derived from bounding box data extracted from videos. Furthermore, a DBSCAN-based noise remover was used to remove the outlier point of the distance estimation model to filter out points with excessive errors. Finally, a Rauch-Tung-Striebel smoother was applied to the output of the noise remover to improve the distance estimation accuracy and generate both the relative position and velocity of the target road user.

It was concluded that the object detection model could achieve an accuracy over 90%, and the distance estimation achieved the highest accuracy when using polar coordinates for all road users, compared with using the cartesian system. The highest R^2 score for distance estimation was obtained with a k-nearest neighbors regression model (with $n = 2$) using the pixel position in x and y direction of center point of the bottom of bounding box, height, and width of the bounding box as input. As a consequence, the e-scooterist model achieved a R^2 score of 0.978, while the cyclist and car models attained commendable scores of 0.92 and 0.96 respectively. This means that the distances predicted from the model are highly accurate. These models can now be used to detect critical interactions among road users using the naturalistic data collected using e-scooters.

Acknowledgements

Throughout the project, we have been fortunate to receive assistance from our supervisors, Marco Dozza, Rahul Rajendra Pai, and Alexander Rasch, who have helped a lot in the form of weekly meetings, fast responses over email, and guiding us in the right direction to finish the project on time. We also want to express our gratitude to Karan Bharti based on whose pedestrian position estimation model we could continue our work more easily. Special thanks to Veoneer (now Magna) for letting us collect data on their test track with their vehicle, and the project e-SAFER and MicroVision that financed the collection. Finally, we extend our thanks to Kumar Apurv, Renran Tian, and Rini Sherony for their previous studies on e-scooterists detection which gave us a dataset of e-scooterists and a framework of the detection model.

Contents

1 Introduction

1.1 Background

Every year, around 1.3 million people are killed globally in traffic-related crashes and between 20 to 50 million people sustain non-fatal injuries according to the World Health Organization (WHO) [1]. In more than half of the annual fatal crashes, VRUs are involved, which means that the risk of fatal crashes is particularly high for this group in particular. A VRU is usually classified as someone who is not inside of a vehicle, in this group there are, for example, cyclists, e-scooterists, and pedestrians [2]. In recent years the number of e-scooters has increased as a form of sustainable transport in our cities. This means that people who are using e-scooters as a form of transportation have a high risk of injury and it is essential to understand the interactions between them and other road users. Since this is a fairly new mode of transport, there isn't a lot of research done on how e-scooterists interact with different road users.
To reduce the involvement of e-scooterists and other road users in crashes, it is important to learn the mechanism of how a crash could happen. Naturalistic data is collected from real traffic which captures the genuine driver behavior and how the driver interacts with other road users before a crash happens. Therefore, it is a good tool to understand the crash mechanism and identify critical events. In the traditional active safety area, critical events are often identified by analyzing the kinematic information based on signals from sensors onboard vehicles. However, the sensor signals are insufficient for near-crash identification since signals would not show any anomalies. Video footage from the onboard camera can easily capture near-crash events, but manually watching those videos to identify critical events can be time-consuming, and kinematic information cannot be derived directly from the video footage. Some vision-based methods need to be developed to estimate the position of different road users and enable automation of the process of identifying critical events. By using artificial intelligence and machine learning this process would not only be automatic, saving time, but also objective, based on a predetermined threshold for what is a critical situation.

1.2 Literature review

Surveying the relevant literature, we come across various methods for e-scooterist identification and distance estimation.
In the article by Apurv et al.[3][4], he introduced a vision-based system for detecting e-scooter riders in natural scenes, employing YOLOv3 and MobileNetV2. While achieving a commendable recall and accuracy, the methodology's strengths lie in its efficient pipeline. However, the absence of an existing dataset poses a limitation.
In the article by Zhu[5], a learning-based model for object-specific distance estimation is proposed. In the article, the features in the image are first extracted by convolutional neural networks, and then the distance is predicted along with the bounding box of the identified object. The strengths include addressing challenges with traditional methods and introducing extended datasets. However, the effectiveness of the model is contextualized primarily for objects on curved roads.
Report by Bharti[6] delves into a machine learning approach for automating video data reduction in analyzing road user interactions. Leveraging LiDAR and camera data, the

study aims to identify the position of pedestrians with respect to the e-scooters to identify critical events. The scope of the work here is limited to the identification of pedestrian positions and excludes e-scooter riders and cyclists. The methodology's robustness lies in its multidimensional data utilization.
The article by Davydov[7] focuses on supervised object-specific distance estimation using monocular images for autonomous driving. The proposed lightweight convolutional deep learning model outperforms existing methods. Its strengths include accurate distance estimation and relevance to advanced driver assistance systems, although challenges may arise in various real-world scenarios.
The papers and reports above address a crucial aspect of road safety by detecting e-scooterists and estimating object-specific distances. As crashes involving VRUs continue to cause injuries, understanding the intricate interactions between emerging sustainable transport modes and other road users becomes imperative. To understand why critical events occur, it is crucial to identify these interactions. Traditional sensor readings may capture crashes, but numerous near-crash scenarios, discernible only in videos, remain unexplored due to resource-intensive manual analysis. Using AI and machine learning, these papers propose algorithms that efficiently identify and assess interactions, pinpointing potential threats and critical scenarios based on predefined thresholds.
However, the current method's limitations lie in its exclusive reliance on computer vision for VRUs identification, without integrating LiDAR and camera data for distance prediction. While traditional camera calibration provides a robust theoretical foundation, it involves manual processes and may face challenges in dynamic environments. In contrast, machine learning automates the process and adapts to diverse conditions. In this project, we use LiDAR data which offers accurate ground truth for model training. This approach proves superior to traditional camera calibration methods, enhancing accuracy and adaptability.

1.3 Aims and objectives

The overall objective of this project is to automatically estimate kinematic information of road users from video data collected from an e-scooter. By developing methods to analyze video data collected in the real world by e-scooterists, we can objectively investigate rider behavior and interactions. The data and knowledge could then be used to develop new active safety systems or make riders aware of high-risk scenarios that can occur during riding. To achieve this goal, a number of tasks were completed:

1. **Object detection and classification** from video data, this was done to ensure that different road users are detected and classified correctly and reliably. But, also be able to differentiate a pedestrian, cyclist, and an e-scooterist which in a traditional computer vision algorithm would be identified as a pedestrian.

2. **Processing LiDAR and GPS data** to acquire accurate distances and angles which were collected in a controlled environment. This data was then used to establish a ground truth which was used to train and test the machine learning algorithms to be able to determine distances from the video data. Finally, the data from the different sensors was synchronized, due to the different activation times and sample rates, to form the training data.

3. **Train the machine learning models** based on the object detection and classification which generated bounding boxes for each road user cyclist, e-scooterist, and

car. The training data for the model is the combination of the data from object detection and the distance data from the LiDAR and GPS (ground truth).

4. **Validation of the models** was done by applying the models on the validation data to compare predicted values to the ground truth.
5. **Filter the predictions** done by the models to ensure that the final predictions are less affected by noise.
6. **Test on a sample** of the naturalistic data to determine the performance in real world conditions.

2 Theory

This chapter will present and explain the different theoretical aspects used during this project.

2.1 Coordinate systems

This subsection will explain the coordinate systems and the transformations used during the work.

2.1.1 Polar coordinate system

Coordinate systems are used to represent the location of a point or object in space, which in this project was used to calculate the radial distances and angles to objects detected by the camera, this information was then related to pixels in the recorded video data.

To represent a point in space with a radial distance (r) and an angle (θ), to the x-axis, the polar coordinate system is used and to transform a point (x, y) in the Cartesian coordinate system to a radius and angle, the following formulas are used:

$$r = \sqrt{(x^2 + y^2)} \text{ and } \theta = atan\left(\frac{y}{x}\right) \tag{1}$$

To transform from polar coordinates to Cartesian coordinates the following formulas are used:

$$x = r \cdot cos(\theta) \text{ and } y = r \cdot sin(\theta) \tag{2}$$

2.1.2 Rotation of coordinate systems

Since different objects are rotated in relation to each other it is useful to rotate one of them so that the systems becomes aligned to more easily do calculations. This can be done by applying the rotation matrix, which has the following form in two dimensions:

$$\begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \tag{3}$$

Which is used then to calculate the rotated coordinates (x' and y') in the following way:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{4}$$

Where x' and y' are rotated by the angle θ. The direction of the rotation is defined by the angle θ, where a positive angle is a counter-clockwise (positive mathematical direction) rotation, and a negative angle rotates the system clockwise.

2.2 YOLOv7

You Only Look Once (YOLOv7) [8] is an object detection model, which is based on the backbone-head pattern. The backbone part is composed of several convolutional layers and max-pooling layers to extract the features at different levels from the input image. There is also a net structure called ELAN to control the gradient to extract more features and have better robustness. The head part takes responsibility for further feature extraction, classification, and detection. It has represented convolutional layers (RepConv in the configuration file) to do the detection part at the low, middle, and high levels to cover the targets of interest from small objects to big objects and from simple detection to complicated detection.
YOLOv7 can be trained on various data, and once the training is finished, a pre-trained YOLOv7 model with its weight and configuration file will be generated. The weights and configuration can be used for other users to implement the detection without training or retraining the model with less time taken.

2.3 MobileNetV2

MobileNet [9] is a lightweight convolutional neural network (CNN), which features depth-wise separable convolution. This feature drastically reduces computational costs while minimizing loss in accuracy. Compared to the previous version, MobileNetV2 [10] includes an inverted residual and linear bottleneck. Linear bottleneck solves the problem that the ReLu activation function causes too much information loss in low dimensions. While the inverted residual structure can reduce the usage of memory. As it is shown in Figure 1, the size of the input to the network is (160,160,3) scaled to [-1,1]. The output is a (5,5,1280) tensor. After adding a dense layer, an output of a value between 0 and 1 can be obtained. 0 represents the prediction result as a pedestrian, while 1 indicates that it is an e-scooter rider.

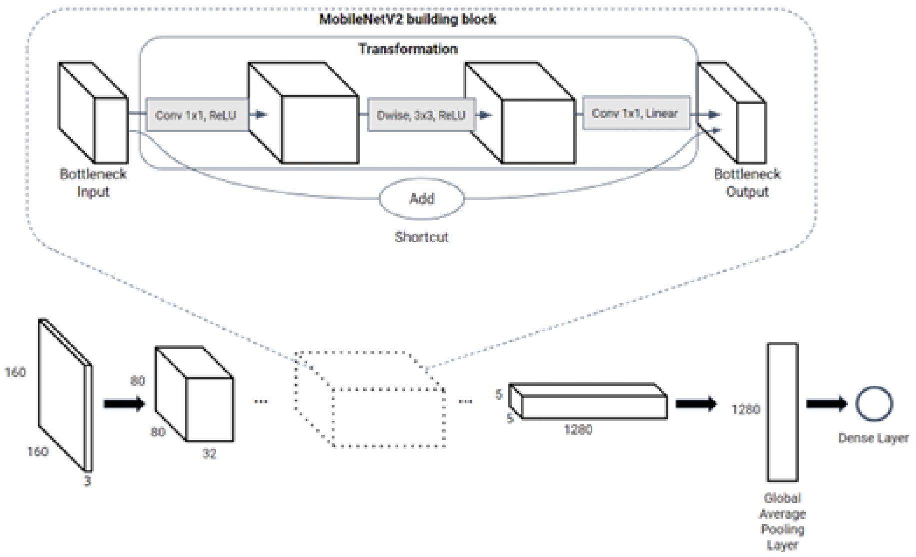


Figure 1: MobileNetV2 architecture for binary classification task [3].

2.4 Machine learning models

2.4.1 Linear Regression

Linear regression is a fundamental regression method that attempts to establish a linear relationship, minimizing the prediction error between input features and output labels. It fits a straight line by minimizing the squared differences between predicted and actual values, using the method of least squares. Linear regression is suitable when there is a linear relationship between features and the target.[11]

2.4.2 Decision Tree Regressor

Decision trees are tree-like structures where each node represents a test on a feature, each branch represents the result of the test, and each leaf node stores a target value. It recursively partitions the data into different regions, with the target value in each region represented by the average of the samples in that region. Decision trees are suitable for capturing non-linear relationships, but they can be prone to overfitting.[12]

2.4.3 K-nearest neighbors

The K-nearest neighbors algorithm predicts by finding the closest K neighbors to a new sample and averaging or weighted averaging their target values. It determines the most similar samples based on the distance in the feature space. The K-nearest neighbors are suitable for data with local patterns and perform well when there is a clear local structure.[13]

2.4.4 Random Forest Regressor

Random forests are an ensemble learning method composed of multiple decision trees. It reduces overfitting by averaging the predictions of each tree. Random forests introduce randomness during tree construction, such as feature subset sampling, to increase model diversity. It performs well on large datasets with many features, effectively modeling complex relationships.[14]

2.4.5 MLP Regressor

Multi-layer perceptron (MLP) is an artificial neural network with multiple layers (input, hidden, output). In regression tasks, the output layer typically has a single node. It learns nonlinear relationships in data through forward and backward propagation algorithms. MLP is suitable for complex non-linear relationships, demonstrating strong expressive power for large-scale datasets and highly nonlinear problems.[15]

2.5 DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise)[16] is a clustering algorithm that groups data points based on their density in the feature space. It identifies core points as those within a specified radius containing at least a minimum number of neighboring points. Clusters are formed by connecting core points and their reachable neighbors, while points outside such clusters are classified as noise. This method effectively captures dense regions in the data, enabling the discovery of arbitrary-shaped clusters and the filtration of noise points, making it robust for various spatial clustering applications. From Figure 2, consider a randomly selected point, A, in the sample dataset. Begin by defining a radius and a minimum number of sample points required within a circular region centered on A. If the circle encompasses enough sample points, such as B, C, M, and N, the center of the circle is shifted to these internal points. This process of drawing circles is repeated until the number of enclosed sample points becomes less than the specified minimum number of points (minPts).

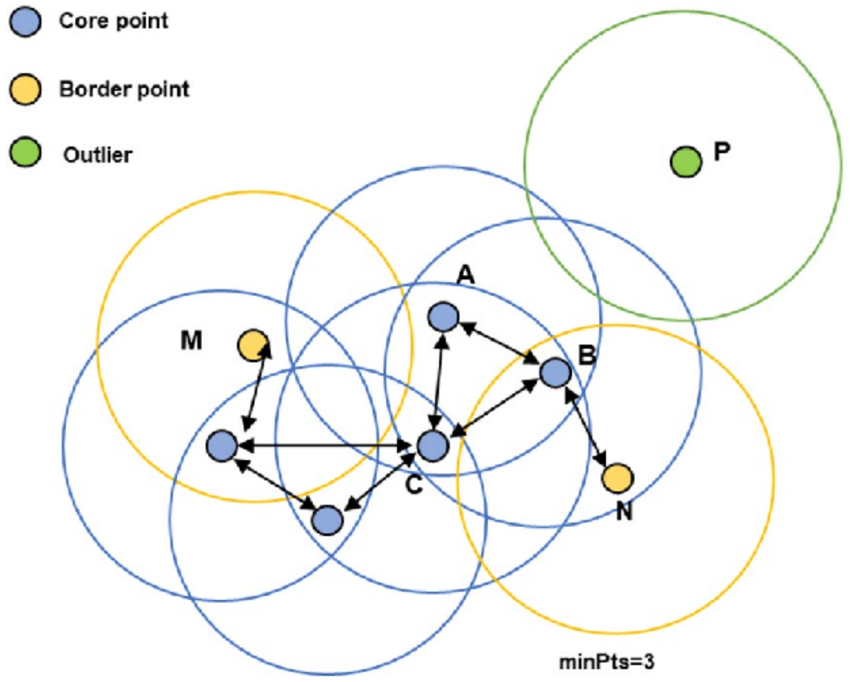


Figure 2: Principle of DBSCAN algorithm[17].

DBSCAN groups data points into clusters, discerning regions of high density while classifying outliers as noise. In the context of filtering, this translates to the removal of data points that deviate from expected patterns or exhibit anomalous behavior. By leveraging DBSCAN's ability to form clusters based on data density, the algorithm enhances data quality by filtering out noise, ensuring that subsequent analyses and modeling efforts are conducted on a more accurate and reliable dataset.

2.6 Rauch–Tung–Striebel smoother

The Rauch–Tung–Striebel (RTS) smoother [18] is a mathematical method used to improve the accuracy of estimating the state variables of a system by incorporating both past and future measurements. The RTS smoother could be represented as a state-space model, which consists of a state transition equation (describing how the system evolves) and a measurement equation (relating the system's state to the observed measurements). The system state is first estimated by a Kalman filter, which recursively processes incoming measurements to estimate the current state of the system. Then, the RTS smoother performs a backward pass through the state data. The backward pass revisits the estimates produced by the Kalman filter and refines them by incorporating information from future

measurements.
Several parameters, including the transition matrix, the observation matrix, the transition covariance, the observation covariance, the initial state mean, and the initial state covariance, need to be specified before implying the RTS smoother. The state is a vector that describes the kinematics information. Both the transition matrix and the transition covariance compose the motion model, and the observation matrix and the observation covariance compose the measurement model.
The RTS smoother not only enhances the accuracy of state estimation by incorporating both past and future measurements, resulting in improved estimates of a system's true state over time, but also, in this project, estimates the parameters (velocities) in the state that not included in the measurements (only positions). The RTS smoother can not only estimate the kinematic information of one object but also improve the data quality and distance estimation accuracy.

3 Methodology

In this chapter, the methods used will be presented and discussed.

3.1 Data collection

The ground truth data that is used to train the position estimation model is collected by an e-scooter with a camera and LiDAR mounted on it as shown in Figure 3. On the car, the GPS is mounted as shown in Figure 6 and is also logged during the data collection. During the data collection, typical objects like pedestrians, cyclists, e-scooter riders and cars will hover in front of the e-scooter so that different angles of different objects at different distances can be captured.

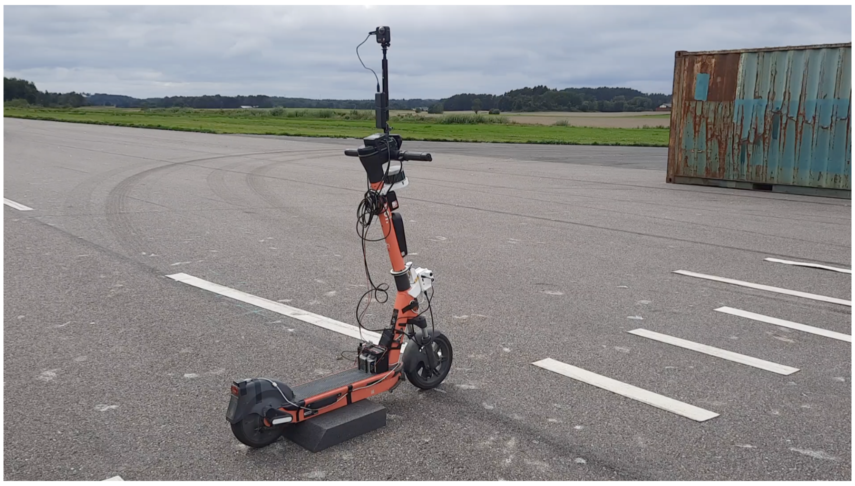


Figure 3: E-scooter for data collection, equipped with a camera, LiDAR, and logging equipment.

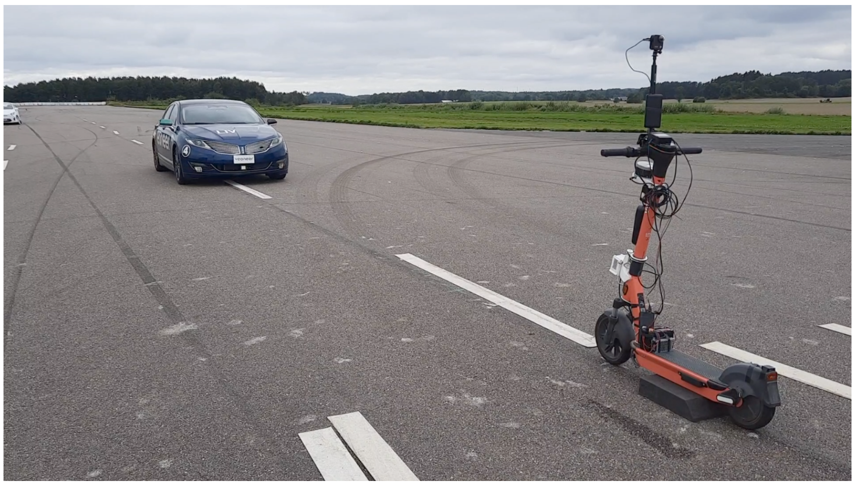


Figure 4: The e-scooter from another angle.

The e-scooter's camera is a monocular fish-eye camera with a resolution of 720x532 pixels, a field of view (FOV) of 220° and a frame rate of 30 frames per second (fps). While the

LiDAR is a VLP16 manufactured by Velodyne Lidar with a vertical and horizontal FOV of 30 degrees and 360 degrees respectively distributed over 16 channels and a frame rate of 10 fps.

The car in Figure 5 that was used during data collection was a Lincoln MKZ with the following dimensions: a length of 4.930, a width of 1.864 m, and a height of 1.478 m. From the car, GPS data was obtained and analyzed, this is described in more detail in Chapter 3.3.1, from the GPS mounted in the geometrical center of the car, i.e., at (length/2, width/2) which can be seen in Figure 6.

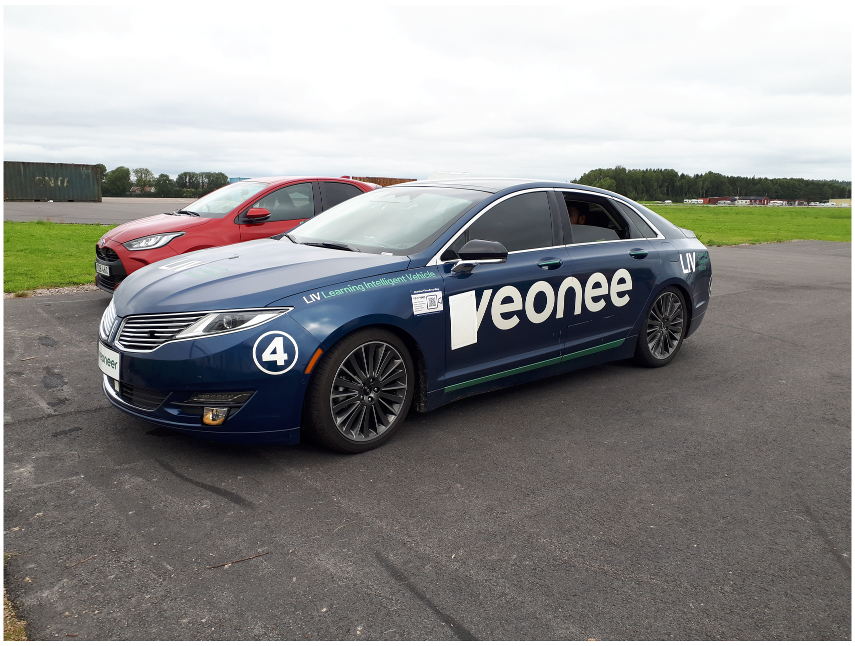


Figure 5: The car used during testing, equipped with GPS mounted in the car's geometrical center, i.e., at (length/2, width/2).

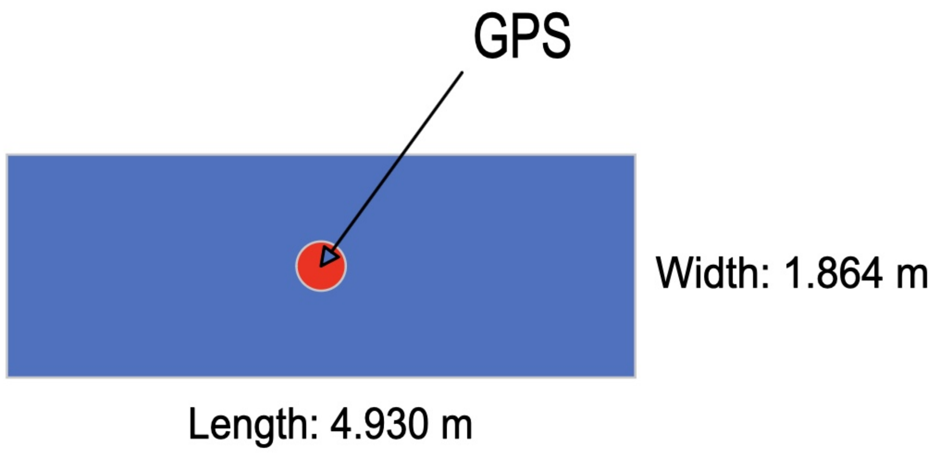


Figure 6: Schematic drawing of the car and the mounting position of the GPS at the car's geometrical center (length/2, width/2).

3.2 Object detection and classification

In this part, the road users in the video were detected and classified. Firstly, the video was inputted to the YOLOv7 model for preliminary detection. Then, the e-scooterists and cyclists would be classified by the Person Classification Model (PCM) and Cyclist Detection Model (CDM). Figure 7 shows the pipeline of object detection and classification.

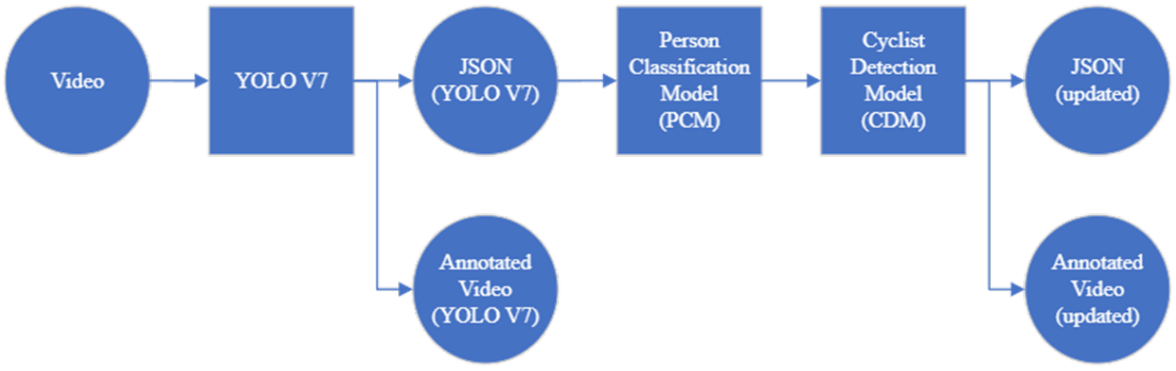

Video
YOLO V7
JSON (YOLO V7)
Person Classification Model (PCM)
Cyclist Detection Model (CDM)
JSON (updated)
Annotated Video (YOLO V7)
Annotated Video (updated)

Figure 7: Pipeline of the object detection and classification model.

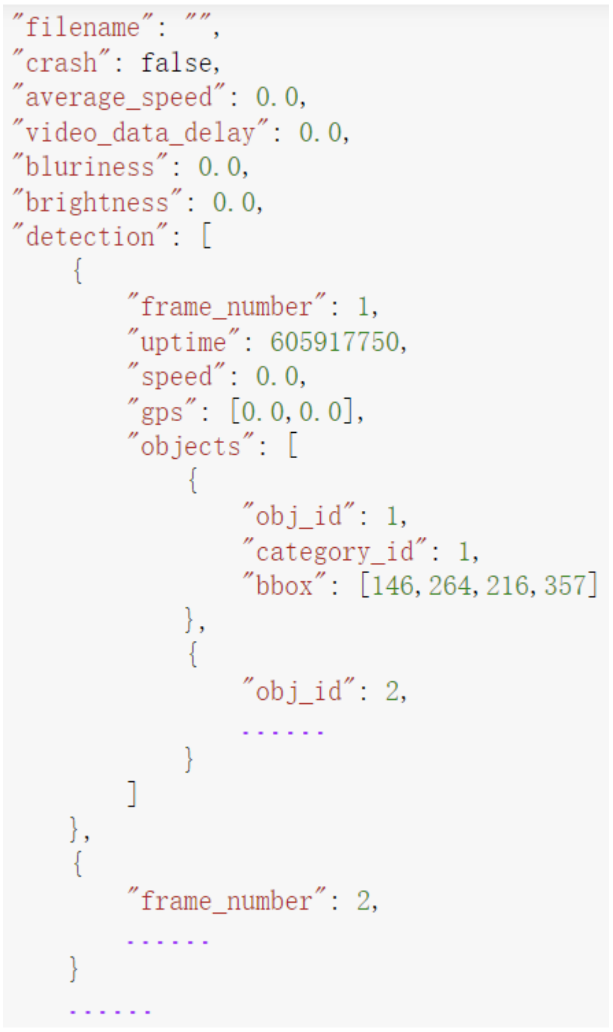
3.2.1 YOLO detection

YOLOv7 could detect the objects of desired categories from a video. The JSON file, whose structure is shown in Figure 8, and the annotated video, are both the outputs of the YOLOv7 model. The weight file 'yolov7x.pt', which is trained by the COCO dataset, is used as the weight of the YOLOv7 of the object detection model. A class filter of YOLOv7 could be set to make the YOLOv7 detect traffic objects only, which include 'person', 'bicycle', 'car', 'motorcycle', 'bus', 'train', and 'truck'. For each frame, the object's ID, the object's class, and the object's bounding box (BBox) could be detected by the YOLOv7 model and then be written to the JSON file. The uptime, which is the timestamp of the frame, could be detected by the Optical Character Recognition (OCR) model which executes the task that recognizing the characters in an image. Then the detected uptime will be written to the JSON file.

3.2.2 E-scooter rider detection

The e-scooter rider (e-scooterist) is identified by the PCM. Figure 9 shows the framework of the PCM model. The PCM model takes images of one person's ID as the input and then returns the class of the person, which could be pedestrian or e-scooterist. In the first step, all frames corresponding to the person's ID are extracted. For each frame, the BBox for the person is identified which is then extended using Equation 5 to obtain an enlarged bounding box. x_1, y_1, x_2, y_2 represent the coordinates of the top left and button right corners of the original BBox respectively. w, h are the width and the height of the original BBox respectively. x', y' are the top left coordinate of the extended BBox, and w', h' are the width and the height of the extended BBox respectively. This extended bounding box forms the input to the Image Classification Model (ICM).

$$\begin{cases} (w,h) = (x_2 - x_1, y_2 - y_1) \\ (x', y', w', h') = (x_1 - w, y_1, 3w, h + h/4) \end{cases} \tag{5}$$



```
″filename″: ″″,
″crash″: false,
″average_speed″: 0.0,
″video_data_delay″: 0.0,
″bluriness″: 0.0,
″brightness″: 0.0,
″detection″: [
    {
        ″frame_number″: 1,
        ″uptime″: 605917750,
        ″speed″: 0.0,
        ″gps″: [0.0,0.0],
        ″objects″: [
            {
                ″obj_id″: 1,
                ″category_id″: 1,
                ″bbox″: [146,264,216,357]
            },
            {
                ″obj_id″: 2,
                ......
            }
        ]
    },
    {
        ″frame_number″: 2,
        ......
    }
    ......
```

Figure 8: Example of the structure of a JSON file with BBox data for each category of detected objects.

The ICM, which is based on MobileNetV2, takes the extended person image as input and then returns a sigmoid score. The range of the sigmoid score is from 0 to 1, and the closer the score is to 1, the more likely the image's class is persons, and vice versa. The ICM is trained by the dataset [19]. By passing all extended images of one person's ID to the ICM, the score in each frame of the person's ID could be computed. According to Equation 6, the weighted score $S_{weighted}$ could be computed.

$$\begin{cases} S_{weighted} = \frac{\sum_{f=1}^{F} W_f S_f}{\sum_{f=1}^{F} W_f} \\ W_f = \begin{cases} 1 & \frac{w'_f h'_f}{160\times160} \geq 1 \\ \frac{w'_f h'_f}{160\times160} & \frac{w'_f h'_f}{160\times160} < 1 \end{cases} \end{cases} \tag{6}$$

In Equation 6, F represents the number of frames that the ID exists, S_f represents the score in each frame, W_f represents the weight of frame f, which is related to the rate between the extended image's size and the desired input size of the Image Classification Model, w'_f and h'_f represent the width and height of the extended image of frame f respectively. The reason for introducing the weight is to minimize the effect of low-quality

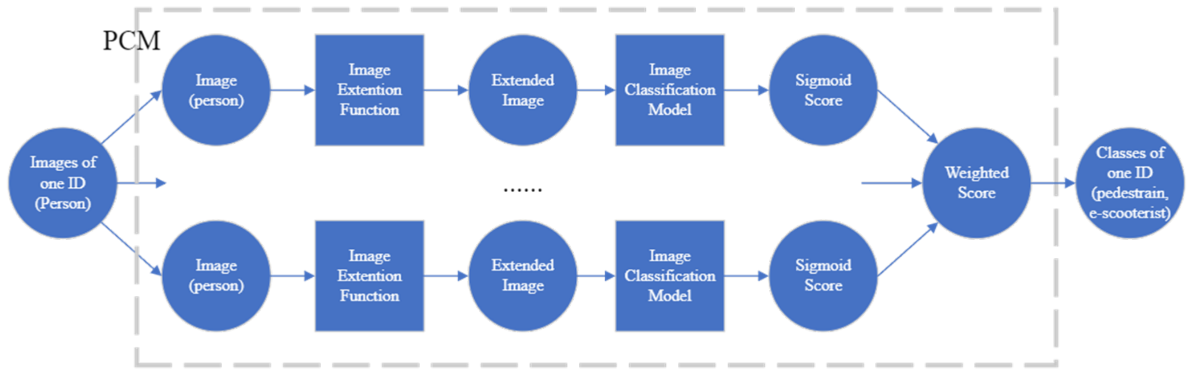

PCM
Images of one ID (Person)
Image (person)
Image Extention Function
Extended Image
Image Classification Model
Sigmoid Score
......
Weighted Score
Classes of one ID (pedestrain, e-scooterist)

Figure 9: The framework of the person classification model.

images which are more likely to be miss classified.

3.2.3 Cyclist detection

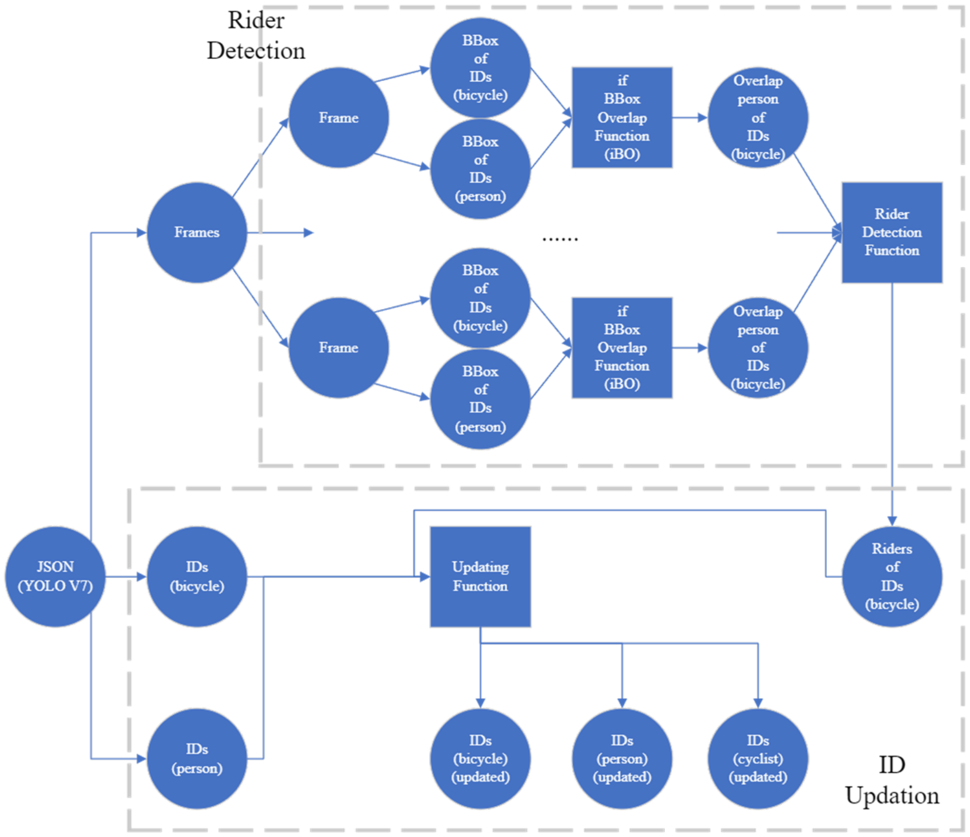


Figure 10: The framework of the cyclist detection model.

YOLOv7 can detect either persons or bicycles, but it cannot classify the person riding the bicycle as a cyclist. Figure 12 shows the framework of the cyclist detection model. The key to enable cyclist detection in the object detection model is the overlap area of the person and the cyclist. The left part of the Figure 11 shows the overlap area. If the overlap area between the person and the bicycle is larger than the threshold, the person could be considered as the overlap person of the bicycle in the current frame. The 'if BBox Overlap Function' (iBO) can check whether the person is the overlap person of the bicycle by comparing the metric $rate_{\text{overlap}}$, which is the rate between the overlap area and the whole area of the bicycle and person, with a threshold. Equation 7 shows how to calculate the metric, where variables' definitions are shown in Table 1, but both BBoxes should pass to the Overlap Checking Function which is shown in Figure 12 to make sure the overlap area is larger than 0.

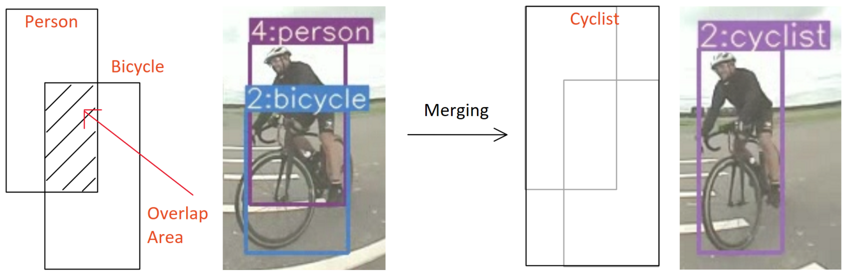


Figure 11: Schematic of person bicycle overlapping and BBox merging.

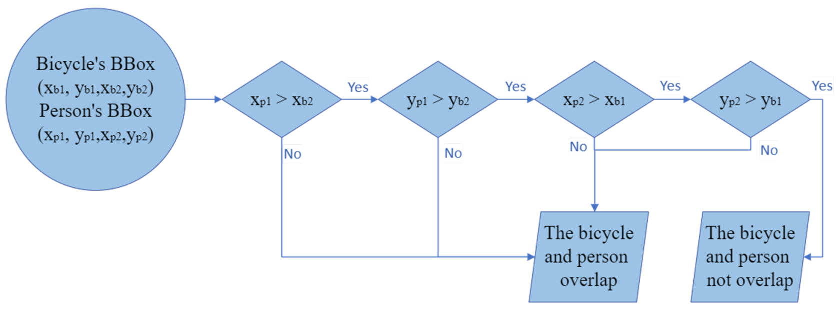


Figure 12: A schematic drawing of the overlap checking function.

$$\begin{cases} A_p = (x_{p2} - x_{p1})(y_{p2} - y_{p1}) \\ A_b = (x_{b2} - x_{b1})(y_{b2} - y_{b1}) \\ A_{\text{overlap}} = (\text{abs}\,(\min(x_{p2}, x_{b2}) - \max(x_{p1}, x_{b1})))\,(\text{abs}\,(\min(y_{p2}, y_{b2}) - \max(y_{p1}, y_{b1}))) \\ r_{\text{overlap}} = \frac{A_{\text{overlap}}}{A_p + A_b - A_{\text{overlap}}} \end{cases} \tag{7}$$

To convert the overlap person as the rider of the bicycle, the overlap person should either overlap in most of the frames of the corresponding ID of the bicycle or have a long overlap duration that is beyond a threshold time. Figure 13 shows how the rider detection function works. The model has two branches according to the frame count of the bicycle's ID. The overlap time of the rider should be long to avoid noise. The reason that the lower branch of Figure 13 is included in the framework is the overlap person may vary because of the YOLOv7 detection limitation.
The Updating Function in Figure 12 can merge the BBox, update the bicycle label to the cyclist, and delete the frame of the person from the list when the person is detected as a rider. Finally, the updated JSON file could be created.

Table 1: Definitions of Variables in Equation 7

Variables	Definition
A_p	BBox area of person
A_b	BBox area of bicycle
$A_{overlap}$	BBox overlap area of both person and bicycle
x_{p1}, y_{p1}	Top left coordinate of BBox of person
x_{p2}, y_{p2}	Bottom right coordinate of BBox of person
x_{b1}, y_{b1}	Top left coordinate of BBox of bicycle
x_{b2}, y_{b2}	Bottom right coordinate of BBox of bicycle
r_{overlap}	Rate between overlap area and overall area of both person and bicycle

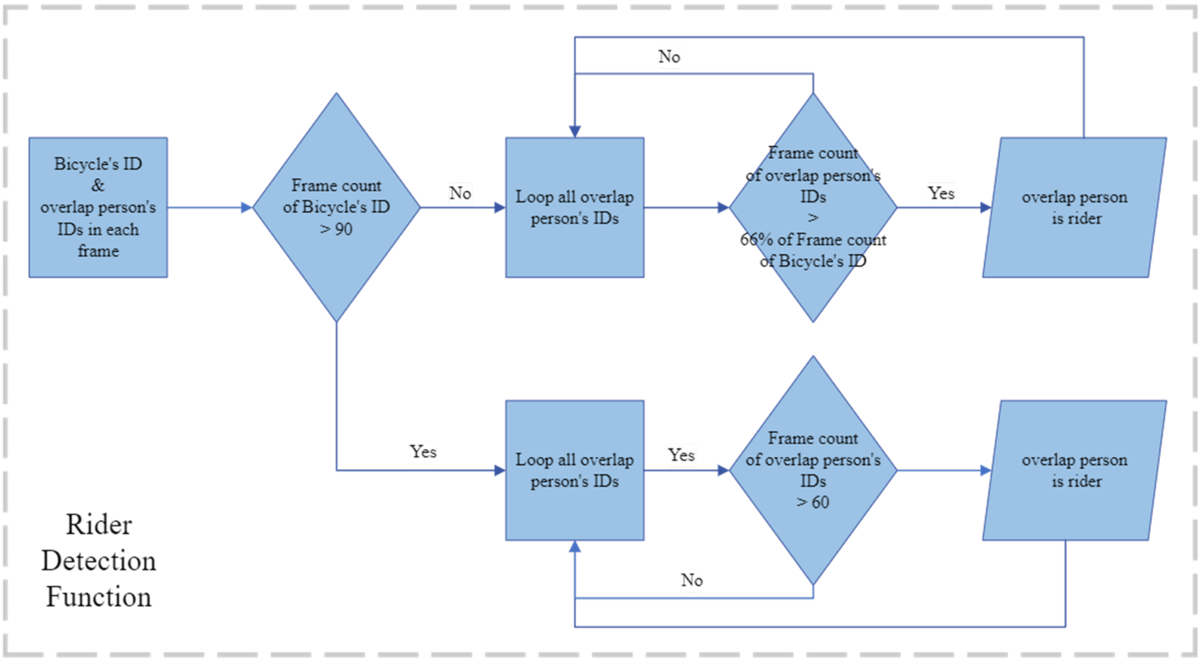

Bicycle's ID & overlap person's IDs in each frame
Frame count of Bicycle's ID > 90
No
Loop all overlap person's IDs
Frame count of overlap person's IDs > 66% of Frame count of Bicycle's ID
No
Yes
overlap person is rider
Yes
Loop all overlap person's IDs
Yes
Frame count of overlap person's IDs > 60
No
overlap person is rider
Rider Detection Function

Figure 13: A schematic drawing of the framework of the rider detection function

3.3 Object position extraction

The ground truth for the position used as input to the machine learning algorithms was obtained from a GPS that was mounted in the center of a car and a LiDAR mounted on the e-scooter. The position data for the car was obtained from the car's GPS, while the data for the e-scooter and bike where obtained from the LiDAR mounted on the stationary e-scooter.

3.3.1 Position from GPS

In the GPS data, the following information was obtained:

- UTC timestamp
- Latitude and longitude in degrees
- Heading angle in GPS reference system
- The 2-D velocity

To simplify the data analysis the latitude and longitude coordinates were converted to the WGS84 reference system and then rotated by the geographical angle (40.333 degrees) of the airfield. Since the coordinates of the e-scooter is known this point was set as origin (0,0) and the coordinates from the car could now be expressed as the position in x and y from this new origin. This means that the coordinates after these conversions represent the distance to the center of the car from the e-scooter. An example of this is plotted below:

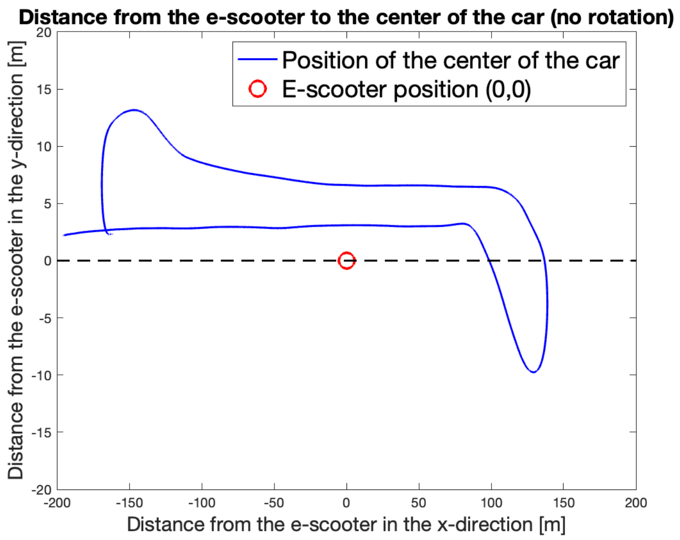
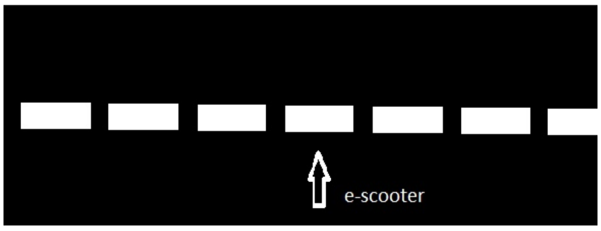
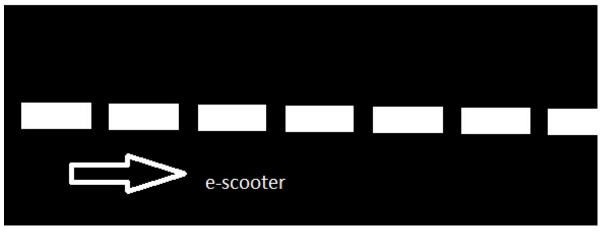


Figure 14: GPS data which shows the distance between the e-scooter to the center of the car during one test.

The e-scooter was oriented at different angles in relation to the runway, this is visualized in the figure below:



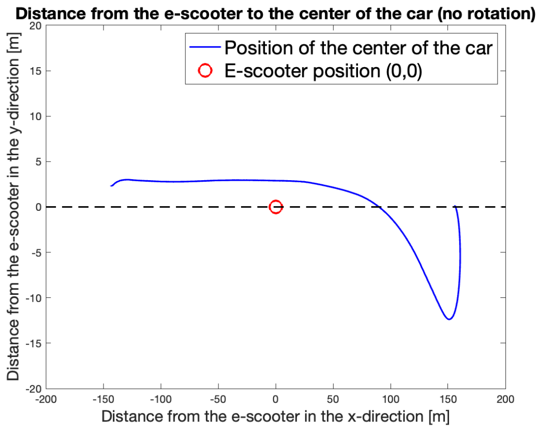
(a) Illustration of 0° rotation of the e-scooter relative to the runway.



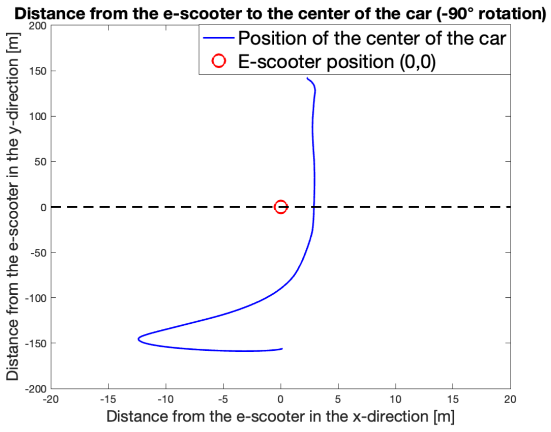
(b) Illustration of -90° rotation of the e-scooter relative to the runway.

Figure 15: Illustration of different rotations of the e-scooter relative to the runway.

There were also cases where the e-scooter was rotated -45° and 45° relative to the runway. Since the e-scooter was oriented with different angles to the runway it has to be accounted for to ensure that the coordinate system of the car would match the camera´s system. This was done by applying a rotation matrix on the x and y values, this is illustrated in the figures below:



(a) This example shows the GPS data before rotated by the e-scooter's angle to the runway.



(b) This example shows the GPS data after the rotation matrix is applied.

Figure 16: The two plots show first the GPS data before taking the e-scooter's angle relative to the runway into account. After the rotation matrix was applied the second plot was obtained which gives the correct orientation.

The information about the car's position cannot be used directly since the model is ideally based on the distance to the closest point of the object that's being tracked rather than the geometrical center of the car where the GPS was mounted. In other words, the car's position is actually the geometrical center's position of the car. It is easy to imagine a scenario where the geometrical center is still a distance to the e-scooter while some part of the car has already touched the e-scooter due to the car's dimensions. Therefore, it is necessary to convert the position of the geometrical point to the position of the closest point of the car. This problem was solved by first considering the dimensions of the car. Because the car is not driving in a straight line the heading angle of the car will affect which point of the car that will be the closest. This was solved by applying a rotation matrix with the same angle as the heading angle of the car. Then, the closest point of the car to the e-scooter was calculated by placing 100 points on the sides, front, and rear of the car (400 total), which could then be used to calculate which point of the car was closest. These points' coordinates were then transformed to the polar system to find the polar radius which was then used to find the closest point. This can be illustrated by the following schematic:

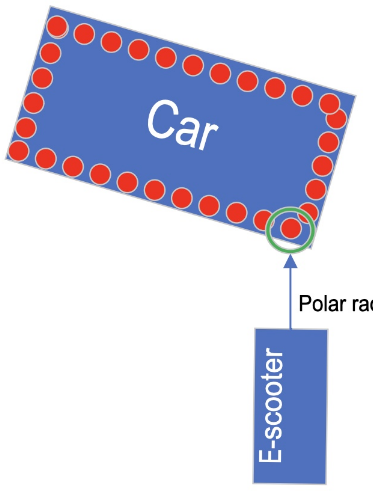


Figure 17: Schematic over how the closest point of the car to the e-scooter was calculated. The red dots represent the points placed along the front, side, and rear of the car. The polar radius for each point was calculated and the smallest radius was identified and saved. The green circle shows the point of the car which is closest to the e-scooter.

By doing this, the plot of the closest point of the car to the e-scooter was obtained as shown in Figure 18:

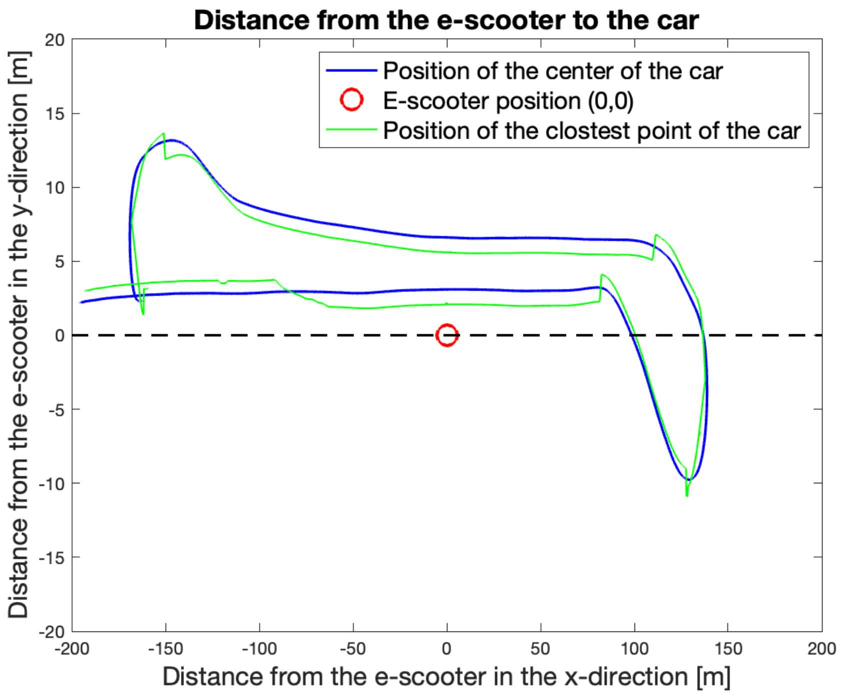
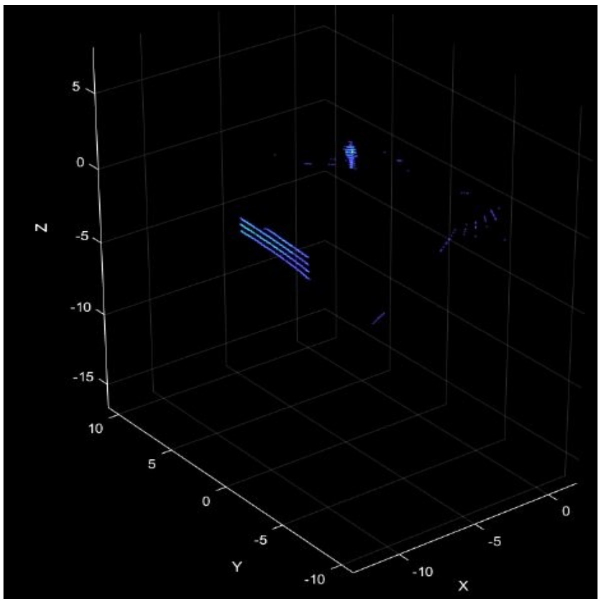


Figure 18: Based on the same data as Figure 14 with the addition of the position of the closest point of the car to the e-scooter.

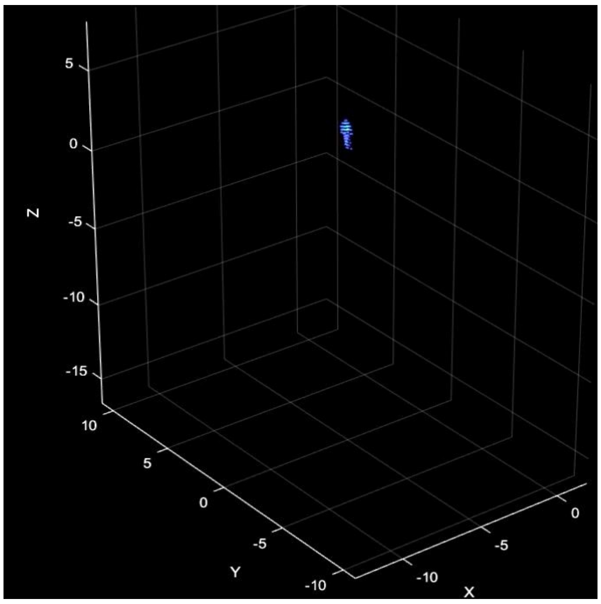
The radius, angle, and timestamps were saved to a CSV file, which was used to be used to generate training data for the position estimation model. During the transition of closest point from one side of the car to the other a sudden fluctuation is observed causing the jump in the green line as shown in Figure 18.

3.3.2 Position from LiDAR

The position identification from LiDAR is based on the work by Bharti [6]. This uses DBSCAN to process the LiDAR point cloud map. In the point cloud map, the pedestrian, the e-scooter rider, and the cyclist are tracked. The utilization of the Unix epoch time from the first frame in VeloView serves as a standardized and synchronized time reference for the script. This practice enhances consistency, ensures accurate alignment of data, and facilitates the proper initialization of the processing environment. To achieve this, the initial step involved using VeloView to extract the Unix epoch time from the first frame, subsequently employing this time as the script's starting point. After the initial settings were completed, the object could be tracked, as shown in Figure 19a and Figure 19b below:



(a) Point cloud map before applying DBSCAN to remove noise.



(b) Tracked object point cloud map after applying DBSCAN.

Figure 19: The two plots show how to track traffic participants.

For different tracking objects, different X-axis and Y-axis ranges need to be set to ensure that the object could be continuously tracked and more data related to the object could be obtained. If the calculated distance between consecutive centroids exceeds this threshold, it suggests that the tracked object has undergone rapid movement within the specified time frame. It's important to note that certain objects may exhibit faster motion, leading to the adjustment of the threshold used to determine segment loss. Finally, by processing 22 LiDAR point cloud files, the X position, Y position, and angle of the tracked objects were saved in different CSV files, which were used for future training and trajectory prediction.
The polar coordinate system used to represent the position from the LiDAR is the following:

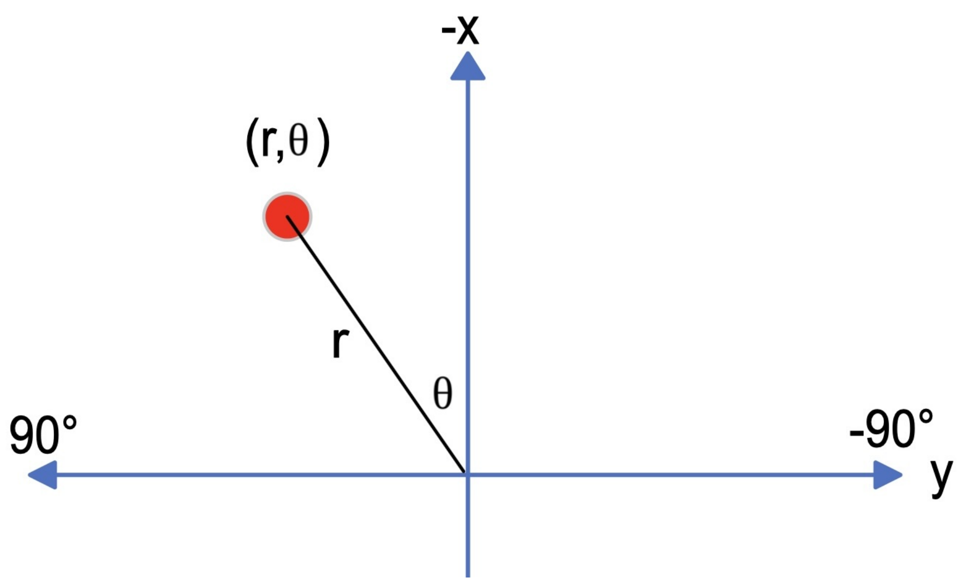


Figure 20: The polar coordinate system used to represent the position data from the LiDAR.

By using this coordinate system equation 2 has to be modified to:

$$x = -R \cdot \cos\theta \text{ and } y = -R \cdot \sin\theta \tag{8}$$

3.4 Training data

To form the training data for the position estimation, the position data from the LiDAR or GPS was used in combination with the JSON file from the video data. The first step was to synchronize the timestamps from the LiDAR and GPS to the timestamps of the frames from the camera. This was done by interpolating the different time vectors to get one that was related to the tracked IDs and the position to the object from the e-scooter. Then, for each frame that contains the object of interest, the position, and information about the BBox are saved to form the training data. Below is a table that explains the variables saved for each frame that contains the tracked object.

Table 2: Variables saved from the tracked object for each camera frame that contains the tract object. This data is saved and used as training data for the models.

X	x-coordinate of the object
Y	y-coordinate of the object
R	Radial distance to the object
θ	Angle to the object
x	Lower mid x of BBox
y	Lower y of BBox
h	Height of BBox
w	Width of BBox
y top	y value of top of BBox

This data was then inputted to different machine learning algorithms, to avoid overfitting, different combinations of inputs were tested to find the optimal set of inputs.

3.4.1 Pre-processed ground truth data

For the LiDAR and GPS data processing, the data from all the traffic participants was saved in different CSV files with the format of Table 3. The CSV files contain the UTC time, position of the X-axis, position of the Y-axis, and angle at each frame of the tract object, which was used as input for the model training.

Table 3: Example of the content of the LiDAR and GPS CSV files.

UTC	**X position**	**Y position**	**Distance**	**Degree**
2023-08-17 12:16:53.399	-17.0146	0.78729	17.0335	-2.649273
2023-08-17 12:16:53.510	-16.3035	0.84513	16.326	-2.967397
2023-08-17 12:16:53.620	-15.5918	0.8581	15.616	-3.150115
2023-08-17 12:16:53.731	-14.8761	0.86372	14.9017	-3.322912
2023-08-17 12:16:53.842	-14.1617	0.88015	14.1896	-3.556359
2023-08-17 12:16:53.953	-13.439	0.8971	13.4694	-3.819038
2023-08-17 12:16:54.064	-12.7228	1.0048	12.7629	-4.515852
2023-08-17 12:16:54.174	-12.0091	0.95308	12.048	-4.537653
...	...	...	...	...

From the test data set the following number of CSV files for the ground truth was obtained:

Table 4: Number of CSV files for the ground truth for the three different road users.

Road user	Sensor	Total number of data points
Cyclist	LiDAR	1764
Car	GPS	1387
E-scooterist	LiDAR	4374

3.4.2 Extracting bounding box information from camera frames

The video data processed by the object detection model was saved in the JSON file which contains every camera frame with the frame number, GPS time, and uptime of this frame, as well as the BBox, object type, and ID of the target object appearing in this frame. By indexing the ID of the target object, the corresponding frame number, GPS time, uptime and information of the BBox could be extracted. In this step, the information of the BBox includes x and y coordinates of the left top point of the BBox and its height and width. The information of the BBox was used as the input features for the position estimation model and GPS time which indicates at what time the target object appears in the video was used to synchronize camera data and ground truth data.

3.4.3 Synchronizing LiDAR/GPS and camera data

The whole dataset for position estimation should be based on camera data which means each camera frame containing the target object should have corresponding ground truth. However, because the camera has a frame rate of 30 Hz while LiDAR and GPS have a frame rate of 10 Hz and 50 Hz respectively, the camera data needs to be synchronized with LiDAR/GPS. LiDAR and GPS are interpolated to match the timestamp of the camera frames. In terms of LiDAR, it has a lower sampling frequency than that of the camera, which means there are more camera frames between two continuous LiDAR frames. The interpolation was done on the current camera frame according to the two nearest LiDAR frames which were before and after the camera frame respectively. By doing the interpolation, the camera frame can have the corresponding positions X, and Y, as well as the distance and angle of the target object. The synchronization for GPS was done based on the same process.

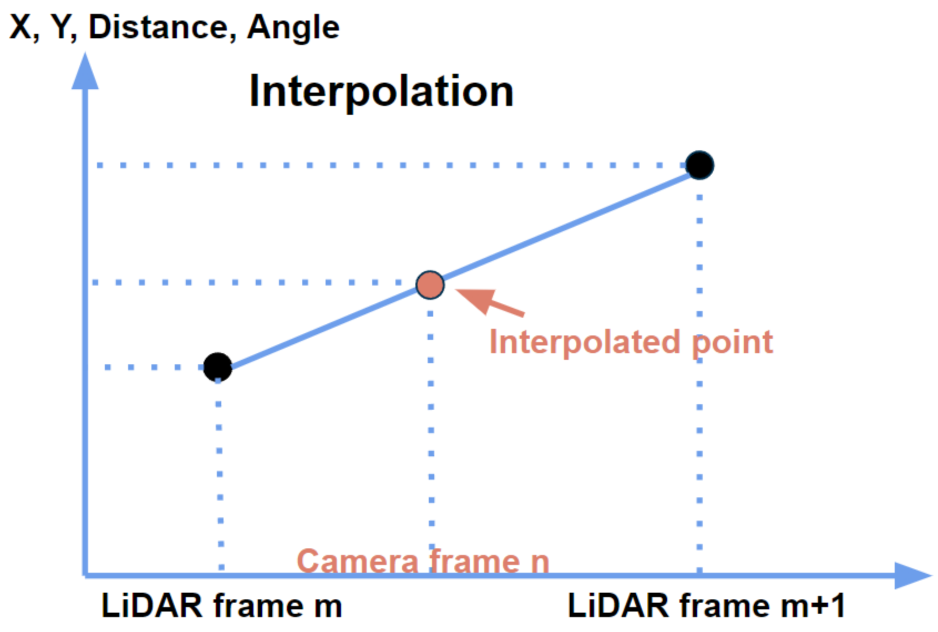


Figure 21: Interpolation to synchronize camera and LiDAR.

3.5 Data post-processing

To both improve the model's estimation data quality and estimate the kinematic information, an outlier removing function based on DBSCAN and a Rauch–Tung–Striebel (RTS) smoother could be applied to the raw output of the position estimation algorithm.

3.5.1 Trajectory extraction

The input of the outlier removing function and the RTS smoother is the trajectory of the object, which is time series data describing the object's position in corresponding time. Since the output JSON after the position estimation contains the objects' positions in each frame of the input video, rather than the trajectory of each object, a trajectory extraction process should be executed. However, the extracted trajectory may not time continue since the position information of one object is missing in some time (frame), but the following function expects a continuous trajectory as the input. As a result, the missed position information should be found and then complementary by linear interpolation to get a continuous trajectory.

3.5.2 Outlier Removing

The output of the position estimation may include some outliers (eg. Figure 22) that are far away from the fine trajectory points due to insufficient training data or some reasons.

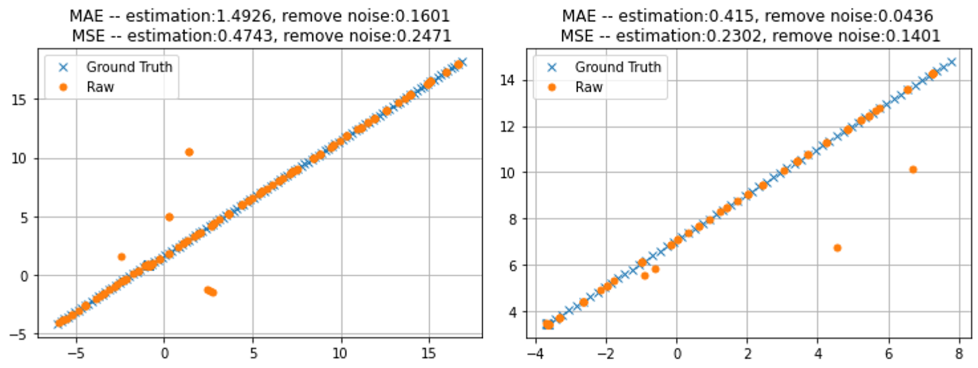


Figure 22: two trajectories examples that include a few outliers

Table 5: Selection Rule of min-samples

n (points of trajectory)	$n < 20$	$20 < n < 100$	$n \geq 100$
min-samples	2	$0.1n$	10

To find the outlier, DBSCAN, a cluster that can group the linked points and detect the outlier points, could be applied to the trajectory points of one object to detect the outlier points.

To imply the outlier removing function, two hyper-parameters, eps (the maximum distance between two samples, whose one sample is a point belong one neighbor point) and min-samples (minimum number of one neighbor), should be tuned. The eps value is set as 2 according to the motion limitation. The value of min-samples could be set based on the number of points of a trajectory (Table 5). To make this removing function figure out the outliers, which are a few points far away from the trajectory, in other words, which are a minority of all points, the value of min-samples needs to be set as a small value. However, the outlier points may grouped (more than 2 points) if the n is large, but the number of the grouped outliers will not be too many (less than 10 points). As a result, the value of min-samples needs to be changed according to n.

3.5.3 RTS smoothing

The state vector of the e-scooter kinematics information includes the lateral position $x_{k+1}^{p_x}$, the longitudinal position $x_{k+1}^{p_y}$, the lateral velocity $x_{k+1}^{v_x}$, and the longitudinal velocity $x_{k+1}^{v_y}$. The motion model is shown in equation (9), where the left 4×4 matrix is the transition matrix, and the covariance of the motion noise $\boldsymbol{q}_k$ which follows a normal distribution is the transition covariance. The parameter T is the time interval between two frames. Since the frame-per-second of the video is 30, the time interval should be set as $T = 1/30$. The motion model describes a constant velocity process, which means the position in the next state is derived from the position and velocity of the previous state, and the velocity in the next state is almost the same as the previous state, that is, the velocity in the next state is equal to the velocity in the previous state add a zero-mean Gaussian noise (derived from Equation 10).

$$\begin{bmatrix} x_{k+1}^{p_x} \\ x_{k+1}^{p_y} \\ x_{k+1}^{v_x} \\ x_{k+1}^{v_y} \end{bmatrix} = \begin{bmatrix} 1 & 0 & T & 0 \\ 0 & 1 & 0 & T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_k^{p_x} \\ x_k^{p_y} \\ x_k^{v_x} \\ x_k^{v_y} \end{bmatrix} + \boldsymbol{q}_k, \boldsymbol{q}_k \sim \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0.1 \end{bmatrix}\right) \tag{9}$$

$$\begin{bmatrix} \dot{x}_{p_x} \\ \dot{x}_{p_y} \\ \dot{x}_{v_x} \\ \dot{x}_{v_y} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_{p_x} \\ x_{p_y} \\ x_{v_x} \\ x_{v_y} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \sigma_{v_x}^2 \\ \sigma_{v_y}^2 \end{bmatrix}$$
$$\Rightarrow \boldsymbol{q}_k = T \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \sigma_{v_x}^2 & 0 \\ 0 & \sigma_{v_y}^2 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & T\sigma_{v_x}^2 & 0 \\ 0 & 0 & 0 & T\sigma_{v_y}^2 \end{bmatrix} \tag{10}$$

The measurement model is shown in equation (11), where the left 2×4 matrix is the observation matrix, and the covariance of the measurement noise $\boldsymbol{r}_k$ which follows a normal distribution is the observation covariance. Both the transition covariance and the observation covariance can be tuned to optimize the performance of RTS on the training video. The variance in the measurement model is bigger than the motion model since the measurement, which is the output of the position estimation model, has some noise that makes the trajectory not smooth.

$$\begin{bmatrix} y_k^{p_x} \\ y_k^{p_y} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_k^{p_x} \\ x_k^{p_y} \\ x_k^{v_x} \\ x_k^{v_y} \end{bmatrix} + \boldsymbol{r}_k, \boldsymbol{r}_k \sim \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}\right) \tag{11}$$

The position (both lateral and longitudinal) in the initial state is set as the same as the measurement at the beginning, and the velocity in the initial state is derived from the first and the second position measurements as a constant velocity process. The initial state covariance could be set as $\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 9 & 0 \\ 0 & 0 & 0 & 9 \end{bmatrix}$, whose variance of velocity is relatively big because the derived velocity is not accuracy.

3.6 Evaluation methods

When evaluating the performance of a regression model, there are three common metrics that are typically used: mean squared error (MSE, Mean Squared Error), mean absolute error (MAE, Mean Absolute Error), and coefficient of determination (R^2, R-squared).

3.6.1 MSE

MSE is the average of the squared differences between predicted and actual values.

$$\mathrm{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{12}$$

where n is the number of samples, y_i is the actual value, and $\hat{y}_i$ is the corresponding predicted value. The lower the MSE, the better the prediction performance of the model.

3.6.2 MAE

MAE is the average of the absolute differences between predicted and actual values.

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{13}$$

where n is the number of samples, y_i is the actual value, and $\hat{y}_i$ is the corresponding predicted value. Also, the lower the MSE, the better the prediction performance of the model.

3.6.3 R-squared

R-squared measures the proportion of the variance in the dependent variable that is predictable from the independent variable(s). It ranges from 0 to 1, where a higher value indicates a better explanatory power of the model.

$$\text{R}^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2} \tag{14}$$

where n is the number of samples, y_i is the actual value, $\hat{y}_i$ is the corresponding predicted value, and $\bar{y}$ is the mean of the actual values.

4 Results

In this chapter, the results are presented and visualized.

4.1 Object detection model

4.1.1 Object detection model for e-scooterist

Firstly, to test the performance of ICM, the test image set should be built. To build the test image set, all images that are categorized as the person from the test videos need to be extracted, and then need to be manually classified these images into pedestrian and e-scooterist. The Receiver operating characteristic (ROC) curve (Figure 23) and the proportion distribution curve (Figure 24) of the ICM could be then derived.

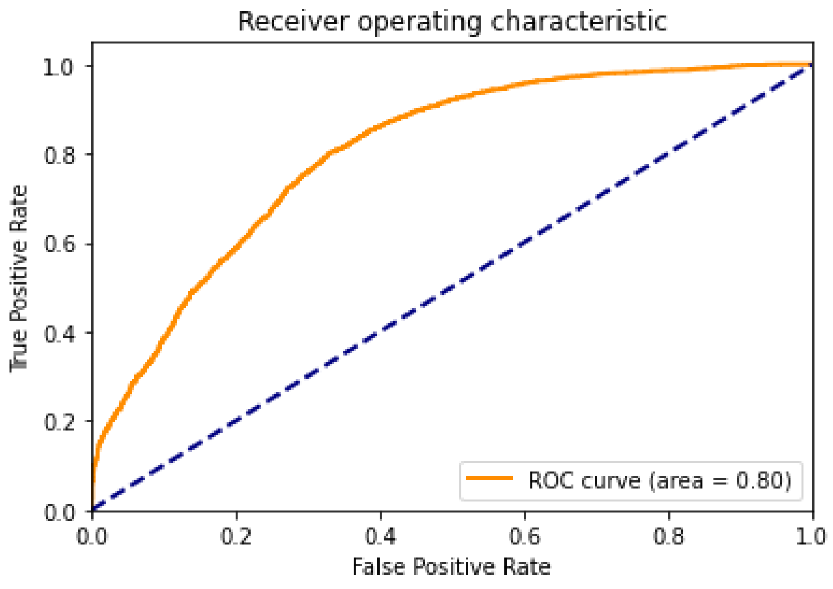


Figure 23: ROC curve of ICM

Figure 24 shows that ICM can distinguish the pedestrian well, but not good in distinguishing the e-scooterist. The threshold (sigmoid score to make classification) of the ICM should be optimized to improve the performance of ICM. The threshold is set as 0.7 to make the model have a high specificity to distinguish the pedestrian and make the sensitivity not low in distinguishing the e-scooterist. The confusion matrix of the selected threshold is shown in Table 6.

Table 6: Confusion matrix of e-scooterist detection (based on the image)

Actual \Predicted	Positive	Negative
E-scooterists	1413	1358
Pedestrian	232	2508

By applying this ICM, the pipeline of PCM can be built, and the performance of PCM can be tested on the test videos. By manual observation, the confusion matrix of the e-scooterist detection could be derived (Table 7). As we can see in Table 7, unlike Table 6, the performance of PCM is good both in distinguishing e-scooterist and pedestrian which gives an accuracy of 90.48%.

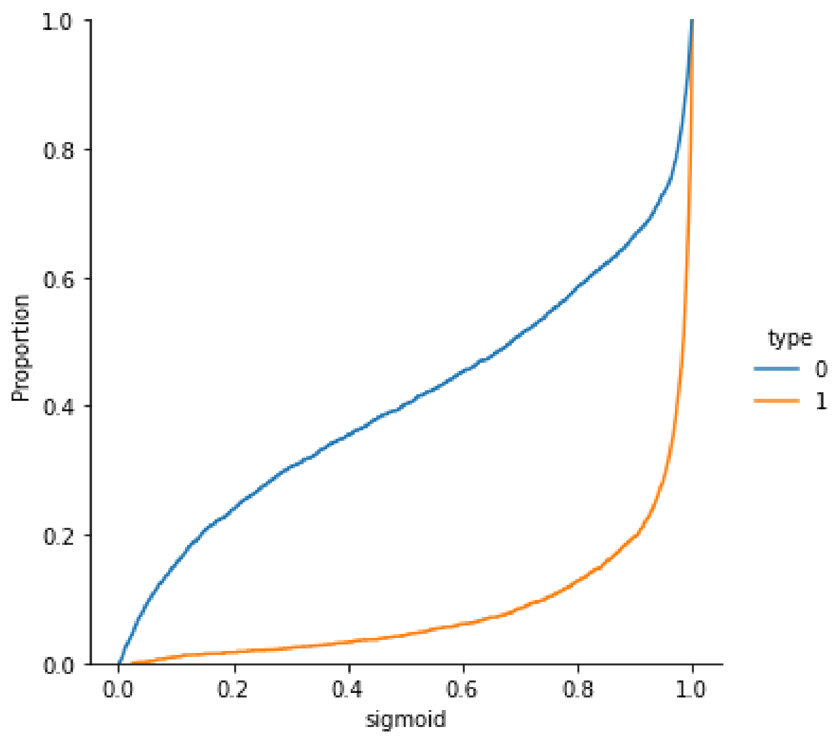


Figure 24: Proportion distribution curve of ICM (0: e-scooterist; 1: pedestrian)

Table 7: Confusion matrix of e-scooterist detection (based on the object)

Actual \Predicted	Positive	Negative
E-scooterists	23	1
Pedestrian	3	15

4.1.2 Object detection model for cyclists

The performance of CDM can be tested on the test videos. By manual observation, 26 IDs of cyclists can be successfully detected, and 2 IDs of cyclists cannot (Table 8). The reason that 2 cyclists cannot be detected is the riders (persons) haven't been detected by the YOLOv7 since they are far away from the ego then affects the CDM's function because the CDM updates the category based on rider detection. Overall, the CDM mostly can detect the cyclists on the test videos with an accuracy of 92.86%.

Table 8: Detection accuracy of CDM

	True	False
Cyclists	26	2

4.2 Position estimation model

To be able to obtain position information from naturalistic data, position estimation models are trained for each kind of road user using different machine learning algorithms like Linear Regression, Decision Tree Regressor, K-Nearest Neighbors Regressor, Random Forest Regressor, and Multi-layer Perceptron Regressor. The output of the model is the distance and angle of the object.

4.2.1 Position estimation model for cyclists

It can be concluded from Table 9 that KNeighborsRegressor has the highest R^2 score and meanwhile has the lowest error. Then, the RandomForestRegressor also has good performance in general. Figure 26 shows the actual position and predicted position estimated by the KNeighborsRegressor in Cartesian coordinate system, while Figure 27 shows it by polar system.

Table 9: Comparison of the performance of different regression algorithms.

Model	MSE	MAE	R^2 Score
LinearRegression	5.692	1.923	0.477
DecisionTreeRegressor	1.842	0.576	0.835
KNeighborsRegressor	0.649	0.268	0.944
RandomForestRegressor	0.908	0.560	0.918
MLPRegressor	2.437	1.162	0.781

Figure 27 and Figure 26 show the reference data used to test and evaluate the polar and cartesian models respectively. From the data, it is clear that the accuracy of cyclist detection decreases as the angle and distance increase.

Table 10: Performance of KNN using different parameters.

Parameters	MSE	MAE	R^2 Score
Low mid x,Low mid y	2.064	0.984	0.819
Low mid x,Low mid y,height, width	1.063	0.606	0.905
Low mid x,Low mid y,height	1.522	0.787	0.861
Low mid x,Low mid y, width	1.242	0.667	0.889

Table 11: Performance of Random Forest using different parameters.

Parameters	MSE	MAE	R^2 Score
Low mid x,Low mid y	2.112	0.947	0.813
Low mid x,Low mid y,height, width	0.942	0.577	0.917
Low mid x,Low mid y,height	1.277	0.722	0.886
Low mid x,Low mid y, width	1.242	0.667	0.889

In Table 10 and 11, two models with high accuracy were chosen for comparison. Change the features during training and combine them, and observe the R^2 value. When the features are selected as Low mid x,Low mid y,height, width, the model performs well.

Table 12: Performance of RF and KNN models for polar and Cartesian coordinates.

Algorithms	MSE	MAE	R^2 Score
RF Polar	0.979	0.639	0.892
RF Cartesian	1.049	0.687	0.854
KNN Polar	0.713	0.365	0.920
KNN Cartesian	0.731	0.402	0.913

It's clear from Table 12 that the R^2 score is the highest when using the KNN model and where the predicted values are the distance and angle in the polar coordinate system, which gives the best model performance.

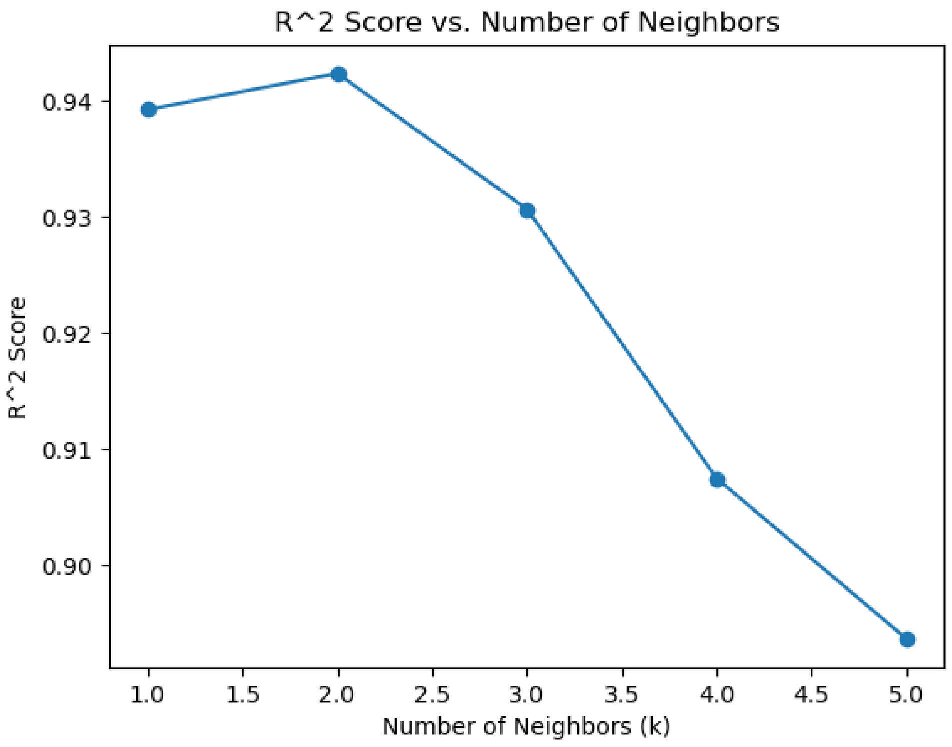


Figure 25: KNeighborsRegressor hyperparameter optimization. Where $k = 2$ gives the highest R^2 score.

By tuning the hyperparameters of the KNN model, as shown in Figure 25, the model performs the best when n_estimators is 2. Therefore, in the best model, n_estimators = 2 and R^2 score is 0.942.

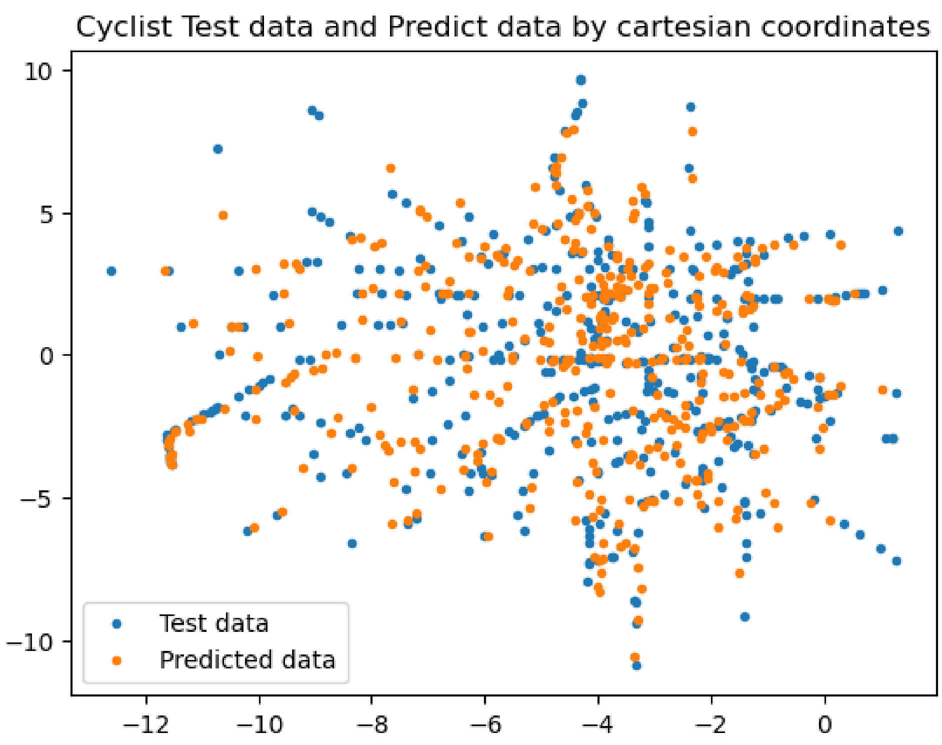


Figure 26: Cyclist test data and predicted data using Cartesian coordinates. Where the blue points correspond to the test data and the orange points correspond to the predicted points by the model.

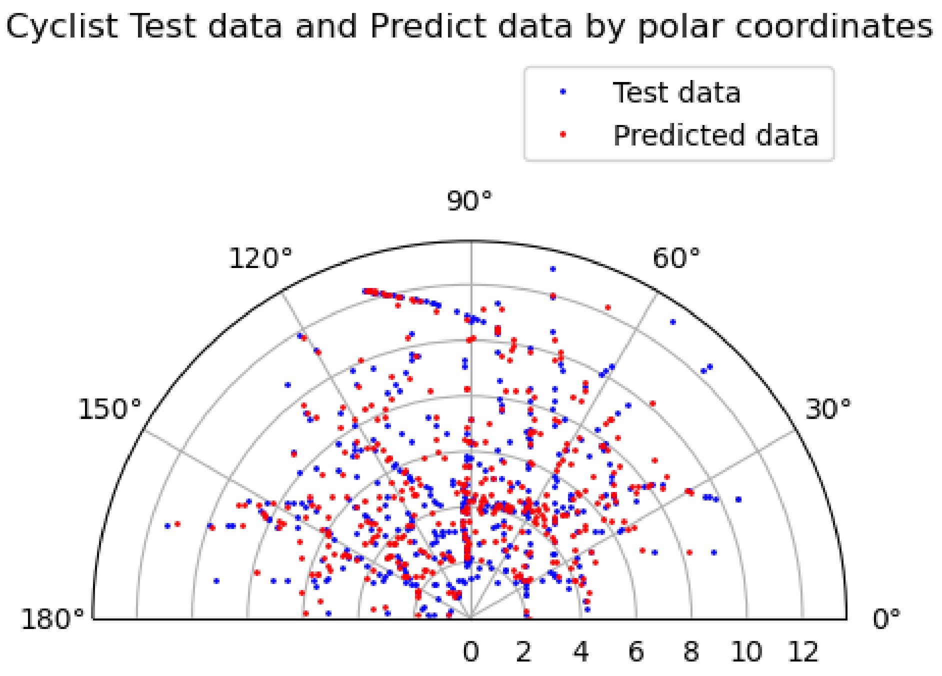


Figure 27: Cyclist test data and predicted data using polar coordinates. This plot shows the same data as in Figure 26 but represented in polar coordinates.

Plot the predicted data against the test data (shown in Figure 26 and Figure 27). In general, the model predictions are very close to the actual values.

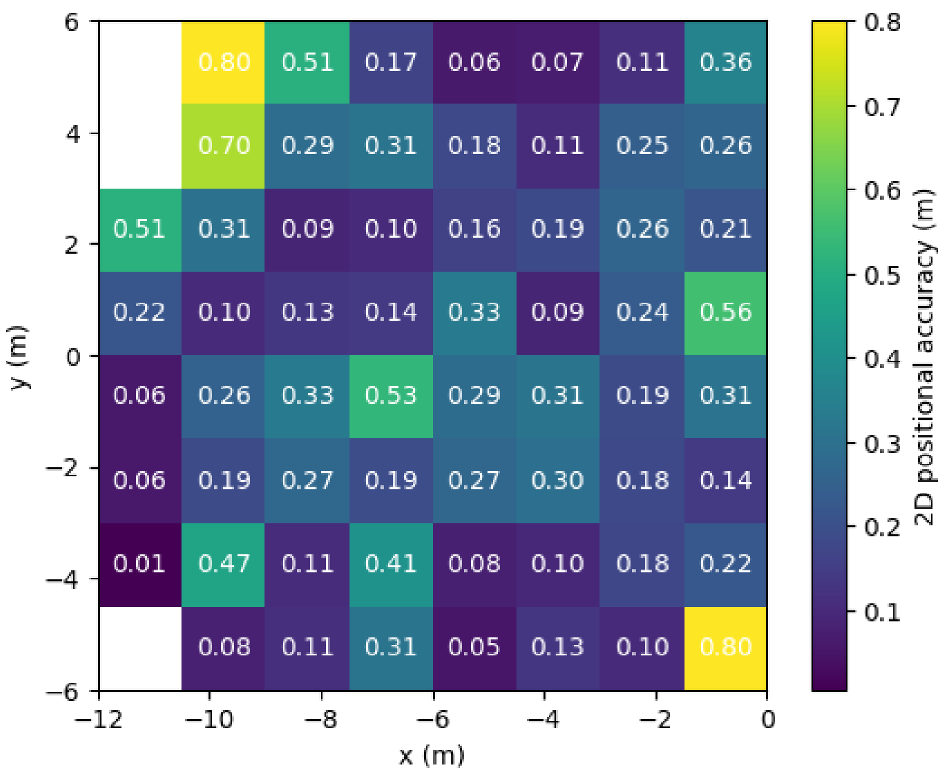


Figure 28: Cyclist model error heatmap in Cartesian coordinates. Which shows small errors, with a maximal error of 0.8 m.

While the cyclist model exhibits overall good performance (as shown in Figure 28, the prediction error is within 0.4m for most areas), certain challenges arise in predictions when the cyclist is positioned at a significant distance and angle (as shown in Figure 29 and Figure 32). Additionally, notable angular errors are observed in proximity to the scooter, indicating specific areas where the model's predictive accuracy may be improved

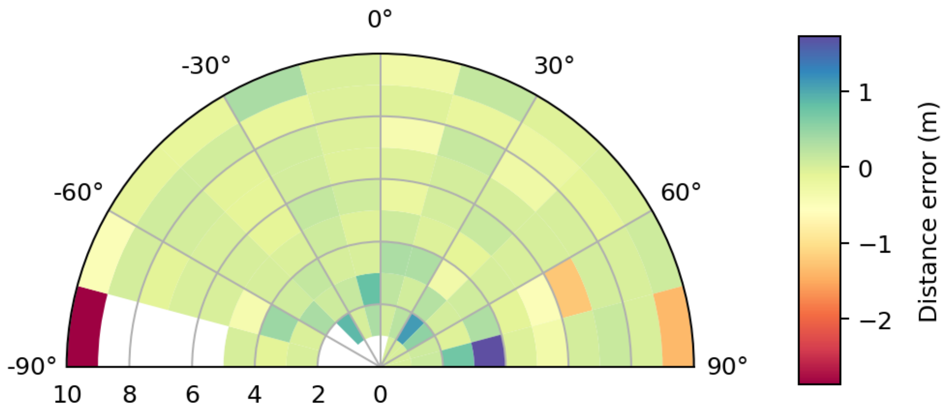


Figure 29: Cyclist model distance error heatmap when using polar coordinates. Notable, is the error when cyclist has a big angle to the e-scooter and is far away, this is where the biggest distance error occur in the predictions.

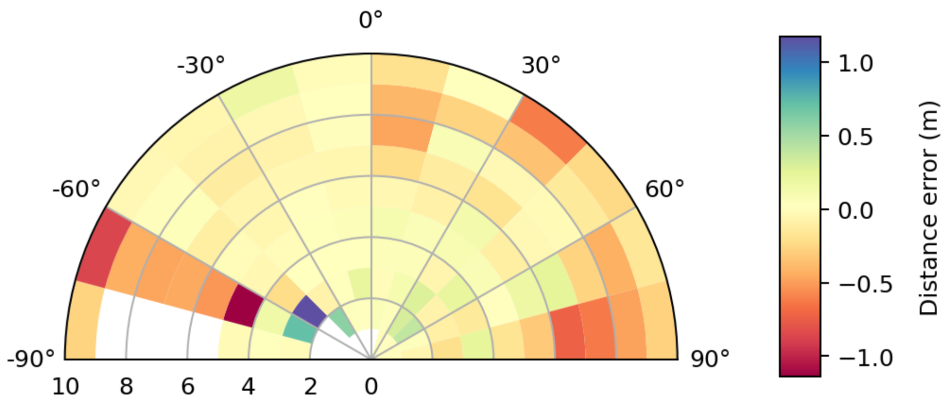
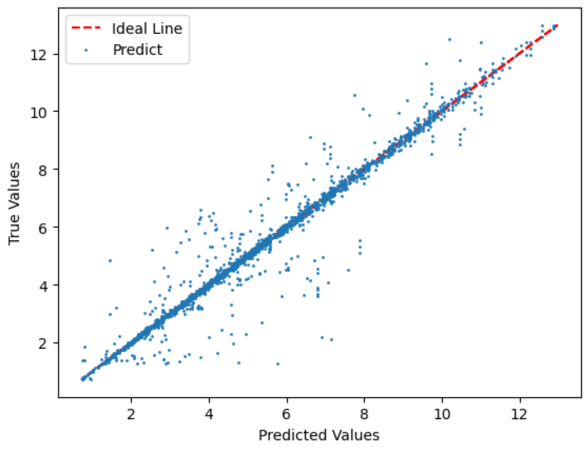


Figure 30: Cyclist model distance error heatmap by polar coordinates after DBSCAN is applied to the predictions done by the model.



(a) KNN model distance prediction before filter. (b) KNN model distance prediction after filter.

Figure 31: Pre-filter and post-filter K Neighbors Regressor (n = 2) normalised polar model distance prediction vs ground truths.

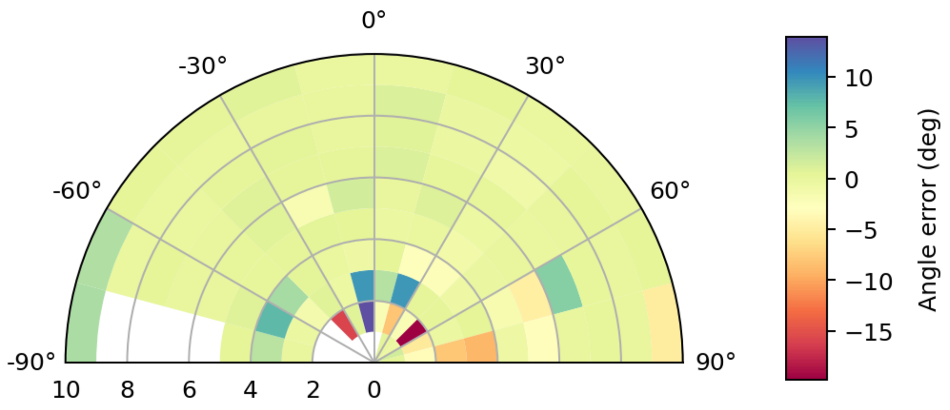


Figure 32: Cyclist angle error heatmap by polar coordinates

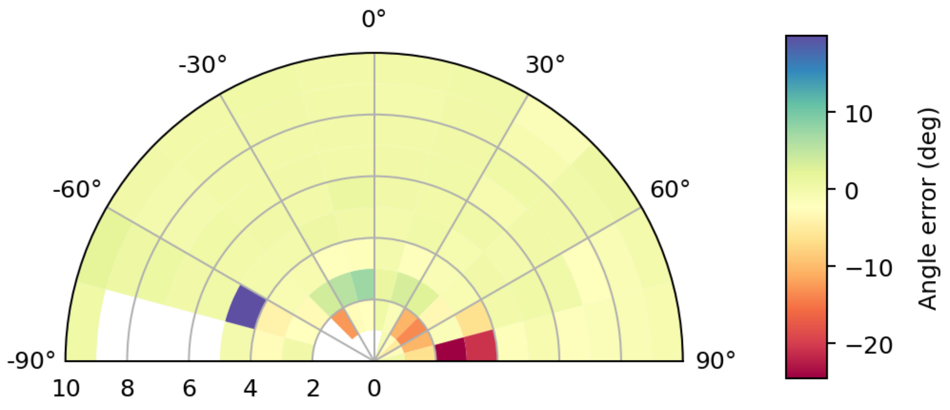
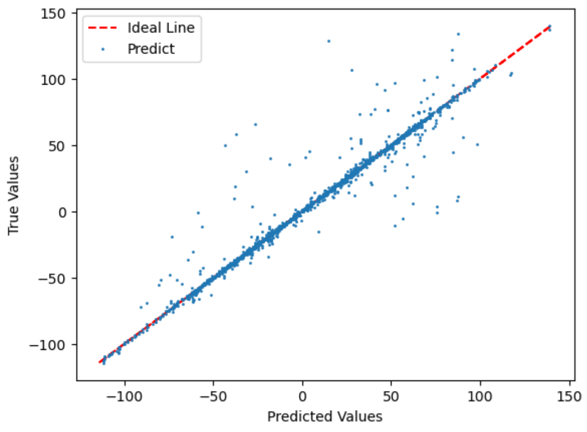
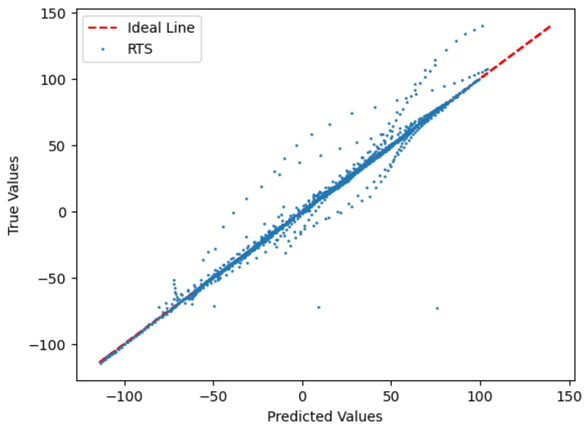


Figure 33: Cyclist angle error heatmap by polar coordinates after filter



(a) KNN model angle prediction before filtering.



(b) KNN model angle prediction after filtering is done.

Figure 34: Pre-filter and post-filter K Neighbors Regressor (n = 2) normalised polar model angle prediction vs ground truths.

After filtering and smoothing, the results are shown in Figure 30 and 33. Compared to the error thermogram before filtering, the error is significantly reduced. By plotting the regression curves (distance and angle) before and after filtering (Figure 31a and 31b, Figure 34a and 34b), we can see that the filtered model predicts better.

4.2.2 Position estimation model for e-scooter riders

In this case, KNeighborsRegressor still has the best performance according to Table 13. The R^2 score reaches 0.9875, meanwhile, the mean absolute error is only 0.2371. The RandomForestRegressor also has a similar performance.

Table 13: Performance of different regression algorithms.

Model	MSE	MAE	R^2 Score
LinearRegression	7.391	1.999	0.682
DecisionTreeRegressor	0.527	0.288	0.976
KNeighborsRegressor	0.268	0.237	0.987
RandomForestRegressor	0.312	0.280	0.986
MLPRegressor	1.336	0.705	0.935

From Figure 36 and Figure 37, it can be seen that the consistency between predicted and actual position is very high, which indicates that the model can make good predictions.

Table 14: Performance of KNN with different parameters.

Parameters	MSE	MAE	R^2 Score
Low mid x,Low mid y	0.611	0.463	0.975
Low mid x,Low mid y,height, width	0.342	0.297	0.984
Low mid x,Low mid y,height	0.432	0.367	0.982
Low mid x,Low mid y, width	0.413	0.345	0.982

Table 15: Performance of Random Forest with different parameters.

Parameters	MSE	MAE	R^2 Score
Low mid x,Low mid y	0.612	0.495	0.975
Low mid x,Low mid y,height, width	0.393	0.320	0.982
Low mid x,Low mid y,height	0.479	0.416	0.979
Low mid x,Low mid y, width	0.621	0.393	0.970

Table 16: Performance of RF and KNN models using polar and Cartesian coordinates.

Algorithms	MSE	MAE	R^2 Score
RF Polar	0.553	0.385	0.969
RF Cartesian	0.564	0.386	0.967
KNN Polar	0.386	0.294	0.978
KNN Cartesian	0.394	0.304	0.978

Similarly, it can be concluded from Table 16, 15 and 14 that the KNN model using the polar coordinate as output still has the best performance.
Next, the escooter rider data is processed in the same way as the cyclist data, the inputs are Low mid x,Low mid y,height, width, the KNN model is chosen for prediction, and the hyperparameters are adjusted to get the final model. Where $k = 2$ gives the highest R^2 score (best performance), as can be seen in figure 35. This gave a final R^2 score of 0.978.

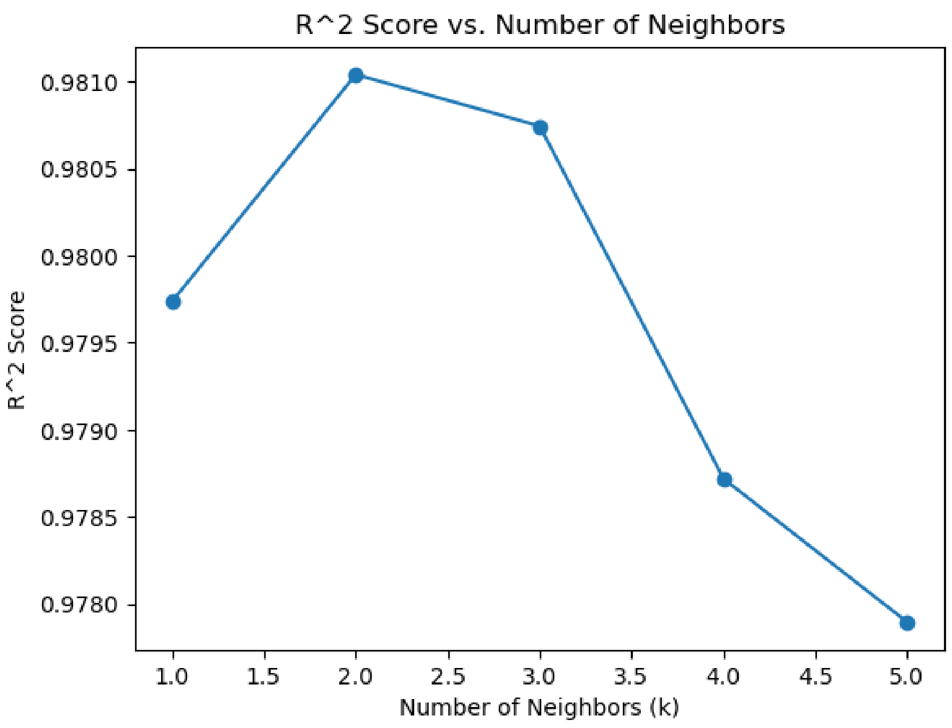


Figure 35: KNeighborsRegressor hyperparameter optimization, where $k = 2$ gives the best result.

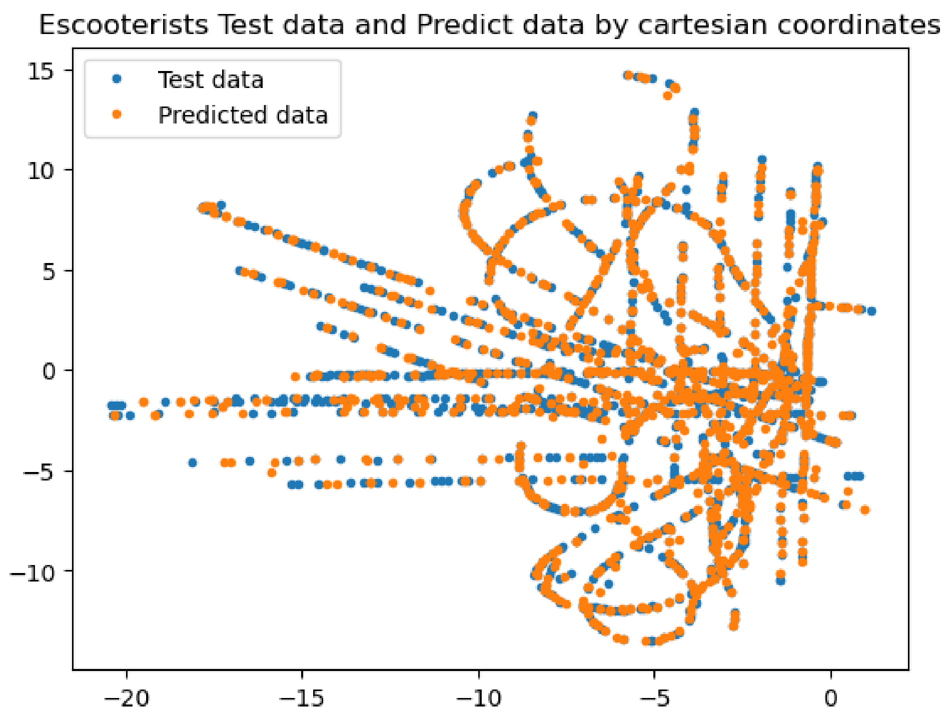


Figure 36: E-scooter Test data and Predict data by Cartesian coordinates.

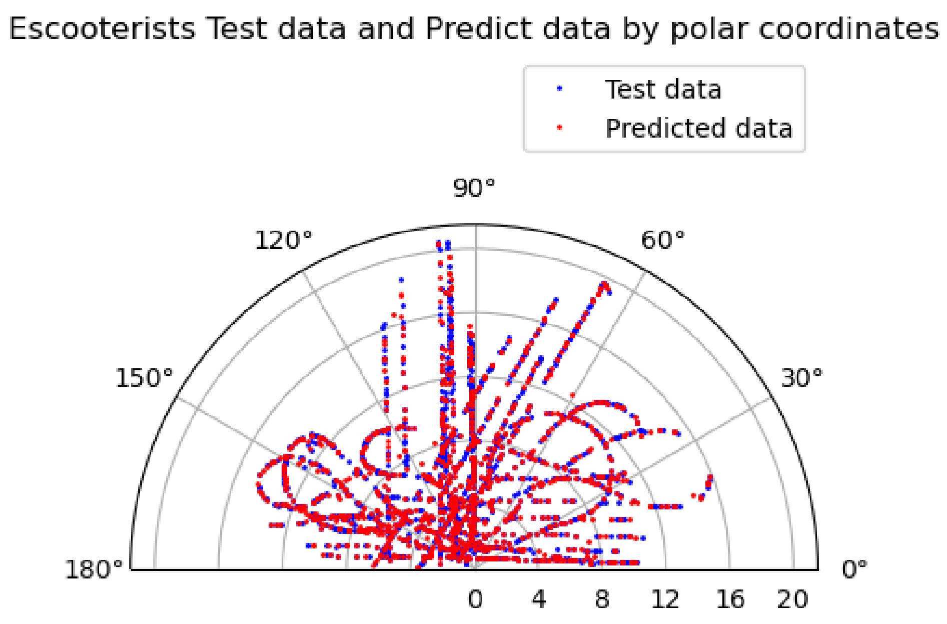


Figure 37: E-scooter Test data and Predict data by polar coordinates.

Plot the predicted data against the test data for escooterists (in Figure 36 and 37). Overall, the model predictions closely align with the actual values.

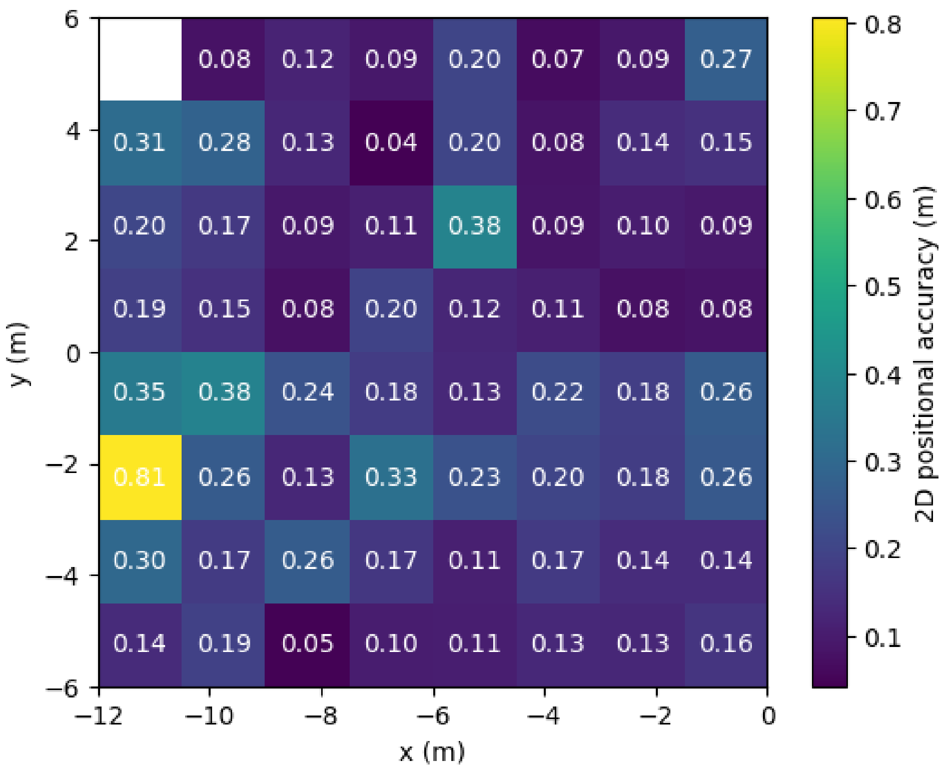


Figure 38: E-scooter error heatmap by Cartesian coordinates.

The analysis reveals negligible deviations in both distance(in Figure 39) and angular errors (in Figure 42) across the majority of areas. At the same time, the prediction errors of the distances were very small, mostly within 0.4m (in Figure 38). Notably, only a small fraction of regions exhibited angular errors surpassing 10 degrees.

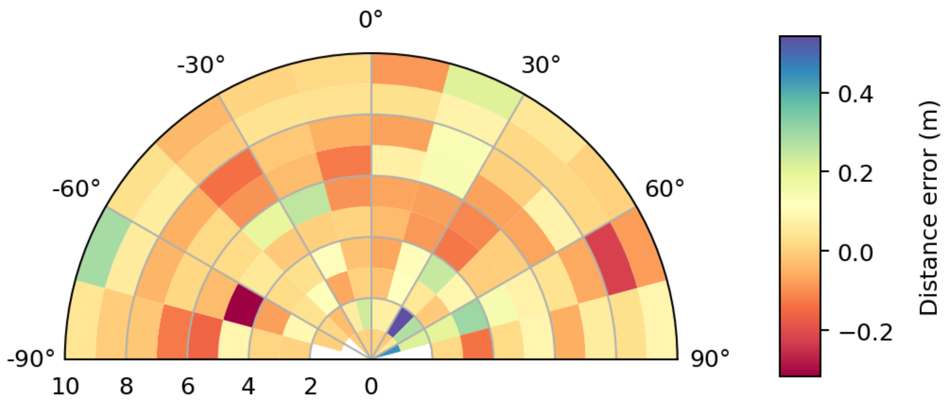


Figure 39: E-scooter distance error heatmap by Polar Coordinates.

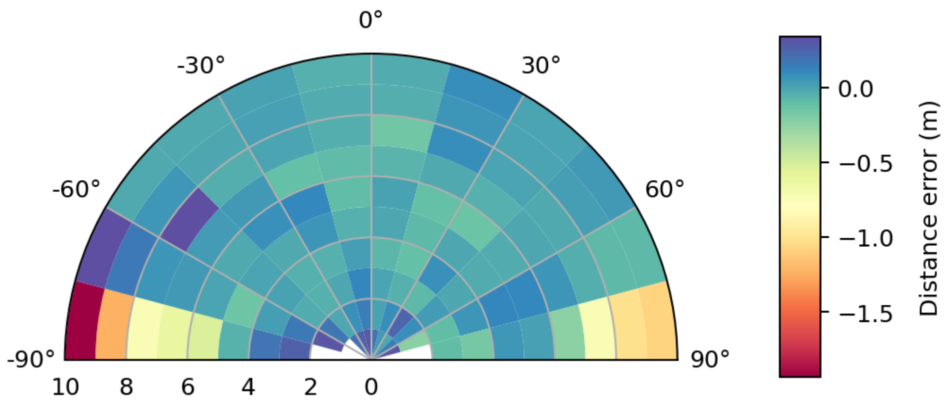
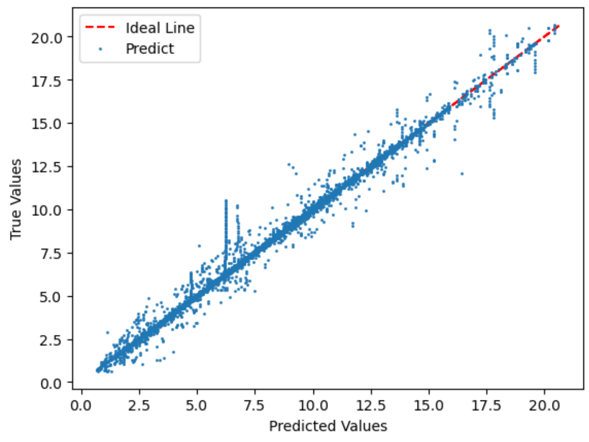
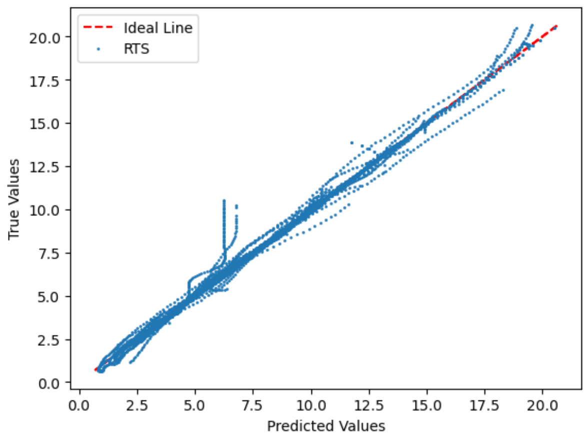


Figure 40: E-scooter distance error heatmap by polar coordinates after filter.





(a) KNN model distance prediction before filtering.

(b) KNN model distance prediction after filtering is done.

Figure 41: Pre-filter and post-filter K Neighbors Regressor (n = 2) normalised polar model distance prediction vs ground truths.

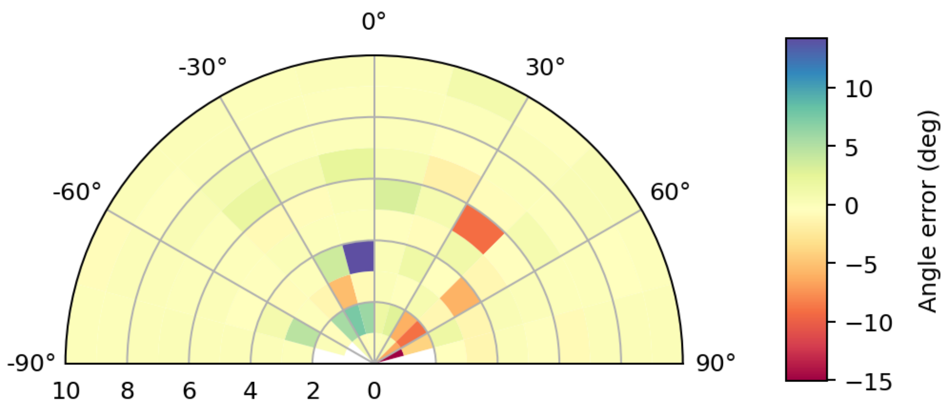


Figure 42: E-scooter angle error heatmap by polar coordinates.

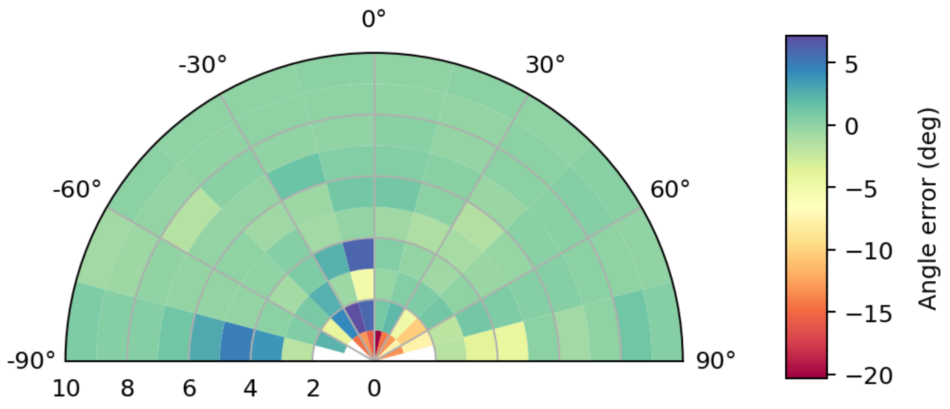
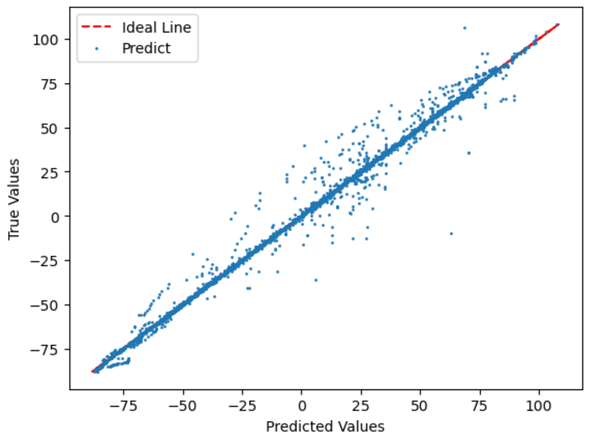
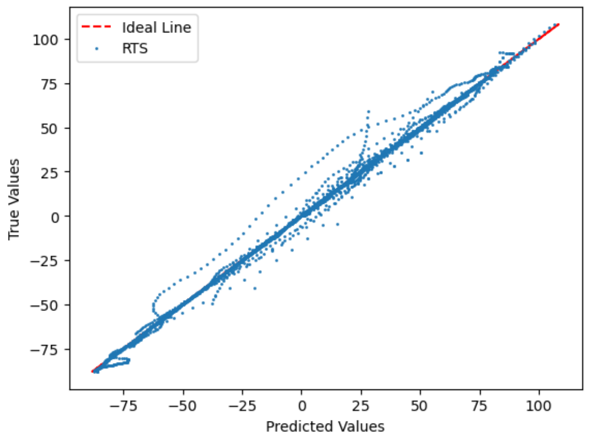


Figure 43: E-scooter angle error heatmap by polar coordinates after filter.



(a) KNN model angle prediction before filtering.



(b) KNN model angle prediction after filtering is done.

Figure 44: Pre-filter and post-filter K Neighbors Regressor (n = 2) normalised polar model angle prediction vs ground truths.

After filtering and smoothing, the prediction error results are shown in Figure 40 and 43. Compared to the error thermogram before filtering, the error is significantly reduced. By plotting the regression curves (distance and angle) before and after filtering (Figure 41a and 41b, Figure 44a and 44b), we can see that the filtered model predicts better.

4.2.3 Position estimation model for cars

It can be concluded from the previous steps that KNeighborsRegressor and RandomForestRegressor are optimal algorithms. So, the performance of these two algorithms is compared in this section.

Table 17: Performance of different regression algorithms.

Model	MSE	MAE	R^2 Score
KNeighborsRegressor	0.829	0.278	0.989
RandomForestRegressor	1.872	0.618	0.974

Table 18: Performance of KNN with different parameters.

Parameters	MSE	MAE	R^2 Score
Low mid x,Low mid y	4.866	0.770	0.933
Low mid x,Low mid y,height, width	1.566	0.326	0.979
Low mid x,Low mid y,height	2.339	0.470	0.968
Low mid x,Low mid y, width	1.504	0.341	0.980

Table 19: Performance of Random Forest with different parameters.

Parameters	MSE	MAE	R^2 Score
Low mid x,Low mid y	4.522	1.017	0.938
Low mid x,Low mid y,height, width	2.223	0.655	0.970
Low mid x,Low mid y,height	2.481	0.730	0.966
Low mid x,Low mid y, width	2.706	0.724	0.963

Table 20: Performance of RF and KNN models using polar and Cartesian coordinates.

Algorithms	MSE	MAE	R^2 Score
RF Polar	3.291	0.843	0.955
RF Cartesian	3.475	0.883	0.951
KNN Polar	2.804	0.558	0.963
KNN Cartesian	2.845	0.563	0.961

From Table 17, 18, 19 and 20, it can be seen that the KNN model using polar coordinate as output has the same R2 score when it uses Cartesian coordinate. However, the MSE and MAE values are slightly lower when polar coordinate is used. Therefore, KNN model using polar coordinate as output has the best performance. Using 2 as the number of neighbors (k=2) gives the highest result for the car's models, see figure 45.

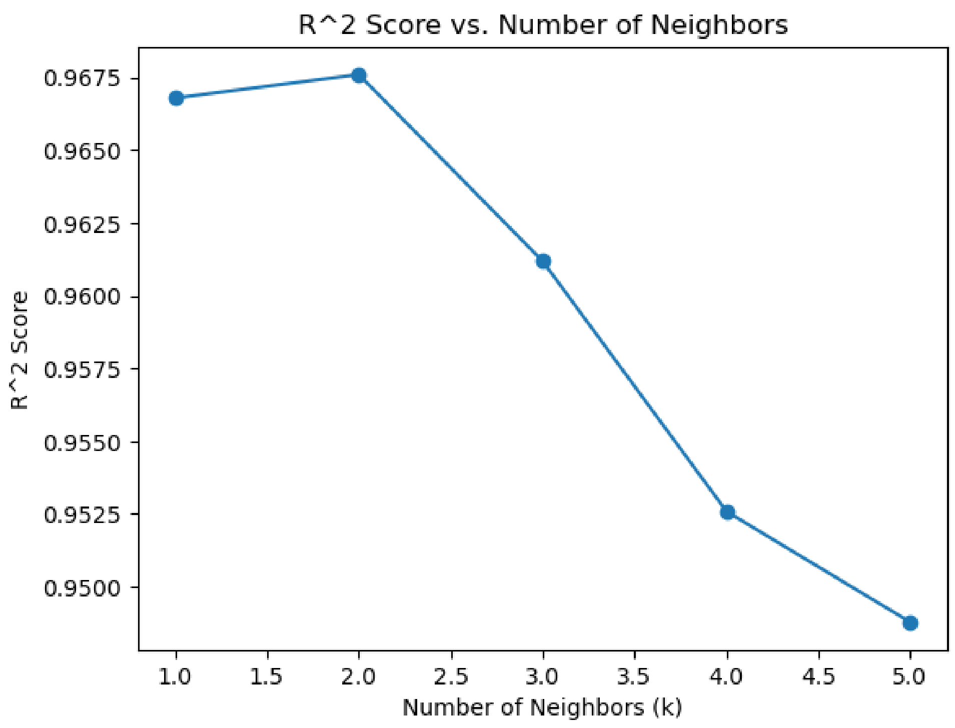


Figure 45: KNeighborsRegressor hyperparameter optimization. From the figure it can be concluded that using $k = 2$, number of neighbors, gives the highest R^2 score.

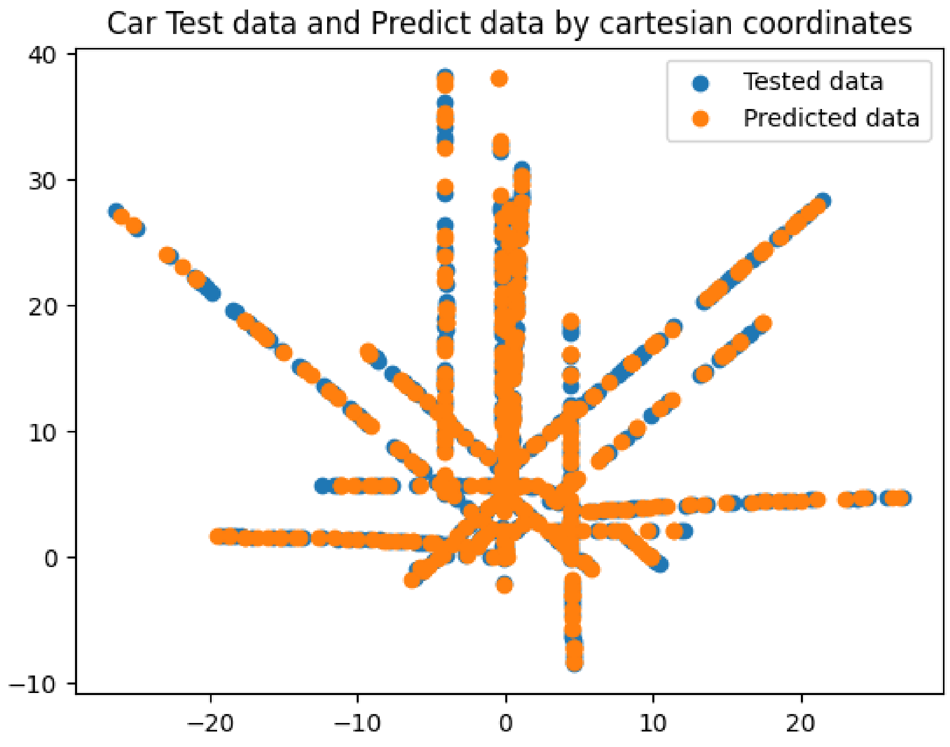


Figure 46: Car test data and predicted data by Cartesian coordinates.

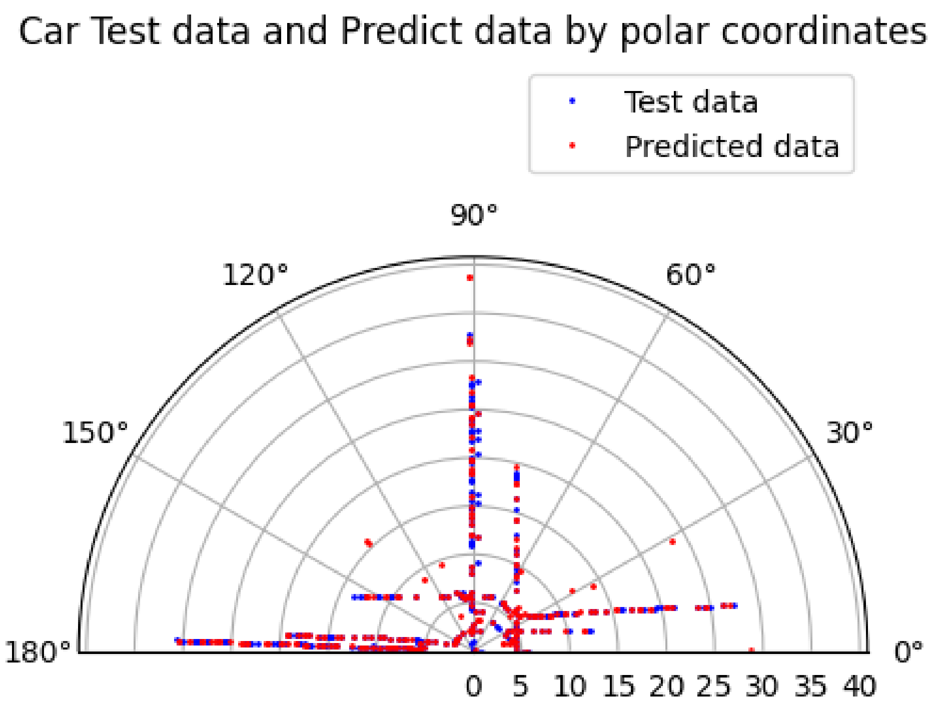


Figure 47: Car test data and predicted data by polar coordinates.

By comparing the predicted values in Cartesian coordinate (Figure 46) and polar coordinate (Figure 47), the predicted and true values are almost matched, which means that the prediction performance of the model is good.

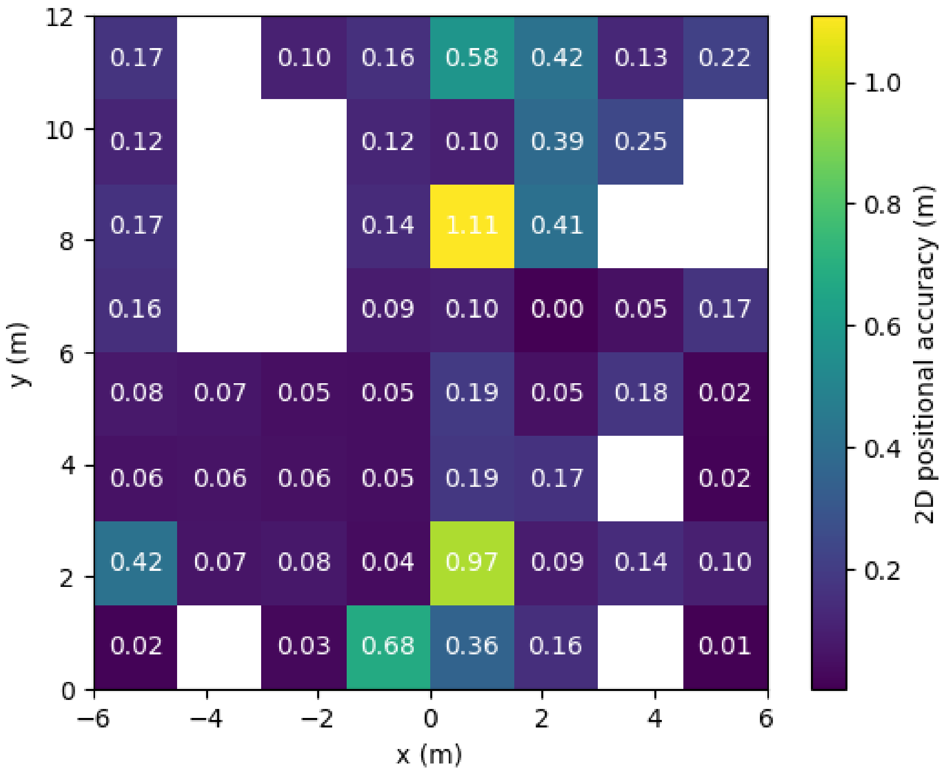


Figure 48: Car error heatmap by Cartesian coordinates.

Utilizing our model predictions, we can effectively constrain the distance prediction error of the vehicle to within 2 meters, from Figure 48. Certain regions may exhibit white spaces (in Figure 48, 49 and 52), indicative of missing data, a consequence of the constraints imposed by our limited test dataset.

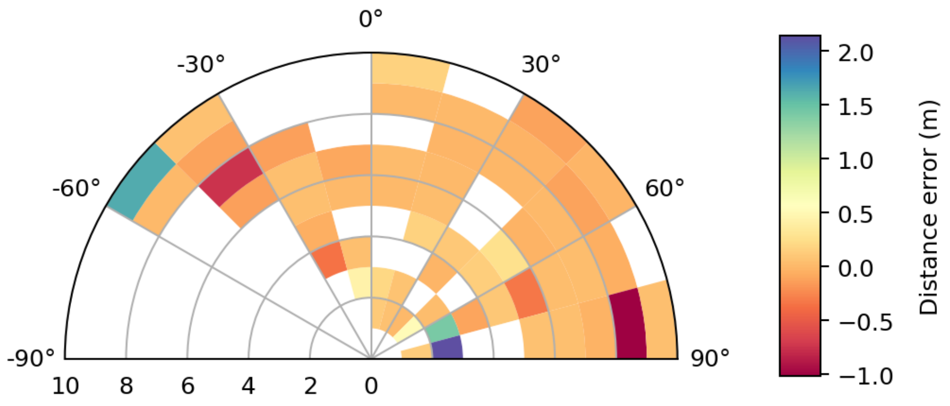


Figure 49: Car distance error heatmap by polar coordinates.

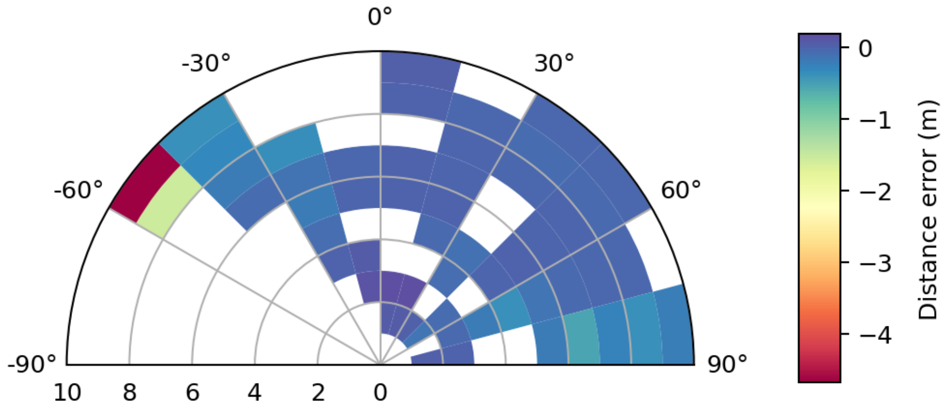
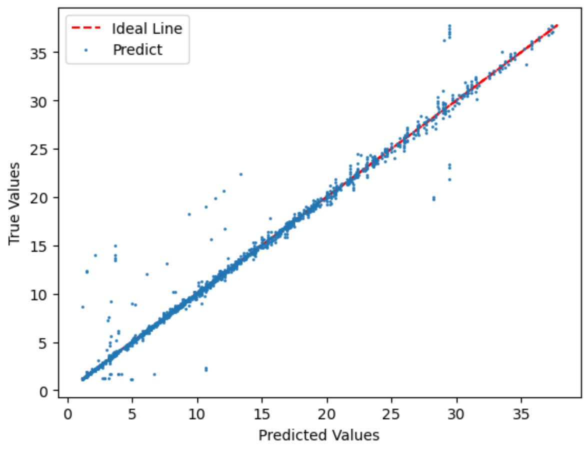
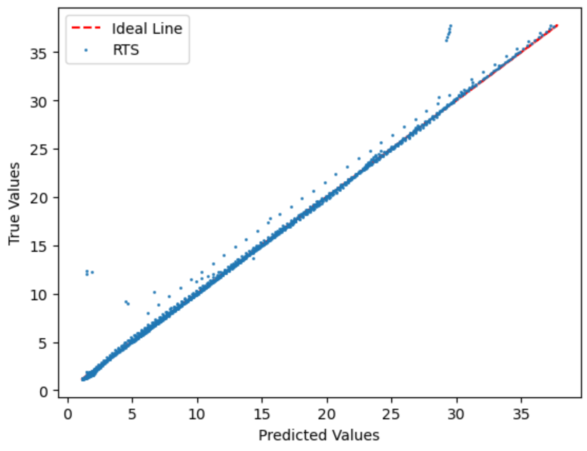


Figure 50: Car distance error heatmap by polar coordinates after filter.



(a) KNN model distance prediction before filtering.



(b) KNN model distance prediction after filtering is done.

Figure 51: Pre-filter and post-filter K Neighbors Regressor (n = 2) normalised polar model distance prediction vs ground truths.

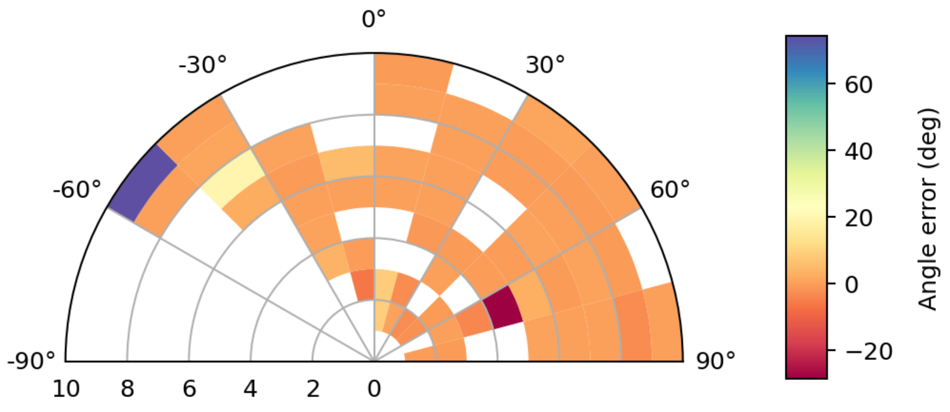


Figure 52: Car angle error by polar coordinates.

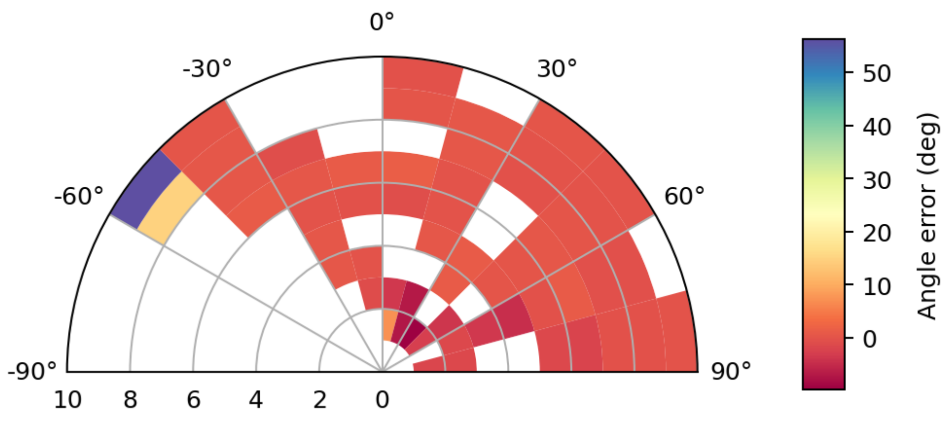
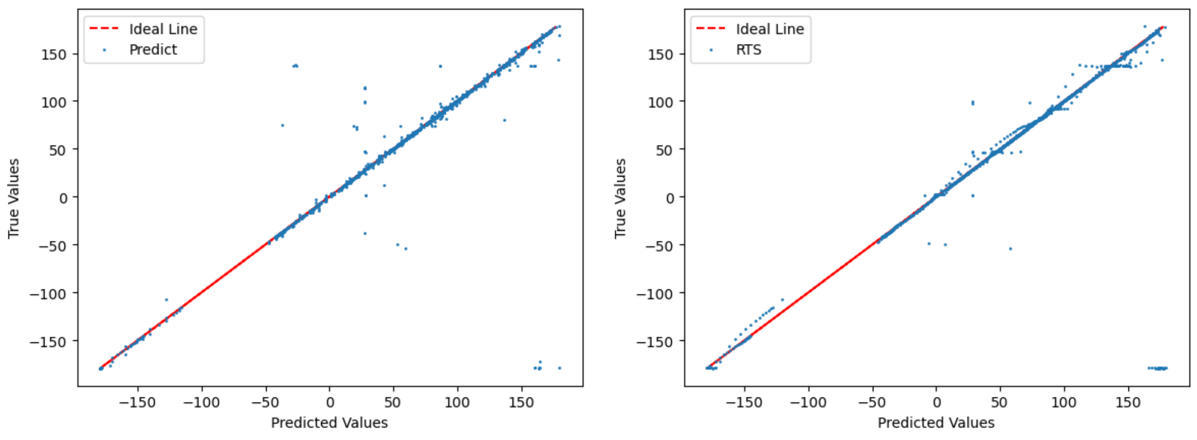


Figure 53: Car angle error by polar coordinates after filter.



(a) KNN model distance prediction before filtering.

(b) KNN model distance prediction after filtering is done.

Figure 54: Pre-filter and post-filter K Neighbors Regressor (n = 2) normalised polar model distance prediction vs ground truths.

After filtering and smoothing, the prediction error results are shown in Figure 50 and 53. Compared to the error thermogram before filtering, the error is significantly reduced. By plotting the regression curves (distance and angle) before and after filtering (Figure 51a and 51b, Figure 54a and 54b), we can see that the filtered model predicts better.
In conclusion, the entire training process adheres to a systematic pipeline, encompassing the selection of a machine learning model, definition of its input and output parameters, subsequent hyperparameter tuning, and final evaluation of its performance. The final results are shown in Table 21

Table 21: Final model selection results

Target	Model Selection	Input Selection	Output Selection	Hyperparameter
Cyclist	KNN	Low mid x,Low mid y,height, width	KNN Polar Coordinates	n = 2
Escooterists	KNN	Low mid x,Low mid y,height, width	KNN Polar Coordinates	n = 2
Cars	KNN	Low mid x,Low mid y,height, width	KNN Polar Coordinates	n = 2

5 Discussion

In this chapter, various aspects of the work done will be discussed and potential limitations presented.

5.1 Discussion on the overall result

In the results discussion section, we first delve into the performance of the model, particularly in the detection of e-scooterist and distance estimation models. When validating the model on the training set, we observed commendable performance. However, challenges arise when validating natural data sets, particularly in the case of pedestrian prediction. For instance, when visually estimated distances of 10 meters in a video are compared with the model's prediction of 3 meters, a significant discrepancy is evident, which indicates notable errors. This highlights the need for a more in-depth investigation into the model's generalization capabilities in real-world scenarios.
A significant challenge stems from the absence of ground truth values in natural datasets, hindering our ability to accurately assess the model's performance in real-world environments. The limitations imposed by the lack of known ground truth values underscore the importance of future work in collecting additional real-world data to enhance the robustness of our evaluation framework.
Furthermore, in the context of distance prediction models, we observed minimal differences between the prediction results of the KNN model and the Random Forest Regression models. This makes it challenging to determine which model performs better on Naturalistic datasets definitively. This suggests a need for further optimization in model selection and parameter tuning to enhance performance in real-world environments.

5.2 Training dataset for YOLOv7

The training dataset used for the weight "yolov7x.pt" of YOLOv7 was based on COCO which is a more general training dataset which could introduce more errors for object detection or classification compared if a more specific training dataset would have been used. The benefit of using a more traffic-related dataset is unknown, due to time this was not investigated, but it could improve the performance of the models. An example of when this potentially caused a problem was an incident where the model classified a road marking as a cyclist.

5.3 Detection accuracy of PCM & CDM

Figure 24 shows the proportion distribution curve of the ICM, which is the model that classifies the single image into pedestrian or e-scooterist. Ideally, the e-scooterist should has a high proportion when the sigmoid score is near 0 (in other words, the blue line (e-scooterist) should have a high proportion value (>90%) when the sigmoid value is smaller than 0.5), and vice versa pedestrian. However, only pedestrian detection has a good performance, and the performance of e-scooterist detection is worse than pedestrian detection. The most significant reason for this phenomenon is that the image dataset that trains the ICM is different from the image extracted from the track test video. The image dataset that trains the ICM is not captured by a fisheye camera, and the camera mounting locations and image capture environments also vary. All these factors could

affect the performance of ICM. Another possible reason is that the picture quality of some of the person objects that are too far away is too poor (too small BBox), leading to inaccurate predictions by the ICM. As a result, the weighted score algorithm (Figure 9) is introduced to mitigate the shortcomings of the ICM model. By checking the result of PCM (Table 7), we found that the detection accuracy is acceptable and much better than solely using the ICM model itself as the e-scooterist detection model, which means the weighted score algorithm works and is necessary.
The CDM is a rule-based algorithm based on the overlap area. Although using the rate between the overlap area and the overall area to do cyclist detection is not rigorous, the detection performance on the test track video (Table 8) shows an acceptable accuracy.

5.4 Limitations of the test data for distance estimation

Since the test dataset only includes a few people acting as VRUs, the developed algorithms may be biased towards people with similar height and size. In normal traffic situations, there could be a number of different people on the road, which could introduce problems. To combat this issue the algorithms should be ideally trained on people of all ages and heights, to avoid bias. It would also be preferred to include strollers and wheelchairs. The same could be said about the car used, since there are a lot of different vehicles on the road and not just sedans. To properly train the algorithms, the test data should also include (but not be limited to): trucks, buses, and trams. Finally, the impact of different weather conditions would also be of great interest to investigate. But to include such a vast number of road users, vehicles, and different weather conditions, the cost of the test data would increase by a lot, making it difficult to finance.
From table 4, the number of tests for the e-scooter rider is half of the number for the cyclist and car, which is not ideal. Ideally, there should be the same number of tests for each road user.
As can be seen in Figure 48 to 50 and Figure 52 to 53, there are parts of the figure which are white, which means that there are no data in these areas. This results in a model that is not trained properly, since training data is not complete. This has a risk of lowering the model's performance which makes predictions in some parts of the video frames less accurate. This is not the case for e-scooterists and cyclists since the training data does not have the same problem.

5.5 Calculation of the closest point of the car

As can be seen in Figure 18, the plot for the closest point of the car to the e-scooter deviates more than expected from the center of the car when the car is far away. This is most likely due to rounding errors and the fact that a small error in the polar angle will cause a greater error in the calculation of position due to the large radial distance. This didn't cause any significant problems since this error occurs a large distances where the camera is unable to detect the car. Since the machine learning algorithms are based on frames where the car is detected, this error is not included in the model.

5.6 E-scooter orientation

The transformation of the coordinate systems was based on the orientation of the e-scooter in relation to the runway, there could be errors introduced in the calculations. The angles were determined from the videos, which were: -90°, 90°, -45°, and 45°, but it is possible that the actual angle could be slightly different. This is due to the fact that the angles are based on visual estimation, which could be improved by using the containers, as can be seen in Figure 55, as a reference by using LiDAR data. This is why the orientation could introduce errors when using polar coordinates, especially when the distance is large. Because the lateral and longitudinal distances are based on the distance and angle. Below follows an example of -45° and 45°, which could deviate from the actual angle, and rotation of the e-scooter relative to the runway:

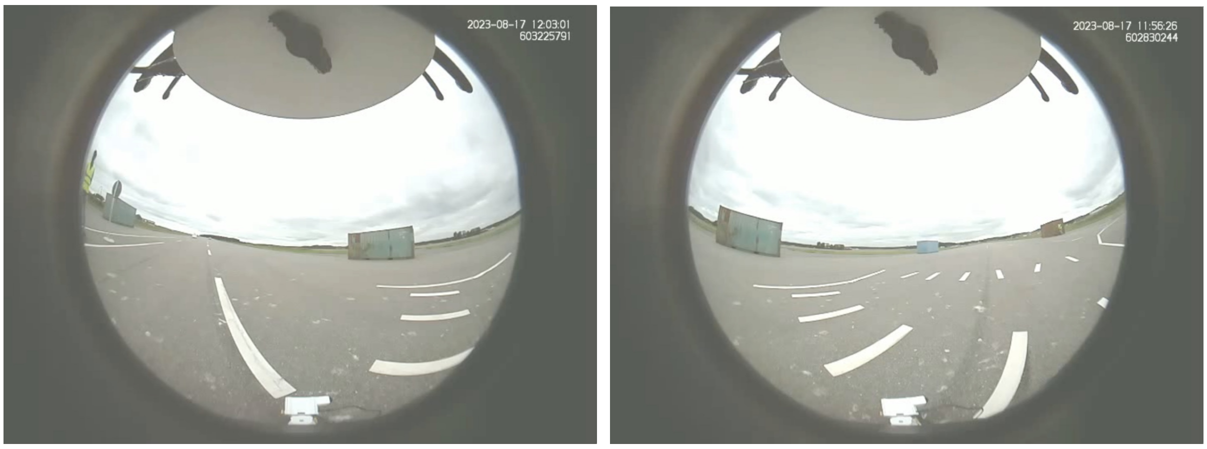


Figure 55: 45° and -45° rotation of the e-scooter relative to the runway.

5.7 Object detection at extreme angles and distances

When watching the videos produced by YOLOv7 the ID of the tracked object can change over time due to, the object moving out of frame or loss of detection for a short time. When synchronizing the video and distance data each id of the tracked object is used to make sure that the amount of training data is maximized from each video. The most problematic case is when the tracked object is partially in the frame, this is shown below:

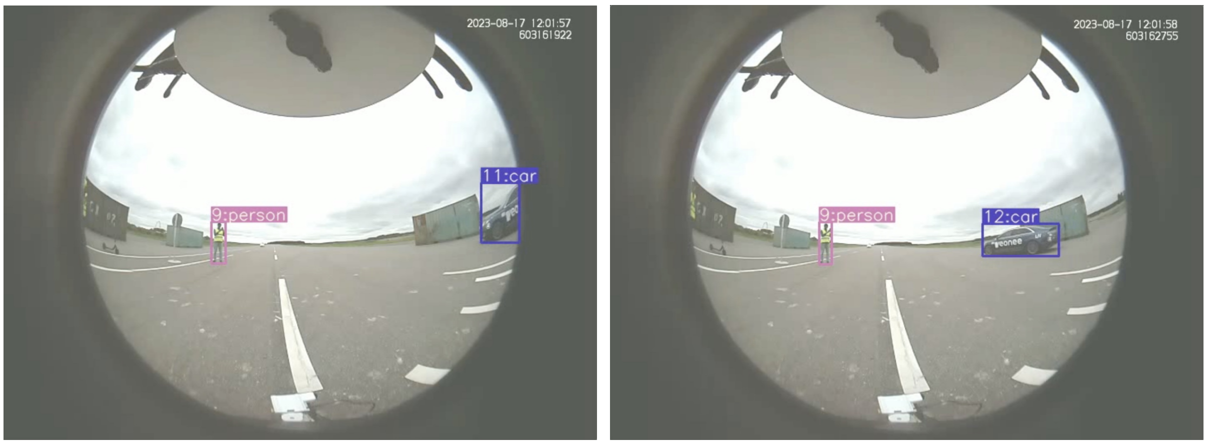


Figure 56: ID of the tracked object changes as the object moves into frame.

As shown in the figure above, the ID of the car changes as the car moves into frame, from 11 to 12, which can be a problem due to the performance of the camera is poor at extreme angles due to the distortion of the fish-eye lens is greatest at extreme angles, as discussed in the report by Bharti [6]. This causes the data to be noisy which could affect the training data and in turn the models, since the video data differs substantially from the ground truth. However, due to the fact that the amount of data from these cases is small compared to the whole training data set, this problem is not likely to affect the machine learning algorithms noticeably. As the distance increases the same type of problems can occur. These frames could be excluded by only considering objects that exist in a more limited field of view or decreasing the distance limit of the detection model to ensure better camera performance for all recorded frames. Another compromise, which is not ideal, could be to exclude IDs which only appear for a short time (<1 second) from the training data set.

5.8 Critical event detection

Currently, the relative kinematics of the traffic objects could be derived from video, but the critical events are not able to be precisely detected. Firstly, the accuracy of the detected kinematics on Naturalistic video data hasn't been verified. The Naturalistic video does not have ground truth data of its traffic objects. As a result, the accuracy of the kinematics of traffic objects cannot be rigorously proven. By observing the Naturalistic video, we found that some objects labeled as persons are far away from the ego, but the model detects them close, although the model works perfectly on the test video. As a result, we couldn't believe the kinematics derived from the model when the model was applied to the Naturalistic video. Secondly, the velocity and heading of the ego e-scooter are not included in the given Naturalistic data. Currently, we only have the relative position and velocity of the traffic objects. A more accurate trajectory prediction of the traffic object could be made by combining both the relative kinematics of traffic objects and the ego e-scooter kinematics such as velocity and heading. Thirdly, a simple critical event detection algorithm could be developed and set, for example, if the predicted trajectory of the traffic object in 2s is close to the ego e-scooter, then the algorithm is been triggered, but the accuracy of the algorithm would not be good, which means the false positive rate and the false negative rate would be high. In conclusion, the information derived from the current model is insufficient to detect the critical event from the Naturalistic video data.

6 Conclusion

In this section, the project will be summarized, including the most important steps during the project.
The work done in this project shows that it is entirely possible to estimate the position of the road user although with some uncertainties. For all road users, the whole model, including both position estimation and smoothing, can reach a R^2 above 0.9 on the test track video. Furthermore, by manually visualizing the result, we also found that the model has a good position estimation accuracy on naturalistic video. Estimating road-users kinematics will assist in the identification of near-crash scenarios from naturalistic data. Using kinematics to extract critical interactions is not only more objective, as each viewer has their own definition of what is and what is not a critical interaction, but also saves time compared to visual inspection.

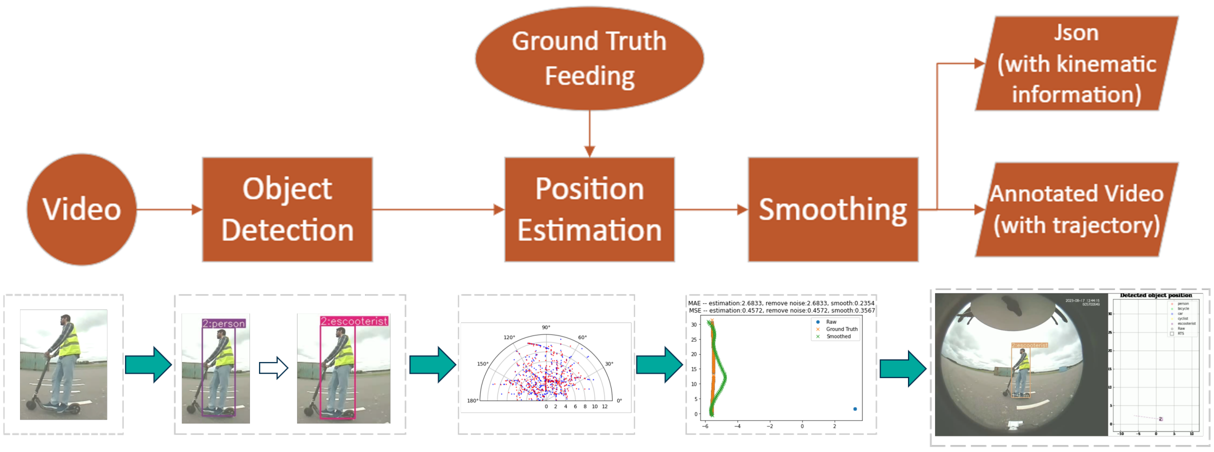


Figure 57: A summarized view of the pipeline, where a regular video is analyzed to detect the different road users. Then the positions of the road users are estimated using k-neighbors regressor which is then filtered using DBSCAN. Finally, the annotated video with trajectories and the JSON file with the kinematic information is obtained.

The overall process of the project work was described in Figure 57. When the video was saved from the e-scooter, it would be fed to the object detection algorithm which would detect and classify each road user. Then both the distance and position of the road users would be estimated using the machine learning algorithms. Then, to remove any outliers or noise from the predictions a DBSCAN-based noise remover was implemented to filter out the outliers and produce a more accurate result. After that, an RTS smoother was applied to both estimate the kinematics information and improve the distance estimation accuracy. Finally, a JSON file that contains the kinetic information and an annotated video with a plot that shows the detected object trajectory was generated. Both these final outputs could then be used to automatically and objectively identify critical interactions between different road users.
Using this pipeline video data from the e-scooter can be analyzed without a human needing to watch everything. By defining a threshold for critical interactions, it would be possible to objectively and automatically find these critical situations.

7 Future work

For future work, it would be interesting to apply the methods used in this project on a larger data set and compare the results. A larger data set would ideally contain more road users, making it possible to train the model on a more diverse environment which would make the models more representative of real traffic. It would also be beneficial to obtain more data for each of the different road users to increase the robustness of the models, especially for the car. In terms of the training of YOLOv7, similar projects could benefit from using a more traffic-specific training data set instead of the more general COCO training set. To further improve the YOLOv7 model, more training data is needed since the performance of the models will likely increase with more diverse data of different road users. Also, the rule-based cyclist detection model used in this project can be improved as it's not robust enough when the cyclist appears for a short time. A deep learning-based Convolutional Neural Network (CNN) model can be built to distinguish between a person, a bicycle, and a cyclist, leveraging its effectiveness and accuracy in image recognition tasks.

Reference

[1] WHO. *Road traffic injuries.* URL: `https://www.who.int/health-topics/road-safety#tab=tab_3`.

[2] European Commission. *ITS Vulnerable Road Users.* URL: `https://transport.ec.europa.eu/transport-themes/intelligent-transport-systems/road/action-plan-and-directive/its-vulnerable-road-users_en`.

[3] Kumar Apurv. *E-scooter Rider Detection System in Driving Environments.* 2021. URL: `https://doi.org/10.25394/PGS.15057183.v1`.

[4] Kumar Apurv, Renran Tian, and Rini Sherony. "Detection of E-scooter Riders in Naturalistic Scenes". In: *arXiv preprint arXiv:2111.14060* (2021).

[5] Jing Zhu and Yi Fang. "Learning object-specific distance from a monocular image". In: *Proceedings of the IEEE/CVF International Conference on computer vision.* 2019, pp. 3839–3848.

[6] Karan Bharti. *Estimating road-user position from a camera: a machine learning approach to enable safety applications.* 2023. URL: `https://odr.chalmers.se/items/c5c843b1-773e-4c9d-a8a7-b291ca6614fd`.

[7] Yury Davydov, Wen-Hui Chen, and Yu-Chen Lin. "Supervised object-specific distance estimation from monocular images for autonomous driving". In: *Sensors* 22.22 (2022), p. 8846.

[8] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao et al. "YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors". In: arXiv, 2022. URL: `https://doi.org/10.48550/arXiv.2207.02696`.

[9] Andrew G. Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: *arXiv* (2017).

[10] Mark Sandler et al. "MobileNetV2: Inverted Residuals and Linear Bottlenecks". In: *arXiv* (2018).

[11] Gareth James et al. *An introduction to statistical learning.* Vol. 112. Springer, 2013.

[12] Trevor Hastie et al. *The elements of statistical learning: data mining, inference, and prediction.* Vol. 2. Springer, 2009.

[13] Christopher M Bishop. *Pattern Recognition and Machine Learning by Christopher M. Bishop.* Springer Science+ Business Media, LLC, 2006.

[14] Trevor Hastie et al. *The elements of statistical learning: data mining, inference, and prediction.* Vol. 2. Springer, 2009.

[15] Jeff Heaton. *Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning: The MIT Press, 2016, 800 pp, ISBN: 0262035618.* Vol. 19. 1-2. Springer, 2018, pp. 305–307.

[16] Erich Schubert et al. "DBSCAN revisited, revisited: why and how you should (still) use DBSCAN". In: *ACM Transactions on Database Systems (TODS)* 42.3 (2017), pp. 1–21.

[17] Caizhi Zhang et al. "Review of Clustering Technology and Its Application inCoordinating Vehicle Subsystems". In: *Automotive Innovation* (2022). URL: `https://doi.org/10.1007/s42154-022-00205-0`.

[18] F. TUNG H. E. RAUCH and C. T. STRIEBEL. "Maximum likelihood estimates of linear dynamic systems". In: *AIAA JOURNAL* (1965).

[19] E. (n.d.) *Downloadable dataset for e-scooter Rider Detection Task, and a trained model to support the detection of e-scooter riders*. URL: `http://situated-intent.net/e-scooter_dataset/`.