

# An estimation and correction of the error of the lateral position of the vehicle within the road network based of camera data

---

**Mužar, Matija**

**Master's thesis / Diplomski rad**

**2025**

*Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj:* **University of Zagreb, Faculty of Mechanical Engineering and Naval Architecture / Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje**

*Permanent link / Trajna poveznica:* <https://urn.nsk.hr/urn:nbn:hr:235:907343>

*Rights / Prava:* [In copyright](#) / [Zaštićeno autorskim pravom.](#)

*Download date / Datum preuzimanja:* **2025-02-07**

*Repository / Repozitorij:*

[Repository of Faculty of Mechanical Engineering and Naval Architecture University of Zagreb](#)



UNIVERSITY OF ZAGREB  
FACULTY OF MECHANICAL ENGINEERING AND NAVAL  
ARCHITECTURE

# **MASTER'S THESIS**

**Matija Mužar**

**Zagreb, 2025.**

UNIVERSITY OF ZAGREB  
FACULTY OF MECHANICAL ENGINEERING AND NAVAL  
ARCHITECTURE

# MASTER'S THESIS

Mentor:

Prof. dr. sc. Tomislav Stipančić, dipl. ing.

Student:

Matija Mužar

Zagreb, 2025.

## IZJAVA

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

## STATEMENT

I hereby declare that I have made this thesis independently using the knowledge acquired during my studies and the cited references.

## ZAHVALA

Na početku bih se htio zahvaliti mentoru s fakulteta izv.prof.dr.sc. Tomislavu Stipančiću na prihvaćanju mentorstva i za korisne sugestije tijekom izrade ovog rada. Zatim, zahvaljujem se Mariju Đurasu za pruženoj prilici i za omogućavanje izrade diplomskog rada unutar tvrtke dSPACE engineering d.o.o. Nadalje se zahvaljujem Jakovu Topiću kao mentoru unutar tvrtke, te kolegama iz ureda, posebno Luki, na korisnim savjetima i sugestijama te na svojoj pruženoj pomoći.

Najsrdajnije se zahvaljujem obitelji, prijateljima s fakulteta te iz studentskog doma Stjepan Radić koji su bili uz mene tijekom cijelog studija te mi pružali pomoć i podršku.

Na kraju se zahvaljujem svojoj djevojci Valentini na pruženoj potpori tijekom izrade ovog rada.

Matija Mužar

## ACKNOWLEDGMENT

At the beginning I would like to thank Full Prof. PhD Tomislav Stipančić for accepting the mentorship for this master's thesis and for useful suggestions during this study. Next, I would like to thank Mario Đuras for giving me the opportunity to do my master's thesis within the dSPACE engineering d.o.o. I express gratitude to my mentor in the company, Jakov Topić and other colleagues from the office, especially Luka, for giving me useful advices and suggestions and all other help.

I would like to thank my family, friends from college, and friends from the dormitory Stjepan Radić who were with me during the study and gave me help and support.

Finally, I thank my girlfriend Valentina for her support during this work.

Matija Mužar



SVEUČILIŠTE U ZAGREBU  
FAKULTET STROJARSTVA I BRODOGRADNJE



Središnje povjerenstvo za završne i diplomske ispite  
Povjerenstvo za diplomske ispite studija strojarstva za smjerove:  
Proizvodno inženjerstvo, inženjerstvo materijala, industrijsko inženjerstvo i menadžment,  
mehatronika i robotika, autonomni sustavi i računalna inteligencija

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: 602 - 04 / 25 - 06 / 1	
Ur.broj: 15 - 25 -	

## DIPLOMSKI ZADATAK

Student: **Matija Mužar** JMBAG: 0035226276

Naslov rada na hrvatskom jeziku: **Procjena i korekcija greške bočnog položaja vozila unutar cestovne mreže na temelju podataka s kamere**

Naslov rada na engleskom jeziku: **An estimation and correction of the error of the lateral position of the vehicle within the road network based of camera data**

Opis zadatka:

The general approach to generating virtual scenarios required for validation and testing of automatic systems in vehicles typically involves equipping the vehicle with a certain sensor and collecting data through real driving on certain road segments. The collected data is then processed/annotated in order to reconstruct the trajectories of dynamic objects which are then used as input data in the process of generating virtual scenarios from the recorded driving data. However, due to noisy and/or incomplete sensor data, trajectories may be incorrectly determined during the data annotation process which consequently leads to possible errors in the generated scenarios which are used for the purpose of scenario-based testing.

The main goal of this thesis is to develop an algorithm for the localization of accompanying vehicles within the road network with an emphasis on the estimation and correction of the lateral position of the vehicle within the lane using a set of recorded camera images and GPS data. The algorithm needs to be verified in computer simulations using recorded data of real-world drives.

Within the scope of the work, it is necessary to:

- use Lane Line Detection AI model in conjunction with Object Detection AI model to estimate the lateral position error of the accompanying vehicles within the road network (i.e., correct lane and lateral offset within a lane)
- match the annotated vehicles (from recorded data) with the ones detected in the camera images
- calculate lateral position errors within annotated trajectories
- correct the lateral positions of annotated vehicle trajectories based on estimated errors.

In the work, it is necessary to cite the literature used and any assistance received.

Zadatak zadan:

28. studeni 2021.

Zadatak zadao:

Izv. prof. dr. sc. Tomislav Stipančić

Datum predaje rada:

30. siječnja 2025.

Predviđeni datumi obrane:

6., 7. i 10. veljače 2025.

Predsjednik Povjerenstva:

Prof. dr. sc. Ivica Garašić

## CONTENTS

CONTENTS .....	I
LIST OF FIGURES .....	III
LIST OF SYMBOLS AND UNITS .....	V
SAŽETAK .....	VI
SUMMARY .....	VII
PROŠIRENI SAŽETAK .....	VIII
1. INTRODUCTION .....	1
1.1. Background .....	1
1.2. Problem analysis and approach to the task .....	2
2. AVAILABLE DATA AND MODELS .....	3
2.1. Input data.....	3
2.1.1. Camera images/video .....	3
2.1.2. Annotated trajectories .....	5
2.1.3. Road network map .....	6
2.2. ONNX models.....	7
2.2.1. Lane Line Detection model.....	7
2.2.2. Object Detection Model .....	9
3. LANE IDENTIFICATION.....	11
3.1. Determination of road lane lines .....	11
3.2. Identification of driving lanes for fellows.....	14
3.3. Lane matching.....	18
4. LATERAL OFFSET ESTIMATION .....	22
4.1. Lateral offset estimation via horizontal intersections .....	22
4.2. Lateral offset estimation via „angled“ intersections .....	25
5. OBJECT MATCHING .....	28
5.1. Projecting fellow coordinates to image plane .....	28
5.1.1. Transformation from world coordinate system to ego coordinate system.....	28
5.1.2. Transformation from ego coordinate system to camera coordinate system.....	30
5.1.3. Transformation from camera coordinate system to image plane coordinate system .....	31
5.2. Matching the trajectory with camera data.....	32
6. DATA CORRECTION.....	34
7. RESULTS .....	37
7.1. Results for limited dataset.....	37
7.2. Results for whole dataset .....	41
8. CONCLUSION .....	46
BIBLIOGRAPHY .....	49
APPENDICES.....	51





---

**LIST OF FIGURES**

Figure 1.1: Process of data annotation. ....	1
Figure 1.2: Possible lateral offset mistakes. ....	2
Figure 2.1: Example of input image. ....	4
Figure 2.2: Camera used for data collection. ....	5
Figure 2.3: Format of trajectory data for dynamic objects stored in object list data. ....	6
Figure 2.4: Graphical representation of road network map [3]. ....	7
Figure 2.5: Input image with scattered points. ....	8
Figure 2.6: Input image with detected bounding box points. ....	9
Figure 2.7: Input image with detected bounding boxes and labels. ....	10
Figure 3.1: Detected lines using RANSAC and Cluster algorithm. ....	12
Figure 3.2: Distance calculation for lines with positive slope. ....	13
Figure 3.3: Distance calculation for lines with negative slope. ....	13
Figure 3.4: Bounding boxes with corresponding reference points. ....	14
Figure 3.5: First polygon which filters out the irrelevant oncoming traffic objects (marked with yellow dashed lines). ....	15
Figure 3.6: Second polygon which filters out the „out-of-scope“ traffic objects (marked with magenta dashed lines). ....	16
Figure 3.7: Third polygon example for lane number one (marked with red dashed lines). ....	16
Figure 3.8: Example camera image with Object Detection and Lane Line Detection results. ....	17
Figure 3.9: Road Network Data output file format. ....	18
Figure 3.10: Leftmost lane. ....	19
Figure 3.11: Rightmost lane. ....	19
Figure 3.12: Left barrier detected. ....	20
Figure 3.13: Right barrier detected. ....	20
Figure 3.14: Both barriers detected. ....	20
Figure 3.15: None of the barriers are detected, number of lanes is 5 (solvable case). ....	20
Figure 3.16: None of the barriers are detected, number of lanes is > 5 (unsolvable case). ....	20
Figure 3.17: Example image with detection results, including the additional observation case information put at top. ....	21
Figure 4.1: Example results for lateral offset estimation via horizontal intersections. ....	24
Figure 4.2: Example results for lateral offset estimation via „angled“ intersections. ....	26

Figure 5.1: Object matching flowchart. ....	28
Figure 5.2: Image/pixel coordinate system. ....	31
Figure 5.3: Fellows projections on image plane presented by yellow dots.....	32
Figure 5.4: Process of matching fellows. ....	33
Figure 6.1: Execution plan for data correction.....	34
Figure 6.2: ( s/t )–coordinate system. ....	35
Figure 6.3: Calculation of new t-coordinate.....	36
Figure 7.1: Test fellow trajectories for lateral offset estimation via horizontal intersections..	37
Figure 7.2: First input image for test trajectories. ....	38
Figure 7.3: Test fellow trajectories for lateral offset estimation via „angled“ intersections... 39	
Figure 7.4: Smoothened and corrected test trajectories for lateral offset estimation via horizontal intersections.....	40
Figure 7.5: Smoothened and corrected test trajectories for lateral offset estimation via „angled“ intersections.....	41
Figure 7.6: Fellow trajectories for lateral offset estimation via horizontal intersections for the first 250 images. ....	42
Figure 7.7: Case where object matching is not possible. ....	43
Figure 7.8: Fellow trajectories for lateral offset estimation via „angled“ intersections for first 250 images.....	44
Figure 7.9: Fellow trajectories for lateral offset estimation via „angled“ intersection for first 250 images after using the filter. ....	44
Figure 7.10: Fellow trajectories for lateral offset estimation via „angled“ intersections for first 250 images after filter.....	45
Figure 8.1: Scenario where object matching mistakes occur. ....	47
Figure 8.2: Scenario where bounding box rotation could improve lateral position estimation. ....	48

---

**LIST OF SYMBOLS AND UNITS**

Oznaka	Jedinica	Opis
$x$	m	Coordinate axis in global coordinate system
$y$	m	Coordinate axis in global coordinate system
$x'_1$	pixel	Translated coordinate in image coordinate system
$y'_1$	pixel	Translated coordinate in image coordinate system
$x'_{1,rotated}$	pixel	Translated and rotated coordinate in image coordinate system
$y'_{1,rotated}$	pixel	Translated and rotated coordinate in image coordinate system
${}^{ego}T_{global}$		Transformation matrix from global to ego coordinate system
${}^{global}T_{ego}$		Transformation matrix from ego to global coordinate system
$f_x$	pixel	Focal length of camera in x-direction
$f_y$	pixel	Focal length of camera in y-direction
$c_x$	pixel	Coordinate of image center in image coordinate system
$c_y$	pixel	Coordinate of image center in image coordinate system
$u$	pixel	Coordinate axis in image coordinate system
$v$	pixel	Coordinate axis in image coordinate system
$x_{cam}$	pixel	X-coordinate in camera coordinate system
$y_{cam}$	pixel	Y-coordinate in camera coordinate system
$z_{cam}$	pixel	Z-coordinate in camera coordinate system
$t_{min}$	m	Minimum value of t coordinate for specific lane
$t_{max}$	m	Maximum value of t coordinate for specific lane

**SAŽETAK**

Generalni pristup kod kreiranja virtualnih scenarija potrebnih za validaciju i testiranje automatskih sustava u vozilima tipično se sastoji od opremanja vozila s nekom senzorskom konfiguracijom. Konfiguracije mogu uključivati, kameru, radar, LiDAR itd. Nakon što se odabere senzorska konfiguracija, podaci se prikupljaju vožnjom na stvarnim cestovnim segmentima. Podaci koji su sakupljeni su nakon toga procesuirani/anotirani kako bi se rekonstruirale trajektorije vozila u pokretu, koje se koriste kao ulazni podatak u procesu generiranja virtualnih scenarija iz snimljenih podataka. Tijekom procesa anotacije podataka i procesuiranja i zbog netočnih senzorskih podataka, trajektorije mogu biti netočno određene što dovodi do mogućih grešaka u generiranim scenarijima. Unutar ovog diplomskog rada, potrebno je razviti algoritam za lokalizaciju popratnih vozila unutar cestovne mreže s naglaskom na estimaciju i ispravljanje lateralne pozicije vozila unutar vozne trake koristeći set snimljenih slika i GPS podataka. Algoritam će koristiti module umjetne inteligencije za detekciju linija i objekata kako bi se estimirao položaj vozila na cesti. Detektirana vozila će zatim biti uparena s vozilima iz snimljenih trajektorija. Finalno, metoda za procjenu lateralnog pomaka mora biti razvijena, a podaci ispravljani. Algoritam također mora biti verificiran. Cilj je smanjenje nedosljednosti lateralnog pomaka između snimljenih podataka i pripadajuće cestovne mreže tako da u simulaciji imamo istu situaciju koja se dogodila u stvarnome svijetu.

Ključne riječi: testiranje s pomoću scenarija, anotirane trajektorije, lateralni pomak, ispravljanje podataka.

---

**SUMMARY**

The general approach to creating virtual scenarios required for validation and testing of automatic systems in vehicles typically involves equipping the vehicle with a certain sensor configuration. This sensor configuration can include cameras, radars, LiDARs etc. After the sensor configuration is picked, data is collected through real driving on certain road segments. The collected data is then processed/annotated in order to reconstruct the trajectories of moving/dynamic objects, which are then used as input data in the process of generating virtual scenarios from the recorded driving data. Throughout this process of data annotation and processing, and due to noisy sensor data trajectories can be incorrectly determined which consequently leads to possible errors in the generated scenarios. Within the scope of this Master Thesis, it is necessary to develop an algorithm for the localization of accompanying („fellow“) vehicles within the road network with an emphasis on the estimation and correction of the lateral position of the vehicle within the lane using a set of recorded camera images and GPS data. The algorithm will use AI modules for line and object detection to estimate vehicle position on the road. Then detected vehicles have to be matched with the vehicles from the recorded trajectories. Finally, a method for lateral offset estimation needs to be developed, and data has to be corrected. The algorithm needs to be verified. Aim is to reduce lateral position inconsistencies between the recorded trajectories and the corresponding road network so that in the simulation we have the same situation as it happened in the real world.

Key words: scenario-based testing, generated scenarios, annotated trajectories, lateral offset, data correction.

---

**PROŠIRENI SAŽETAK**

Ovaj rad izrađen je u suradnji s tvrtkom dSPACE Engineering d.o.o. u Zagrebu koja je podružnica njemačke tvrtke dSPACE. Navedene tvrtke bave se pružanjem rješenja za simulacije i validacije diljem svijeta u svrhu razvijanja umreženih, autonomnih te električno pogonjenih vozila.

Cilj ovog diplomskog rada je razviti algoritam koji će ispravljati bočne položaje vozila na temelju podataka dobivenih s pomoću modula umjetne inteligencije. Koristeći module za prepoznavanje objekata te za prepoznavanje linija vozila su grupirana u trake po kojima voze. Nakon što su se vozila grupirana, povezana su s vozilima iz trajektorija. Povezana su na način tako da su koordinate vozila iz trajektorija transformirane iz globalnog koordinatnog sustava u koordinatni sustav slike te su povezana s odgovarajućim vozilom. Bočni položaj vozila je estimiran na način tako da se gledao položaj graničnog okvira dobivenog AI modulima unutar vozne trake. Nakon što se dobio položaj, koordinate iz trajektorija su ispravljene prema dobivenim podacima.

Istraživanje [\[1\]](#) govori o tome da kako bi se ispunili zahtjevi funkcionalne sigurnosti za autonomnu vožnju (ISO 26262) autonomno vozilo mora biti testirano na stotine milijuna kilometra. Kako stvarne testne vožnje rezultiraju u povećanom riziku za testnog vozača tako i s visokom cijenom po kilometru. U zoni interesa nije samo veliki broj kilometara nego i reprezentacija relativnih i kritičnih kilometara. Prema tome svoju primjenu pronalaze testiranja u laboratoriju. Testni kilometar može biti standardiziran te je također jeftiniji. Virtualne testne vožnje podijeljene su u dva stadija, a to su testiranja Software-In-Loop (SIL) koji se fokusira na validaciju algoritama neovisno o vremenskim ograničenjima. Testovi se mogu provoditi prije nego hardver uopće i postoji. Drugi stadij, Hardware-In-Loop je stadij u kojem je finalna hardverska platforma (Electronic Control Unit) ECU spojena na simulator koji oponaša ponašanje pravog auta te se tako simulira cjelokupan hardver zajedno s softverom. Relativno na razvitak autonomne vožnje, sve više i više funkcija se ugrađuje u ECU stoga je testiranje bez virtualnih metoda skoro pa nemoguće.

Kod virtualnog simuliranja na temelju scenerija, scenariji igraju jaku bitnu ulogu jer što je scenarij točniji, stvarni uvjeti će biti točnije prikazani te će samim time i testiranje biti sličnije realnom svijetu. Scenariji se prikupljaju tako da testno vozilo s nekom senzorskom konfiguracijom prikuplja podatke te se ti podaci još dodatno anotiraju te se u tome javljaju greške, zbog šuma, greške senzora, greške anotatora itd. Kroz ovaj rad će se ispraviti greške

bočnog položaja vozila te će se tako poboljšati točnost kvaliteta snimljenih trajektorija. Ovaj rad je organiziran u 6 poglavlja te je na kraju dat zaključak.

**Poglavlje 1** – 'Uvod' – U uvodu je dan kratak opis trenutnog stanja automobilske industrije te je opisan postupak kreiranja virtualnih scenarija. Nakon toga je opisana problematika kojom se bavi ovaj diplomski rad te je opisan pristup ka rješenju.

**Poglavlje 2** – 'Materijali i metode' – Unutar ovog poglavlja opisani su ulazni podaci koji se sastoje od videa/slika, mape ceste te trajektorija vozila. Predstavljene su korišteni moduli umjetne inteligencije, a to su modul za prepoznavanje linija te modul za prepoznavanje objekata te su opisani njihovi ulazno/izlazni parametri.

**Poglavlje 3** – 'Identifikacija trake' – U ovom poglavlju opisani su izlazni parametri modula umjetne inteligencije za prepoznavanje linija. Opisan je postupak procesuiranja podataka te su spomenuti neki od algoritama koji su korišteni prilikom istog. Opisan je postupak grupiranja detektiranih linija u vozne trake. Također je i opisan način kako su se detektirane vozne trake uskladile s voznim trakama koje su zadane u trajektorijama.

**Poglavlje 4** – 'Estimacija lateralnog pomaka' – Ovo poglavlje govori o metodama korištenim za pronalaženje bočnog pomaka detektiranih vozila. Predstavljena su dva pristupa za računanje bočnog pomaka te je za svaki od njih detaljno objašnjen.

**Poglavlje 5** – 'Uparivanje vozila' – Unutar ovog poglavlja obrađeno je projiciranje globalnih koordinata u koordinatni sustav kamere. Objašnjeni su koordinatni sustavi te transformacije koje ih povezuju. Nakon što su koordinate vozila prikazane u koordinatnom sustavu slike, detektirana vozila su povezana s vozilima koja su detektirana AI modulom za prepoznavanje objekata.

**Poglavlje 6** – 'Ispravljanje podataka' – Ovo poglavlje govori o načinu kako su podaci iz trajektorija ispravljeni. Opisuje kako se podaci dobiveni u poglavlju 4 koriste ka ostvarivanju cilja ovog diplomskog rada, a to je ispravljanje trajektorija.

**Poglavlje 7** – 'Rezultati' – U ovom poglavlju vizualizirane su dobivene trajektorije. Dobivene ispravljene trajektorije su dodatno uspoređene originalnim neispravljenim trajektorijama. Također, ispravljene trajektorije su dodatno uglađene korištenjem jednog od filtra. Dobiveni podaci prvo su prikazani za testni skup podataka, a nakon toga i na cijeli skup podataka.

---

**Poglavlje 8** – 'Zaključak' – Unutar zaključka opisana je uspješnost algoritma u estimaciji i ispravljanju lateralnog pomaka zajedno sa svim ostalim njegovim zadaćama. Spomenuta su neka od ograničenja algoritma te su predložena poboljšanja samog rada.



# 1. INTRODUCTION

## 1.1. Background

Car industry is one of the biggest world industries and a key driver of global economies. Also, it's one of the leaders in innovative engineering solutions. Current state of the car industry is that it's leaning more and more towards development of autonomous vehicles. Huge global car companies are investing in AI-driven solutions for safer and more efficient driving. To fulfil the requirements of functional and effective testing for autonomous driving, autonomous car needs to be test driven on thousands of kilometres. One of the approaches in testing is scenario based testing which is usually performed in simulation with Software-in-the-loop (SIL), but can also be performed with Hardware-in-the-loop (HIL) or in the real world on polygons. Important piece of information for scenario based testings are generated scenarios. Generated scenarios are generally created by equipping the vehicle with a certain sensor configuration and collecting the data through real driving on certain road segments. The collected data is then processed/annotated in order to reconstruct the trajectories of moving/dynamic objects which are then used as input data in the process of generating virtual scenarios from the recorded driving data. The Process of data annotation is shown in Figure 1.1:

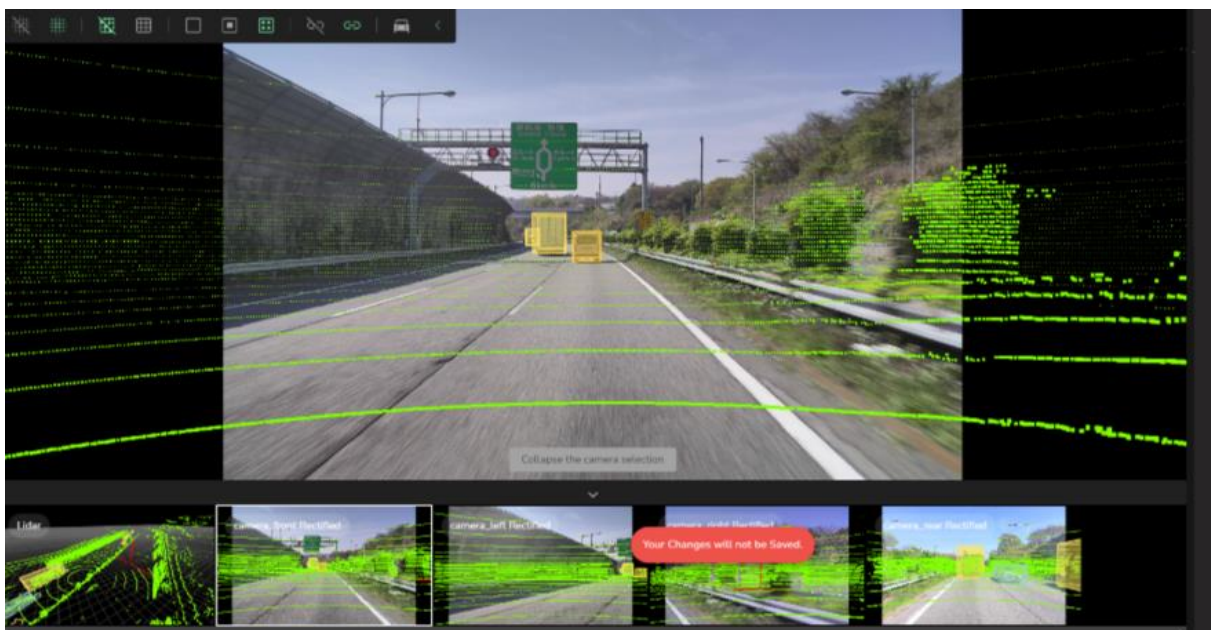


Figure 1.1: Process of data annotation.

Company responsible for data annotation for dSPACE is Understand.ai. Throughout this process of data annotation and processing and due to noisy sensor data, trajectories can be incorrectly determined which consequently leads to possible errors in the generated scenarios.

## 1.2. Problem analysis and approach to the task

The main problem that this thesis is trying to solve is that lateral position of accompanying vehicles in generated scenarios is inaccurate. Main two sources of error come from labelling error and measurement error. Lateral position of the surrounding vehicles can be inaccurate in two ways, lateral offset can be determined wrong, but vehicle is still in correct lane or lateral offset is determined wrong, but also the driving lane is wrong. Figure 1.2 shows those two cases:

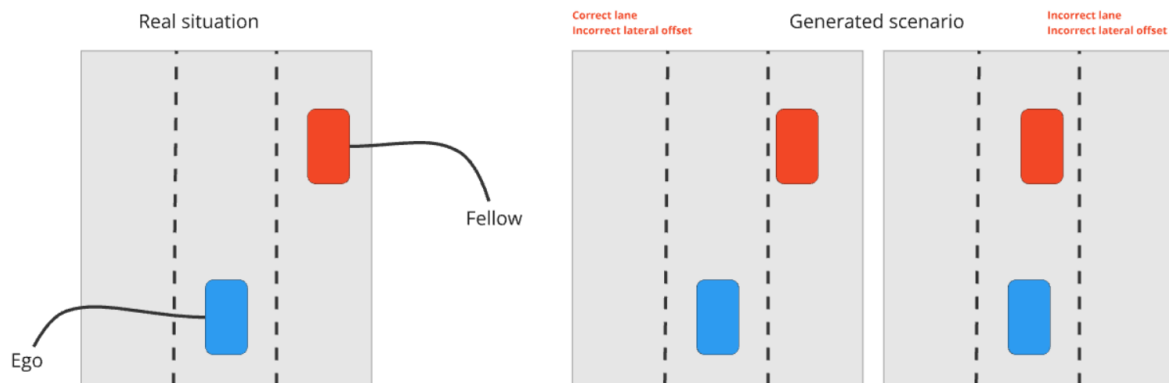


Figure 1.2: Possible lateral offset mistakes.

The approach to the problem is following: AI modules for line detection and object detection are used to detect moving object and lane lines. Results from the modules are processed and detected objects are grouped into driving lanes that they are driving in. After they are grouped into the lanes, lateral offset is calculated in two ways. To connect data from the image and from the trajectory detected objects are matched with objects from the data. That is done in a way that objects from trajectories are projected to the image plane via multiple coordinate system transformations and matched to detected vehicles. Data is then corrected and results are presented.

---

## 2. AVAILABLE DATA AND MODELS

Main focus of this chapter will be on the interpretation of the input data and the AI models that were provided by the company. Output of both AI models will be presented and some of the algorithms used in output data processing will be described. Finally, outputs will be visualised and conclusion will be given.

This chapter starts with introduction of the available input data and related data types. Next, the ONNX format in which AI models are provided is introduced. Then, Lane Line Detection model and Object Detection model are briefly described. Finally, post-processing of output data for Object Detection model is described.

### 2.1. Input data

Input data to be used includes the camera video (split into a set of images), object list data with all the annotated trajectories for dynamic objects and the corresponding road network map.

#### 2.1.1. Camera images/video

The camera data contains the relative lateral positioning between the vehicle and the road network. The lane markings on highways are often clearly defined and can help to determine the correct lateral position of the vehicles.

Camera data consists of video of the road which lasts 3 minutes and 29 seconds. Video is divided into 4180 pictures which is approximately 20 frames per second. Each of those pictures is then loaded and processed in Python. Figure 2.1 shows example of the camera image.



Figure 2.1: Example of input image.

Camera data is collected with camera and LiDAR setup. Camera used is SC5-IMX490C-5300-GMSL2 with the resolution of 2880\*1860. Camera uses Sony's IMX490 RGGB 5.44 Mega Pixel sensor [2].



Figure 2.2: Camera used for data collection.

### **2.1.2. Annotated trajectories**

As mentioned in the introduction, part of the process in creating virtual scenarios is data collection, which is then processed/annotated in order to reconstruct trajectories. These trajectories of moving objects are used as input data. For every dynamic object, a set of coordinates is assigned for every frame/timestamp. Each timestamp is matched with two corresponding images. Figure 2.3 shows illustration of trajectory assignment per dynamic object in object list.

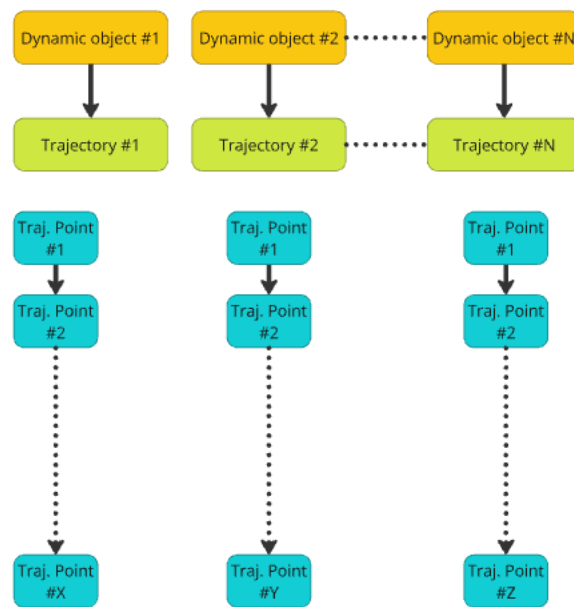


Figure 2.3: Format of trajectory data for dynamic objects stored in object list data.

### 2.1.3. Road network map

The road network-related map data is in OpenDrive format. It provides information about the individual road elements (as components of road network), their number of lanes and the lane widths, that can be used to transform into the position estimations obtained from the camera data to determine the absolute trajectories of the vehicle.

Assumption is that the map data is the ground truth, because in case of HD maps the road network is probably more reliable than the annotated trajectories. Likewise, it is easier to

modify the trajectories than the road network. Figure 2.4 shows the graphical representation of the road network map.

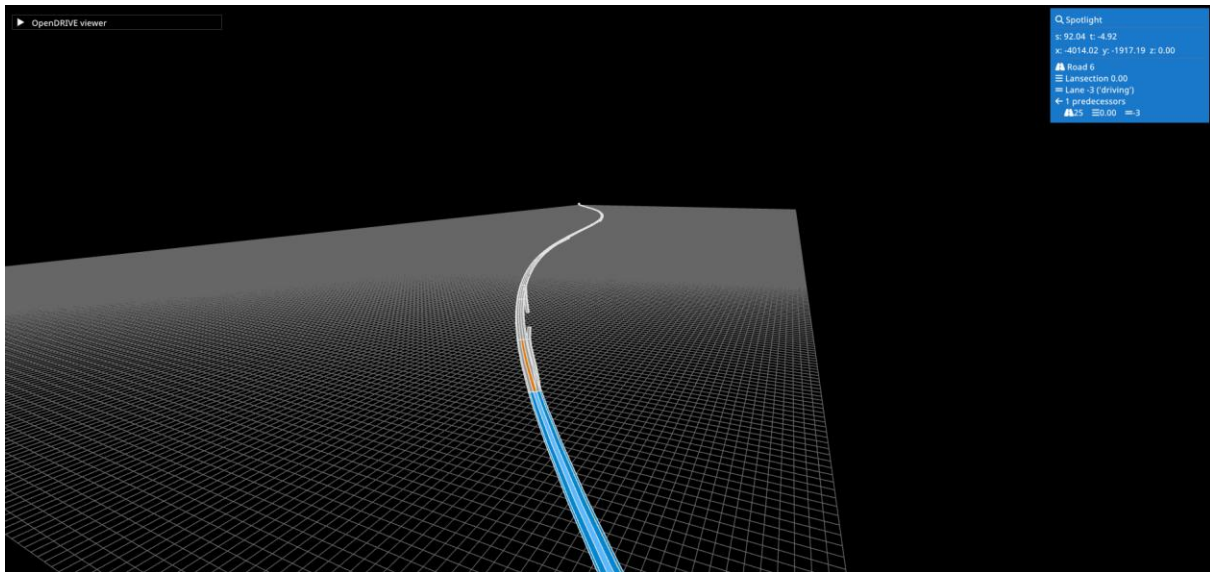


Figure 2.4: Graphical representation of road network map [3].

## 2.2. ONNX models

Both of the models are given in Open Neural Network Exchange (ONNX) format. This format presents a common format used for representing deep learning models that can be then later executed by multiple frameworks, including TensorFlow, PyTorch, etc. ONNX format is useful because any model can be formatted into ONNX format, and from ONNX can be converted back into any desired framework. Also, it is efficient and can be run on a wide range of hardware (GPU, CPU). Provided models were run using Python library ONNX Runtime. Generally speaking, ONNX Runtime is engine that allows users to optimize and accelerate machine learning inferencing [4], [5].

### 2.2.1. Lane Line Detection model

Lane Detection model detects lines on images and sorts them by type (solid line, dashed line, boundary, etc.). Input to model is score threshold and image which is imported using python Open CV library. When image is loaded it is then converted into the array and resized to match the format that model provides. Before running the model, the related configuration

must be loaded. Model configuration is provided in .yaml format and contains information about the model and about the model output.

After the model inputs and the configuration are prepared, model is ran using the main class of the ONNX Runtime library called `InferenceSession`.

Output of the model contains some information about the image. Most important information it contains are points, scores and classes for each point. Score describes how well model predicted something, i.e., reliability in prediction. Points that belong to certain class present the line group that was detected. Using the Matplotlib's function `scatter()` we can visualise the model output. Figure 2.5 shows output of the Lane Line Detection model laid over the original/input image.

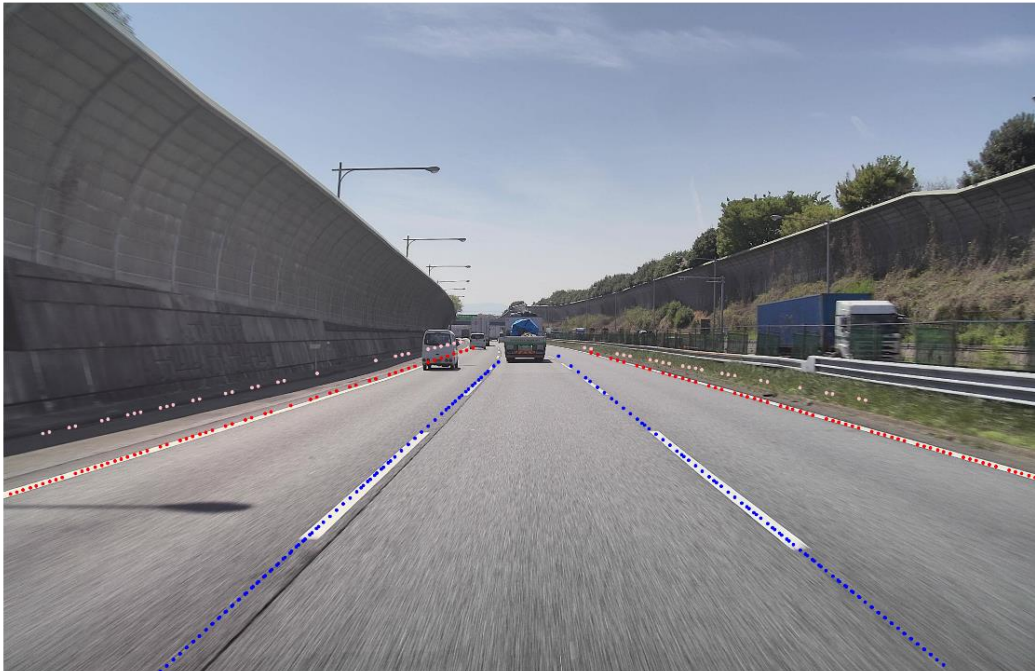


Figure 2.5: Input image with scattered points.

In Figure 2.5 it is visible that points are scattered in three colours: pink, red and blue. Each colour presents a different class to which point belongs. Points drawn pink belong to class „Barrier“ where red points belong to class „Solid Line“ and blue points belong to class „Dashed Line“.

Model has some limitations and those are that it only works on a highway since it is trained on highway data set, and detected lines do not cover the full length of the road in some cases.



### 2.2.2. Object Detection Model

Object Detection Model detects objects (that can be moving vehicle, traffic sign etc.) on images and sorts them by classes they belong to. Herein it is necessary to distinguish between two terms: „ego“ and „fellow“ vehicle. Ego vehicle refers to a vehicle that is equipped with sensors and perceives the surrounding environment, i.e. the surrounding traffic consisting of pedestrians, static objects and accompanying vehicles, which are called fellow vehicles.

Similarly, as the Lane Line Detection model, the Object detection model also takes image as an input together with the score threshold.

Outputs of the Object Detection model are points and classes to which points belong. Those points present starting and ending coordinates of the bounding box around the object that was detected. Figure 2.6 shows those detected points on input image.



Figure 2.6: Input image with detected bounding box points.

Using Python library Matplotlib and its function *Rectangle* bounding boxes are visualised. To differentiate between different classes to which detected objects belong, bounding boxes are plotted in different colours. Also, the labels with vehicle ID and class (i.e., vehicle category), were added, as shown in Figure 2.7.



Figure 2.7: Input image with detected bounding boxes and labels.

Same as the previous model, Object Detection model also has some limitations, it only works on highway data same as previous model, 2D bounding boxes are not accounting for driving direction (heading angle), and it is front camera support only, meaning only fellows positions that are in front of ego can be estimated.

### 3. LANE IDENTIFICATION

This section describes some of the methods and algorithms that are used when post-processing the output data of the Lane Line Detection and Object Detection models. Afterwards, the method that is used to identify the driving lane of the fellows is described. Finally, the process of mapping detected lanes with lanes from the road network map is described.

#### 3.1. Determination of road lane lines

After the points were outputted/predicted with Lane Line detection model, next step is connecting those points into the lines which then represents the road lane lines. Problem that occurs is that points are provided in the separate lists, where each list contains all the points that belong to one class. For example, all the “blue” points which belong to multiple lines are grouped into a single list. To separate those groups of points into lane lines, the clustering algorithm was used.

The applied algorithm is DBSCAN from the Python *Scikit-learn* library, which contains many efficient tools for machine learning and statistical modelling. DBSCAN-Density Based Spatial Clustering of Applications with Noise finds core samples of high density and expands clusters from them. In general, algorithm is suitable for data which contains clusters of similar density [6].

When data was separated into clusters, another algorithm from *Scikit-learn* library was used, RANSACRegressor. [7] RANSAC (RANdom SAmple Consensus) algorithm is an iterative algorithm for the robust estimation of parameters from a subset of inliers from the complete data set. The algorithm splits the complete input sample data into a set of inliers, which may be subject to noise, and outliers, which are, e.g., caused by erroneous measurements or invalid hypotheses about the data. The resulting model is then estimated only from the determined inliers. Figure 3.1 presents the visualized output of the Lane Detection model after using clusters and RANSAC algorithm on the output data of AI model.



Figure 3.1: Detected lines using RANSAC and Cluster algorithm.

Next step needed to determine the fellows position is to sort lines in order, e.g., from left to right if the leftmost boundary line is detected, so that we can use that information during mapping of detected lanes to the road network map. Lines were sorted in a specific way. First step in sorting the lines is to calculate the slope of each line. With the slope being positive or negative, we split them into two groups. Lines with positive slopes are on the left side of the ego vehicle and lines with negative slopes are on the right side of the ego. That information is very valuable because we can extract number of lanes on left side of the ego and number of lanes on right side of the ego. Thanks to that information, we can get an ego position and know in which lane ego is situated in some cases where it is possible to calculate.

Lines are sorted into the list so that leftmost line comes in at a first place in a list, while the rightmost line comes last in a list. To get that list, first step is to add lines with positive slope since they are on left side of the ego thus they are always more left than lines with negative slope. That is achieved by calculating the distance between point at the start of the line (closest to the camera) and the point on top left corner (image origin in pixel coordinates). Since we are in image/pixel coordinate system point in the top left corner has coordinates (0, 0). Line with the smallest distance comes first in list which means that it is leftmost line. Points on lines that are closest to ego camera were calculated using min/max functions in Python. Figure 3.2 shows those distances. Pink line presents barrier, so it was ignored in distance calculation.



Figure 3.2: Distance calculation for lines with positive slope.

Lines with negative slope were sorted with the same logic as the lines with positive slope, only difference is that reference point for calculating the distance was not point (0,0) but instead point(2880, 1860) which presents point in the bottom right corner as shown in Figure 3.3:



Figure 3.3: Distance calculation for lines with negative slope.

After the lines with negative slopes were sorted in a way so that line with the least distance comes first and line with most distance comes last, they were appended to a list which now contains sorted lines.

### 3.2. Identification of driving lanes for fellows

Method for identifying the driving lane of the fellows includes the polygon creation, where each polygon represents the corresponding lane boundaries, and then checking in which polygon the fellow is situated. The polygons are created by using the Python's library *Shapely*, while the check if fellow is within a polygon is realized by using the *.within()* function in Python.

First step in doing that is to find the reference points of bounding boxes that will be used for „is-fellow-within-polygon“ checks. Reference points x-coordinates are calculated simply as arithmetic mean of starting and ending bounding box x-coordinates, while the y-coordinates are equal to bounding box ending coordinates provided by Object Detection model output.



Figure 3.4: Bounding boxes with corresponding reference points.

After the fellows reference points are determined, three different types of polygons are created.

First, the polygon aims to filter out the irrelevant traffic objects, i.e., the ones which are part of oncoming traffic. It is constructed from point (2880,0) in pixel coordinates, which is the top right corner, the end point of the last lane line (point with the smallest x value) and the last point is the intersection between the elongated last lane line and line that goes through coordinates (2880,0) and (2880. 1860), representing the right edge of the image, as shown in Figure 3.5.



Figure 3.5: First polygon which filters out the irrelevant oncoming traffic objects (marked with yellow dashed lines).

The second polygon aims to filter out the out-of-scope traffic objects, i.e., objects left of the leftmost detected lane line, e.g., in the case of a road section whose number of lanes exceeds the limitations of Lane Line Detection model. It is constructed from the end point of the first lane line (point with biggest x value), point (0,0) which is in the top left corner, and the point at coordinates (0, y), where y is representing the y coordinate of the intersection of the first lane line and line between points (0,0) and (0,1860), representing the left edge of the image, as shown in Figure 3.6.

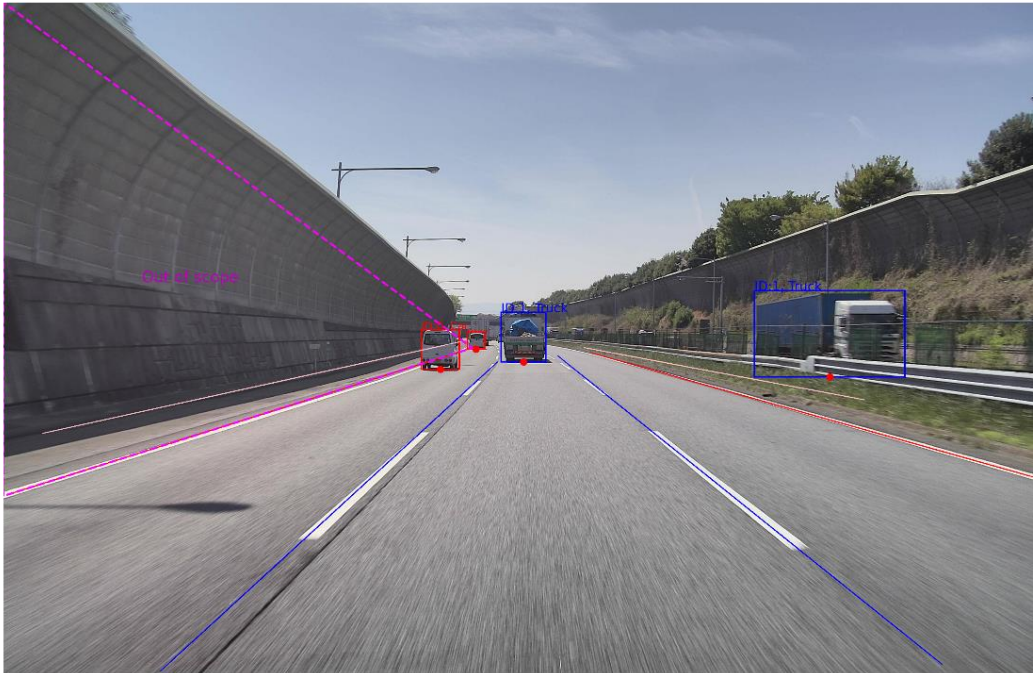


Figure 3.6: Second polygon which filters out the „out-of-scope“ traffic objects (marked with magenta dashed lines).

The third polygon defines the area of a certain driving lane, which is created by each pair of lane lines from the list of lane lines.

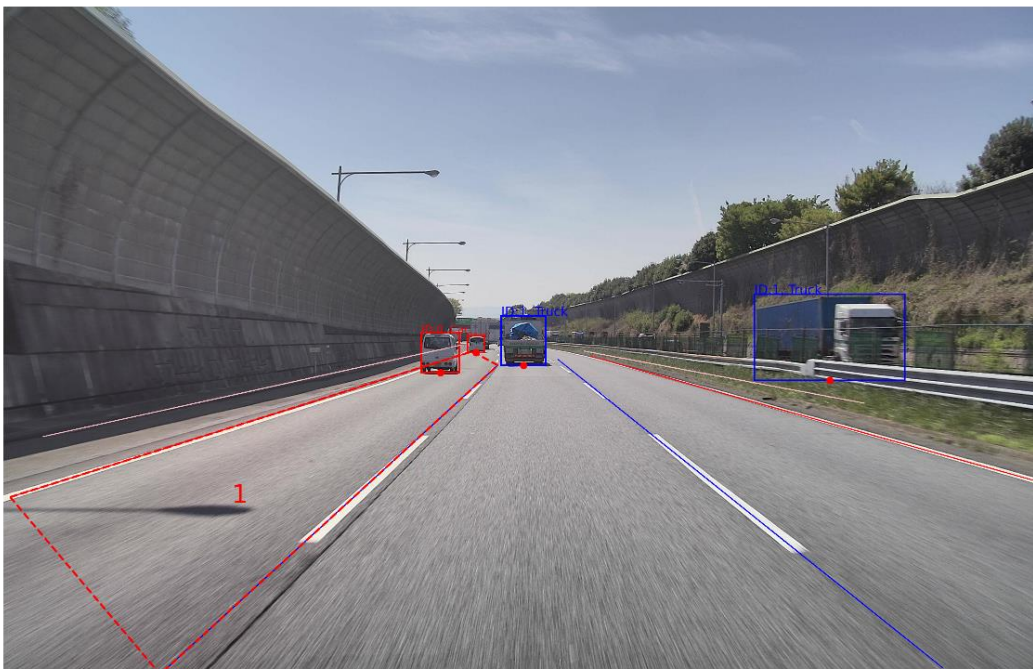


Figure 3.7: Third polygon example for lane number one (marked with red dashed lines).



Final step after post-processing of Lane Line Detection model data together with Object Detection model data is to combine them and visualize it on the same image, as shown in Figure 3.8:

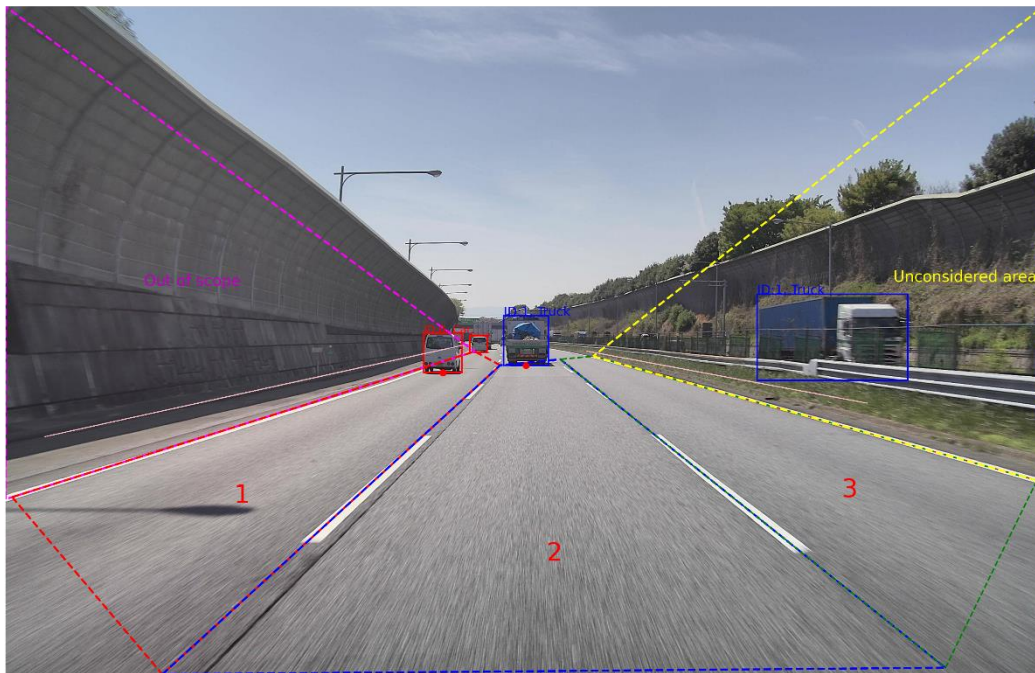


Figure 3.8: Example camera image with Object Detection and Lane Line Detection results.

All the information about the detected fellows on an image are stored in a special dictionary *fellows\_position*, which is structured as follows:

```
{'lanes': {1: [{'matches_to_lane': ''},
               {'bbox': (1160, 906, 1266, 1017),
                'bbox_ref_point': {'u': 1213, 'v': 1017},
                'class': 'Car',
                'id': '0.0',
                'scores': '0.7557713'}]},
  2: [{'matches_to_lane': ''},
       {'bbox': (1382, 858, 1506, 996),
        'bbox_ref_point': {'u': 1444, 'v': 996},
        'class': 'Truck',
        'id': '1.0',
        'scores': '0.6120546'}]}}
```

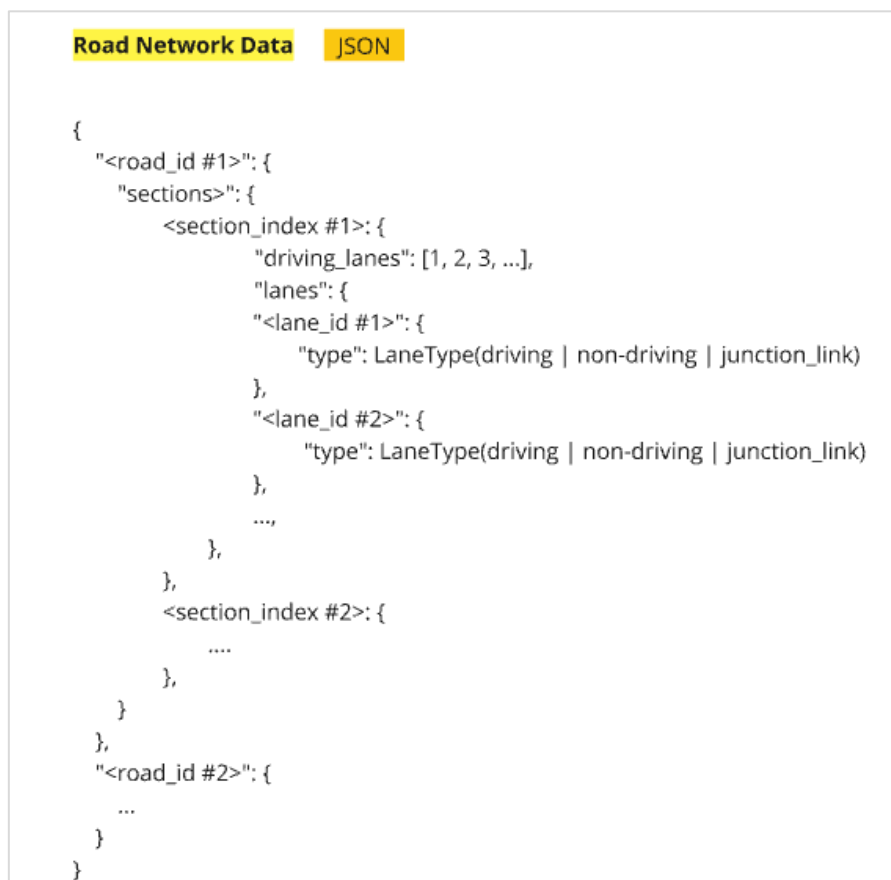
```

3: [{'matches_to_lane': ""}],
'out_of_scope': [],
'unconsidered': [{'bbox': (2086, 799, 2503, 1038),
                    'bbox_ref_point': {'u': 2295, 'v': 1038},
                    'class': 'Truck',
                    'id': '1.0',
                    'scores': '0.79898787'}]]}

```

### 3.3. Lane matching

Next step after the lanes were detected in camera image is to match those detected lanes with lanes from the OpenDRIVE road network map. First step is to load the data from the road file that is in OpenDRIVE format. After the data is loaded, it is then reorganized and written in an output file called Road Network Data in JSON format.



```

{
  "<road_id #1>": {
    "sections": {
      "<section_index #1>": {
        "driving_lanes": [1, 2, 3, ...],
        "lanes": {
          "<lane_id #1>": {
            "type": LaneType(driving | non-driving | junction_link)
          },
          "<lane_id #2>": {
            "type": LaneType(driving | non-driving | junction_link)
          },
          ...
        },
      },
      "<section_index #2>": {
        ...
      },
    },
  },
  "<road_id #2>": {
    ...
  },
}

```

Figure 3.9: Road Network Data output file format.

Formatting data in that way gives a much more cleared and organised view into the data. Most important part of Road Network Data are identified driving lanes, for each road network element (i.e., road or junction element). They are put in a list and assigned as a value to key *driving\_lanes*. This list is then used for matching.

Figure 3.10 shows that for the road network element (RNE) with ID "2\_0" the leftmost lane, relative to the moving direction (marked with red arrows), corresponds to OpenDRIVE lane ID '-2'. Likewise, by looking at the Figure 3.11 it is visible that the rightmost detected lane corresponds to OpenDRIVE lane ID '-4'.

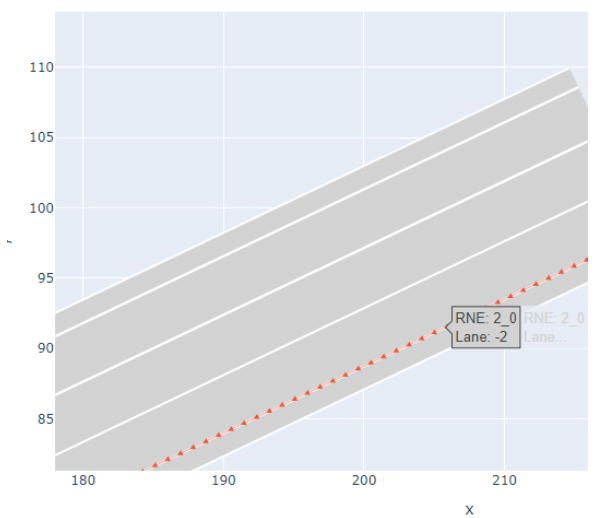


Figure 3.10: Leftmost lane.

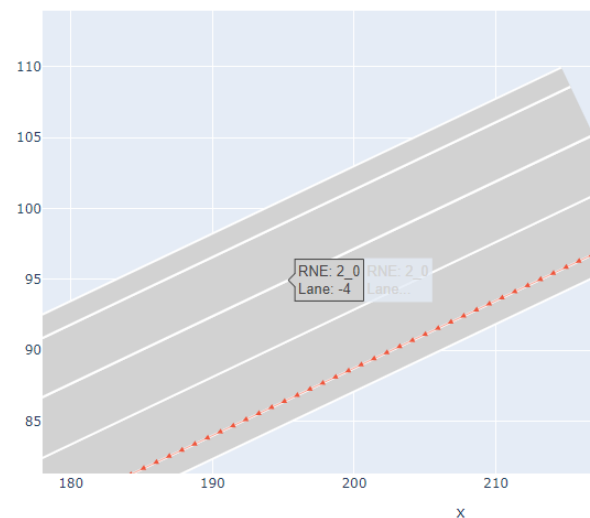


Figure 3.11: Rightmost lane.

Before the lanes were matched, another piece of information is important, and that is information about whether the left and right barriers are detected. Lanes can only be matched if one of the barriers is detected or if the number of lanes is 5. The reason for that is the limitation of the AI models so that a maximum of 5 road lanes can be detected. If one of the barriers is detected, there is a reference for matching the lanes. If there are no references, but there is information that there are 5 lanes, and with the limitation that only 3 lanes are detectable, it can be assumed that ego is in the middle lane and the two lanes that are not detected are lanes next to barriers. All the cases that were taken into concern are presented in the figures below. In the figures, red lines present detected barriers, green lines present detected barriers and black lines present lines that were not detected. The ego vehicle is painted blue, and the fellow vehicle is painted red.

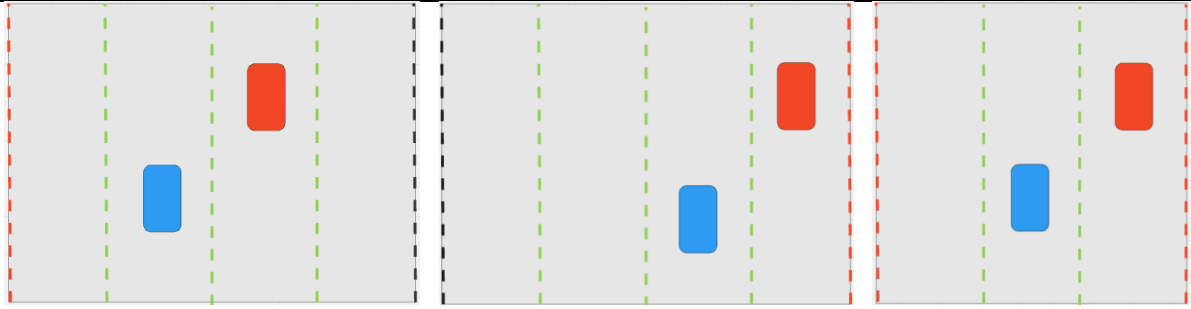


Figure 3.12: Left barrier detected.

Figure 3.13: Right barrier detected.

Figure 3.14: Both barriers detected.

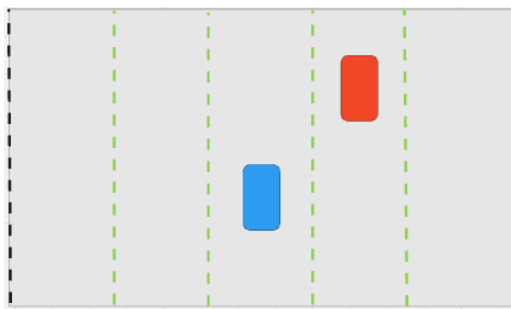


Figure 3.15: None of the barriers are detected, number of lanes is 5 (solvable case).

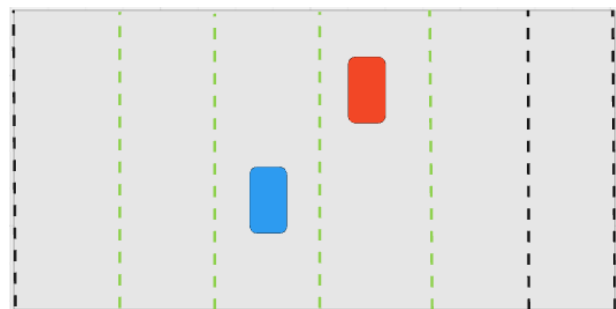


Figure 3.16: None of the barriers are detected, number of lanes is  $> 5$  (unsolvable case).

In the case when only the left barrier is detected, visible in Figure 3.12, the first element of the list of lanes from road network data is matched with the first lane detected (i.e., leftmost lane). The second lane is matched with the second lane from the left and so on. Information about which lane from road network data matches with the detected lane is stored as value to a key *matches\_to\_lane* for every lane.

If only the right barrier is detected, visible in Figure 3.13, matching starts from the last member of the list of lanes from road network data with the rightmost lane (i.e., last in the list of detected lanes), and so on. In the case of both barriers being detected, visible in Figure 3.14 match starts from the left, by convention.

In cases where no barriers are detected, presented in Figure 3.15 and there are 5 driving lanes, matching starts from the ego lane, meaning ego lane is the middle one. The number of lanes is calculated using information about detected lines. That lane is then matched with the middle lane of the driving lanes.



## 4. LATERAL OFFSET ESTIMATION

This section describes how lateral offsets of fellow vehicles are found. Two solutions will be presented together with their advantages and disadvantages. Comparison of their results will be presented in section 7 later on.

### 4.1. Lateral offset estimation via horizontal intersections

First solution for lateral offset calculation uses intersections between the horizontal line going through bounding box lower middle point (reference point) and lines that define the lane in which fellow currently is driving. Solution is implemented in the python using the function which takes as an input 4 points and outputs one point, intersection. First, input points are two points that belong to the left line and other two points are bounding box reference point and point (0, bounding box ref.point Y). The output point is the intersection between the horizontal line going through bounding box reference point and left lane line. Same principle is repeated for the right lane line. Calculus used to find that intersection will be described in the following text.

To find intersection between two lines, first step is to solve system of linear equations that present each line, and those lines describe the fellow lane. Lines are described with following equations:

$$a_1x + b_1y = c_1 \quad (1)$$

$$a_2x + b_2y = c_2 \quad (2)$$

Solving these two equations, we get formulas how to find x and y coordinates of an intersection:

$$x = \frac{(b_2 * c_1 - b_1 * c_2)}{a_1 * b_2 - a_2 * b_1} \quad (3)$$

$$y = \frac{(a_1 * c_2 - a_2 * c_1)}{a_1 * b_2 - a_2 * b_1} \quad (4)$$

It is visible that expressions (3) and (4) have unknown parameters that have to be found. They are found using the equation for the line between two points. Each point is given in coordinates as following:

$$A(x_1, y_1)$$

$$B(x_2, y_2)$$

Distance between two points can be calculated using the formula:

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1} * (x - x_1) \quad (5)$$

Rewriting expression (5) we get:

$$x (y_2 - y_1) + y(x_1 - x_2) - x_1y_2 - x_2y_1 = 0 \quad (6)$$

By comparing expression (6) to line equation in the expression (1): we can see that some unknown parameters for expression (3) and (4) can be calculated as following:

$$a_1 = (y_2 - y_1) \quad (7)$$

$$b_1 = (x_1 - x_2) \quad (8)$$

$$c_1 = -x_1y_2 - x_2y_1 \quad (9)$$

Expression (9) can be more simplified and can be rewritten as:

$$c_1 = a_1x_1 + b_1y_1 \quad (10)$$

That process of distance calculation is done for the first two points that are input to function that calculates lines intersection and also for last two points that describe other lane. Parameters that describe the second line can be calculated as following:

$$a_2 = (y_4 - y_3) \quad (11)$$

$$b_2 = (x_3 - x_4) \quad (12)$$

$$c_2 = a_2x_3 + b_2y_3 \quad (13)$$

When intersections are found, their coordinates are used to calculate the distances from bounding box lower midpoint. First distance is distance between fellow and left line and distance between bounding box and right line. By calculating lane width using formula for distance between two points (expression (5)) and putting mentioned distances to ratio, lateral position of the fellow on the lane is estimated. If fellow vehicle is exactly in the middle, left and right ratios are the same. If the right ratio is bigger, that means that fellow vehicle is closer to the left side of the lane and vice versa.



Figure 4.1: Example results for lateral offset estimation via horizontal intersections.

The flaw of this solution is that it does not take into account distortion of the image where lanes are smaller and smaller as distance from the camera increases. Lanes are also angled at a certain angle, so this solution results in not so correct estimations for vehicles that are not in ego lane. Bigger the lateral distance from the ego, less accurate is the offset estimation.



## 4.2. Lateral offset estimation via „angled“ intersections

This solution adds in extra step to the first solution. When intersections are found, they are rotated for the angle of the line on the left in case for fellows left from ego and for the angle of the right line in case for fellows right from ego. This process is done only for the non-ego lanes because for ego lane horizontal intersections approximate offset the best. When rotating 2D points about another 2D pivot point first step is to translate that point to the origin to be rotated.

The point that we are trying to rotate has following coordinates:

$$T(x_1, y_1)$$

Pivot point that we want to rotate point T about has coordinates:

$$P(p_1, p_2)$$

When translating T coordinates by P coordinates we get new T coordinates:

$$x'_1 = x_1 - p_1 \quad (14)$$

$$y'_1 = y_1 - p_2 \quad (15)$$

Next step is to rotate coordinates for the angle of the left line of the lane. Rotated coordinates are calculated by the following expressions:

$$x'_{1,rotated} = x'_1 \cos(\text{angle}) - y'_1 \sin(\text{angle}) \quad (16)$$

$$y'_{1,rotated} = x'_1 \sin(\text{angle}) + y'_1 \cos(\text{angle}) \quad (17)$$

After the coordinates are rotated, final step is to translate them back by adding the coordinates of the pivot point. Final formula for calculating x and y coordinates of a point T that is rotated by an angle about another point P is:

$$x'_1 = ((x_1 - p_1) \cos(\text{angle}) - (y_1 - p_2) \sin(\text{angle})) + p_1 \quad (18)$$

$$y'_1 = ((x_1 - p_1) \sin(\text{angle}) + (y_1 - p_2) \cos(\text{angle})) + p_1 \quad (19)$$

When new rotated points are found, next step is to find intersections between the line going through found points and between the left line and right line of the lane. Intersections are visible at Figure 4.2.



Figure 4.2: Example results for lateral offset estimation via „angled“ intersections.

Information about the left reference point and right reference point, along with information about the left ratio and right ratio is saved in dictionary *fellows\_positions*. Example of this dictionary for key 1 which presents first lane is:

```
{'lanes': {1: [{'matches_to_lane': -2},
```

```
{'bbox': (1160, 906, 1266, 1017),  
  'bbox_ref_point': {'u': 1213, 'v': 1017},  
  'class': 'Car',  
  'id': '0.0',  
  'left_distance': 98.23007274727775,  
  'left_lane_point': (1114, 1016),  
  'right_distance': 136.99329237603524,  
  'right_lane_point': (1350, 1016),  
  'scores': '0.7557713'}}}]}
```



coordinate system to ego coordinate system. Ego data from trajectories consists of six coordinates: x, y, z, yaw, roll and pitch.

Translation matrix is shown in equation (20):

$$Tran(p_x, p_y, p_z) = \begin{bmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (20)$$

Where value  $p_x$  presents x-coordinate of an ego,  $p_y$  presents y-coordinate and  $p_z$  presents z-coordinate.

Rotation matrix for yaw angle is shown in equation (21):

$$Rot(z, yaw) = \begin{bmatrix} \cos(yaw) & -\sin(yaw) & 0 & 0 \\ \sin(yaw) & \cos(yaw) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (21)$$

Rotation matrix for pitch angle is shown in equation (22):

$$Rot(y, pitch) = \begin{bmatrix} \cos(pitch) & 0 & \sin(pitch) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(pitch) & 0 & \cos(pitch) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (22)$$

Rotation matrix for roll angle is shown in equation (23):

$$Rot(x, roll) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(roll) & -\sin(roll) & 0 \\ 0 & \sin(roll) & \cos(roll) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (23)$$

Transformation matrix  ${}^{ego}T_{global}$  is calculated for every timestamp, and it describes relative position of fellow (global) coordinates to ego. It is calculated as following:

$${}^{ego}T_{global} = inv({}^{global}T_{ego}). \quad (24)$$

Where :

$${}^{global}T_{ego} = Tran(p_x, p_y, p_z) \cdot Rot(z, yaw) \cdot Rot(y, pitch) \cdot Rot(x, roll). \quad (25)$$

When multiplying fellow coordinates in global coordinate system with matrix  ${}^{ego}T_{global}$  as a result we get fellow coordinates in ego coordinate system. Matrix  ${}^{ego}T_{global}$  is calculated as inverse of the matrix  ${}^{global}T_{ego}$  because for calculating matrix  ${}^{global}T_{ego}$ , ego coordinates can just be included in that matrix.

### 5.1.2. Transformation from ego coordinate system to camera coordinate system

After the fellow coordinates are projected to ego coordinate system they need to be projected to camera coordinate system. Ego coordinates present a car with a camera and LiDAR at the top and coordinates that are in trajectories are coordinates of LiDAR. Transformation matrices that transform from LiDAR coordinate system to camera coordinate system are provided by the company. There are two provided matrices. One matrix describes extrinsics and the other one describes rotations to match ego and camera coordinate system.

To find fellow coordinates in camera coordinate system, fellow coordinates in ego coordinate system have to be multiplied as following:

$$\begin{bmatrix} x_{cam} \\ y_{cam} \\ z_{cam} \\ 1 \end{bmatrix} = inv(E) * rot * \begin{bmatrix} x_{lidar} \\ y_{lidar} \\ z_{lidar} \\ 1 \end{bmatrix} \quad (26)$$

Where  $R$  presents rotation matrix and  $E$  presents matrix that describes a camera to LiDAR position. We have to find inverse of the  $E$  matrix to get LiDAR to camera positioning.

### 5.1.3. Transformation from camera coordinate system to image plane coordinate system

Final step in transforming global fellow coordinates to image coordinate system is to project coordinates from camera to image coordinate system. This is a 3-D to 2-D transformation, so the depth information is lost. Calculation is done based on a Pinhole Model.



Figure 5.2: Image/pixel coordinate system.

Transformation matrix that does that is called Intrinsic matrix, and it depends on camera parameters. The intrinsic matrix has the following shape:

$$I = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (27)$$

Where:

$f_x$  and  $f_y$  are focal lengths of the camera in pixels,  $f_x$  in x-direction and  $f_y$  in y-direction and  $c_x$  and  $c_y$  are coordinates of the image centre (principal points), ideally points should be in the centre of the image however in practice and for this thesis it is not. To get points from a camera

coordinate system to image coordinate system, fellow coordinates (in camera coordinate system) have to be multiplied with camera intrinsics as following:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = I * \begin{bmatrix} x_{cam} \\ y_{cam} \\ z_{cam} \end{bmatrix}, \quad (28)$$

Where  $u, v$  present coordinates in image coordinate system visible at Figure 5.2. All that calculation is done in python using module *NumPy* for every fellow from fellow trajectory date for certain timestamp. Scattered fellow coordinates in image coordinate system are visible at Figure 5.3.

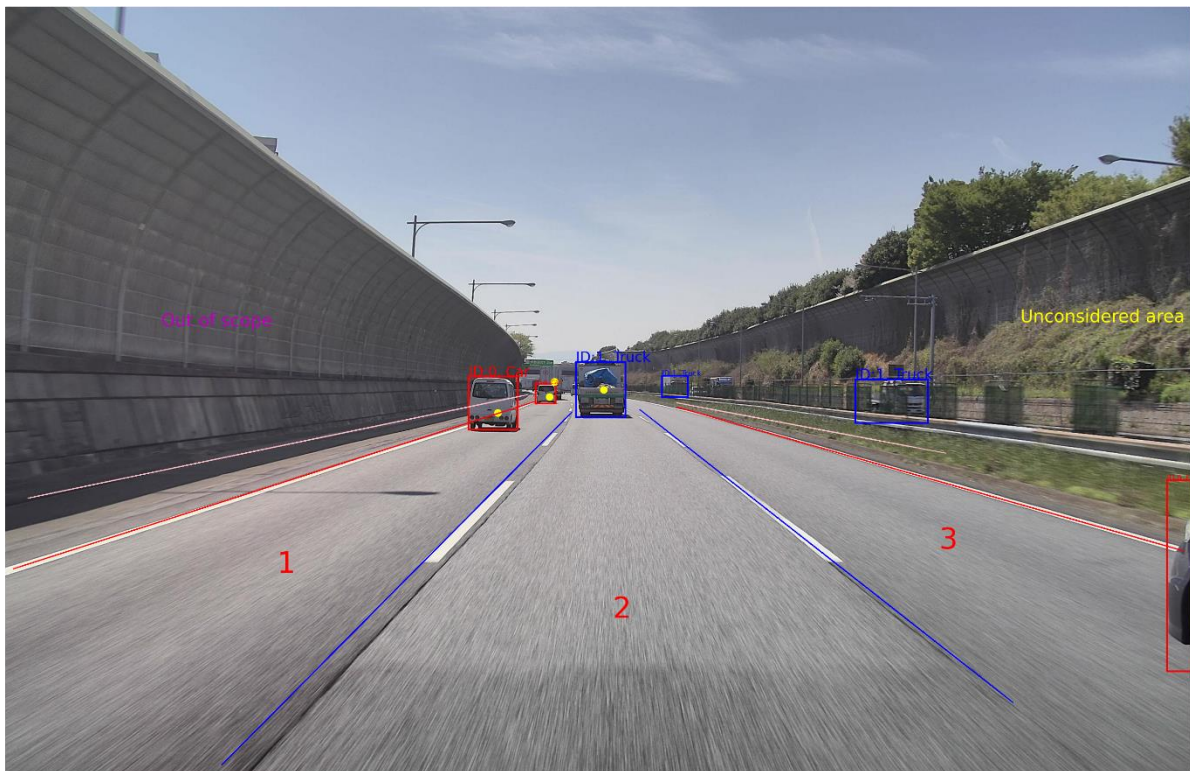


Figure 5.3: Fellows projections on image plane presented by yellow dots.

## 5.2. Matching the trajectory with camera data

Last step is to match fellows by rule point within bounding box. That process is done similar to the process of estimation of the fellow positions. For every bounding box a polygon is created, and every projected fellow is then checked if it is in that bounding box. When fellow that is inside that bounding box is found fellow is matched with the bounding box which



represents detected fellow. That way detected fellows are matched with fellows from the trajectory data. After the fellows are matched, data is written in output file Camera Image Data.



Figure 5.4: Process of matching fellows.

## 6. DATA CORRECTION

Final step after the fellows were matched with trajectory data and left, and right ratios were calculated, is to correct the data. Plan to execute that step is presented at Figure 6.1:

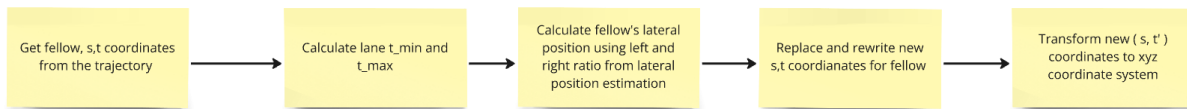


Figure 6.1: Execution plan for data correction.

First step is to load the data from trajectories to get s/t-coordinates for each fellow. That data is then written in the output file called „Camera\_Image\_Data“ containing all the information about the spotted fellow from the image. Before correcting the data, there is one coordinate system that needs to be discussed and that is the road coordinate system. [8] To every road specified in the road network definition document, there is an s/t-type coordinate system assigned. The road reference line defines the s-axis belonging to the (X,Y)-plane of the world coordinate system. The shape of the s-axis line is flat and determined by the geometry of the road projected on the (X,Y)-plane (Z-coordinates equal to 0). The origin of s-coordinates is fixed at the beginning of the road reference line and not affected by an elevation of the road (its inclination in the s-direction). In contrast, multiple t-axes can be defined along the s-axis. Each t-axis points orthogonally leftwards to the s-axis direction and originates on the s-axis at the point with the concerned s-coordinate. All t-axes lie on the surface which is derived from the road surface as if its elevation were not considered. Therefore, t-axes adopt a lateral slope of the road as they are oriented coplanar to the road surface. As the consequence, t-coordinates are not affected by a super elevation of the road.

The following constraints are defined:

- It is assumed the s-axis line has a smooth shape (no leaps nor kinks).
- Being two-dimensional by its nature, the Road coordinate system does not define any vertical positioning. This means positions, specified with (s/t)-coordinates, are on the road surface.
- Both s- and t-coordinates are applicable within boundaries of the respective road only.

- In the case of multiple chained roads, each road defines its own s-axis.
- In the case of roads with a complex lateral profile (for example, uneven surface), applicability and conversion of s/t-coordinates into other coordinate systems might appear problematic or even impossible.

Figure 6.2 shows (s/t)–coordinate system for input road data:

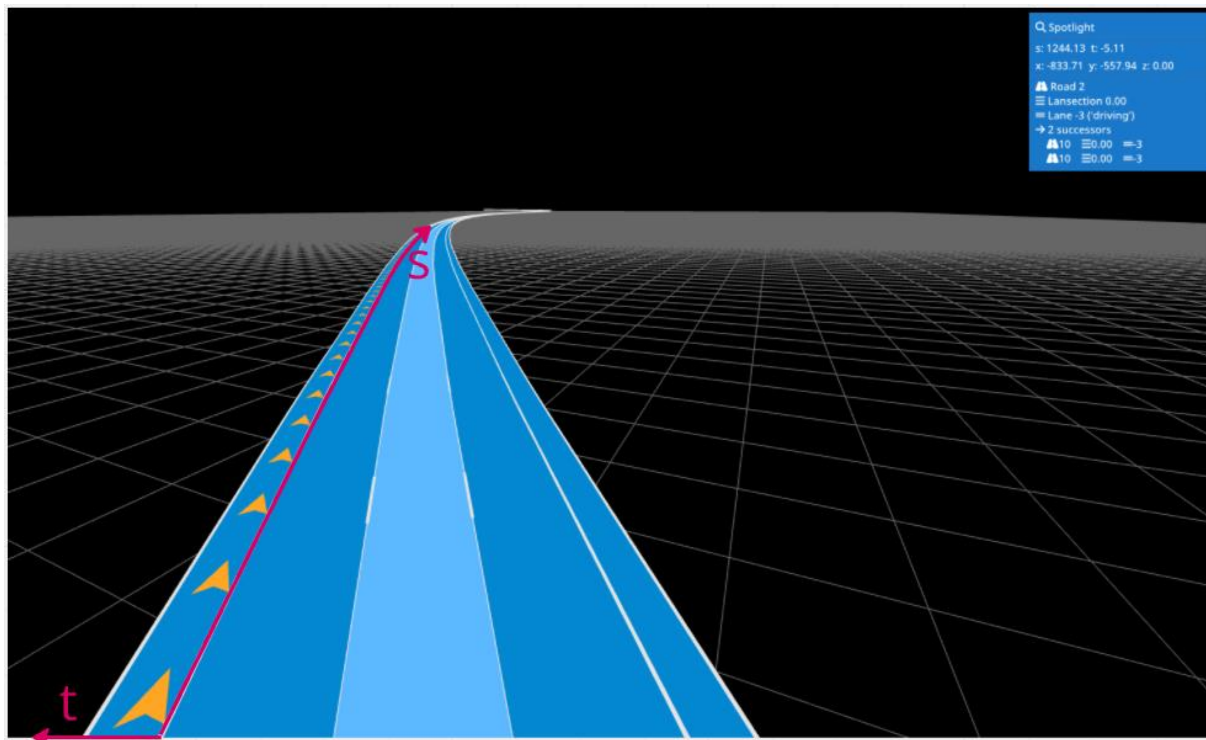


Figure 6.2: (s/t)–coordinate system.

For each lane minimum and maximum value of t-coordinate were extracted from the data and that piece of information is used to calculate errors. For our road data, reference line is on the beginning of leftmost lane meaning that all the values of t for detected fellows are negative. Lane width is calculated as following:

$$width = t_{max} - t_{min} \quad (29)$$

Where :

$t_{\max}$  presents maximum value of the lane coordinate  $t$  and  $t_{\min}$  presents minimum value of lane coordinate  $t$ . For each lane width in meters is calculated, and the next step is to correct  $t$ -coordinates of the fellows because they present lateral offset.

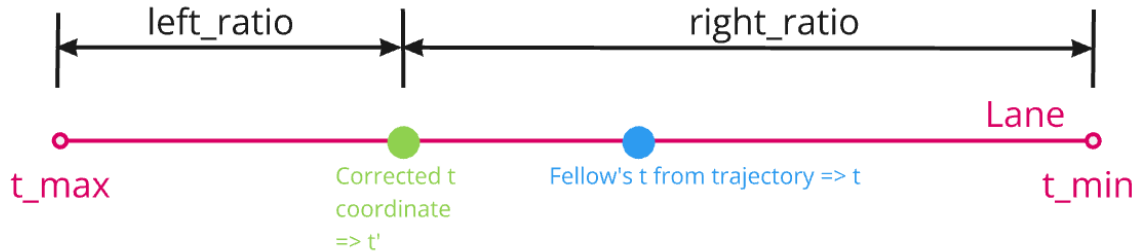


Figure 6.3: Calculation of new  $t$ -coordinate.

Figure 6.3 shows how new  $t$ -coordinate was calculated. New corrected  $t$ -coordinate,  $t'$  was calculated using the ratios that were calculated in the section before for each fellow. The new coordinate was calculated as following:

$$t' = t_{\min} + \text{width} \cdot \text{right\_ratio} \quad (30)$$

By adding product of width and right ratio to minimum value of  $t$  for the lane, new corrected  $t'$  value is calculated for each fellow.

New  $s/t'$  coordinates were then transformed to  $xyz$ -coordinate system, so fellow's position is not only corrected in  $s/t$ - coordinate system but also in  $xyz$ -coordinate system. New coordinates from both coordinate system were then written to output file `Camera_Image_Data`.

## 7. RESULTS

This chapter deals with the results of implementation of the algorithm on two datasets, test dataset consisting of 30 images and whole dataset. First test results will be discussed and visualised. Test results present a testing of the algorithm before whole data was taken as an input.

### 7.1. Results for limited dataset

To make sure algorithm gives good results it was first tested on a smaller dataset. There are two approaches that were tested. First approach that was tested was approach where lateral offsets were estimated via horizontal intersection and second approach that was tested was for lateral offsets that were estimated via „angled“ intersections. Both of those approaches were described in sections 4.1 and 4.2. Results for first approach where lateral offsets were calculated via horizontal intersections is visible at Figure 7.1:

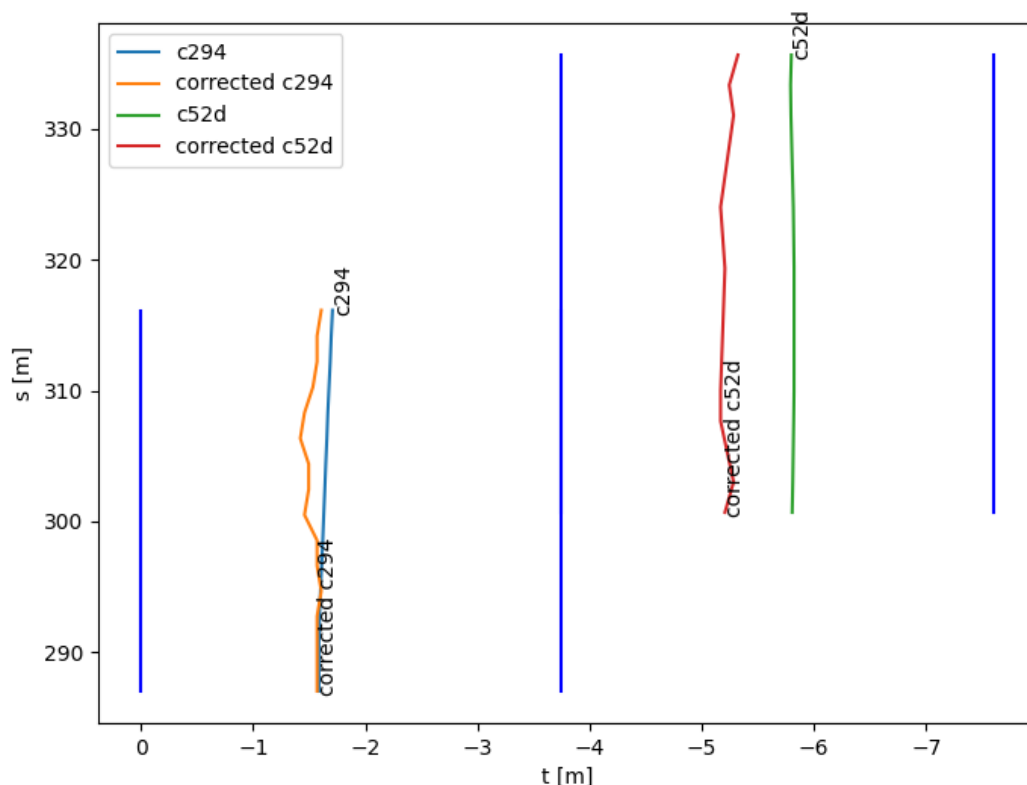


Figure 7.1: Test fellow trajectories for lateral offset estimation via horizontal intersections.

In Figure 7.1 trajectories of fellows for first 30 pictures are visualised. Blue lines represent lane lines. Trajectories of fellows from data are visualised with light blue and green colour while corrected trajectories for those two fellows are visualised with orange and red colour. There are only trajectories of two fellows because other fellows were too far and AI model could not detect them or if they were detected by Object Detection AI module they were not fitted into the lane because they were too far to fit into the polygon. It is also visible in Figure 7.1 that corrected trajectories and trajectories from data are not the same. Corrected trajectories are more to the left than original trajectories that are almost in the middle. Looking at the Figure 7.2 which represents first image it is visible that corrected trajectories present fellow location better because fellows position is not in the middle of the lane but they are positioned more to the left part of the lane. Detected fellows whose trajectories are visible in Figure 7.1 are circled red and green in Figure 7.2.



Figure 7.2: First input image for test trajectories.

Results for second approach where lateral offsets were estimated via „angled“ intersections are visible in Figure 7.3. It is visible that the trajectory for the fellow in the second lane from left has the same trajectory as given in the first approach. Reason for that is that lateral offset for

fellows that are in ego lane is calculated the same in both approaches, via horizontal intersections because it gives better results and estimates positions better. The fellow that is in the first lane from left has a different corrected trajectory comparing to first approach. Corrected trajectory is now even more to the left which describes actual position even better because looking at the Figure 7.2 it is visible that the fellow closest to ego, in the leftmost lane (circled red) is almost completely to the left side of the lane.

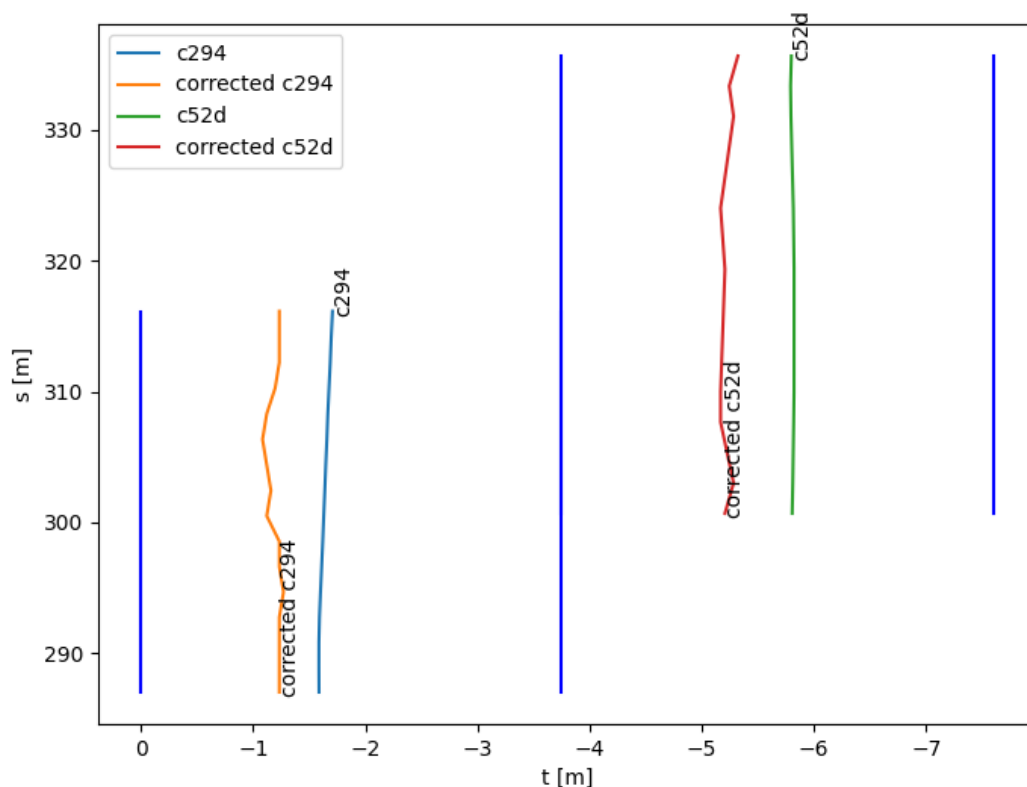


Figure 7.3: Test fellow trajectories for lateral offset estimation via „angled“ intersections.

Last step that was done is data smoothening. Even corrected data may have some outliers that don't fit because of some isolated case, and they need to be smoothened. They were smoothened by Savitzky-Golay filter. [9] The Savitzky-Golay filter, developed by Abraham Savitzky and Marcel J. E. Golay in 1964, is a digital filter widely used for data smoothing and differentiation. Unlike many other smoothing methods, which can distort the signal by blurring its sharp features (take the median or mean filters as an example) the Savitzky-Golay filter excels in maintaining the integrity of the original signal. This makes it particularly useful in applications where preserving the shape and features of the signal is one of the constraints. Data smoothening was done in the python by using *Scipy* library [10]. Test results after data smoothening of fellow

trajectories for lateral offset estimation via horizontal intersection are shown in Figure 7.4, and test results after data smoothening of fellow trajectories for lateral offset estimation via „angled“ intersection is shown in Figure 7.5.

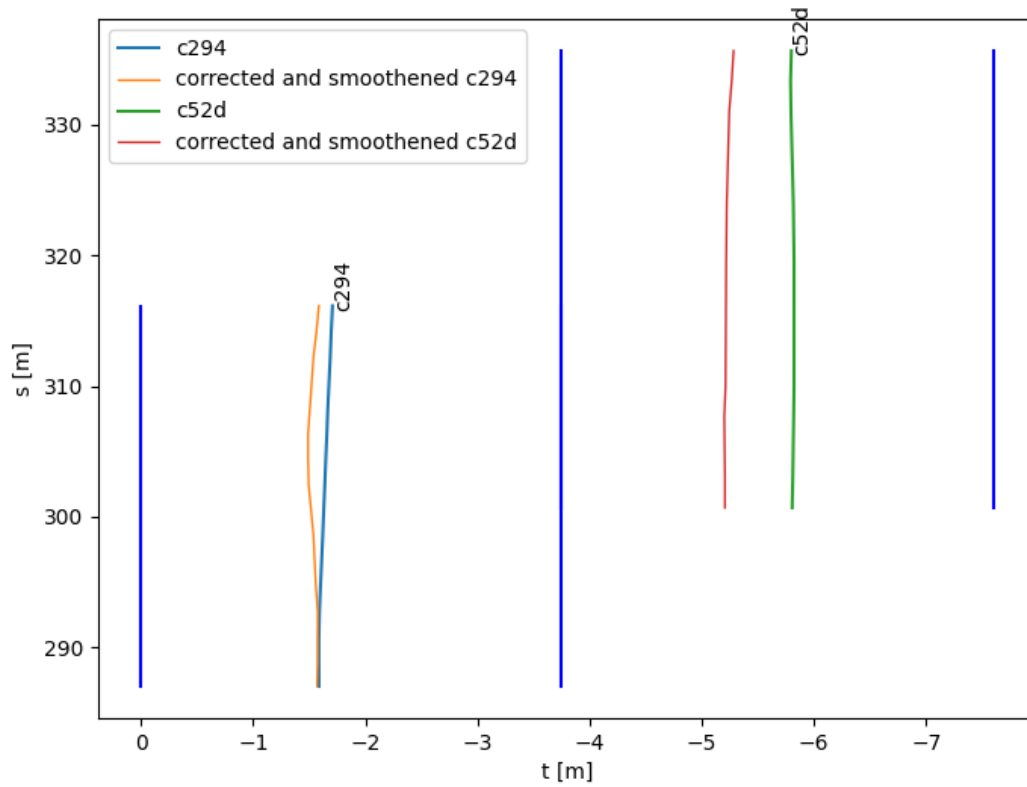


Figure 7.4: Smoothened and corrected test trajectories for lateral offset estimation via horizontal intersections.



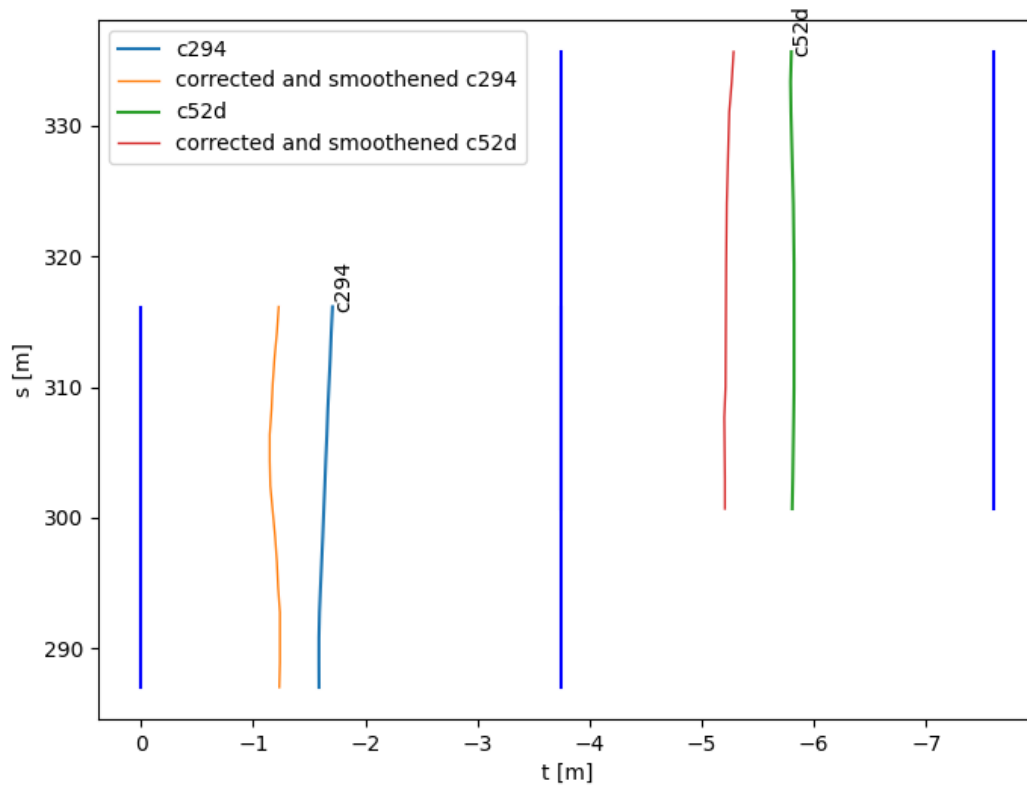


Figure 7.5: Smoothed and corrected test trajectories for lateral offset estimation via „angled“ intersections.

By observing Figure 7.4 and Figure 7.5 it is visible that by using Savitzky-Golay filter trajectories are much smoother which matches better a real life driving on the highway because generally there aren't many lane position changes on the highway.

## 7.2. Results for whole dataset

Next step left to do after testing the algorithm on a test data set, was to test the algorithm on the whole data set which consists of 4180 images. Since showing all the trajectories in one figure is not very organised, and it's quite messy, in figures below trajectories that were calculated from first 250 images were shown while all the other trajectories from all data are included in the appendix. Trajectories of the cars that were recreated using first approach for lateral offset estimation via horizontal intersections is shown in Figure 7.6:

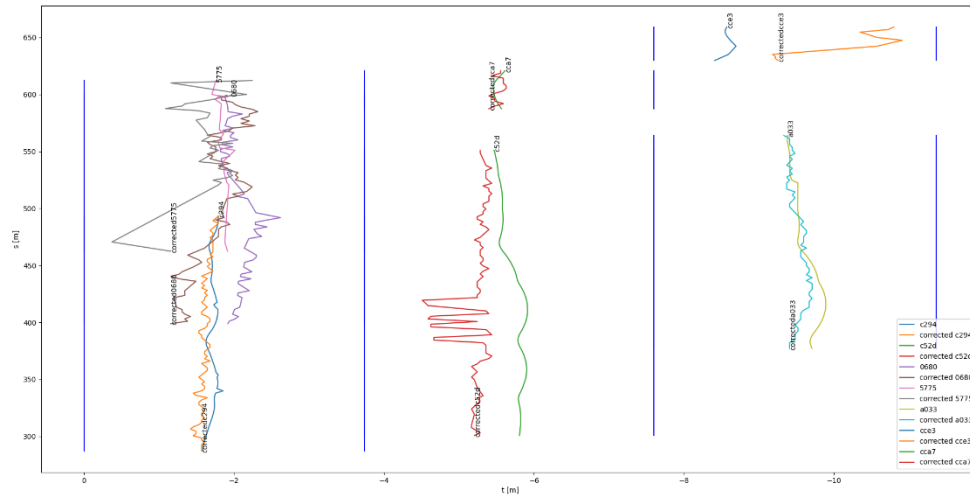


Figure 7.6: Fellow trajectories for lateral offset estimation via horizontal intersections for the first 250 images.

From the Figure 7.6 it is visible that that corrected trajectories are somehow not very similar to original trajectories but by visual inspection it was concluded that corrected trajectories give much better representation of a real world drive in some cases where matching is possible. It is also visible that some parts of corrected trajectories are not corrected well and there are some oscillations. That happens because of errors in trajectories which results in wrong coordinate projections and wrong matching. Case where matching is not correct is shown in Figure 7.7:

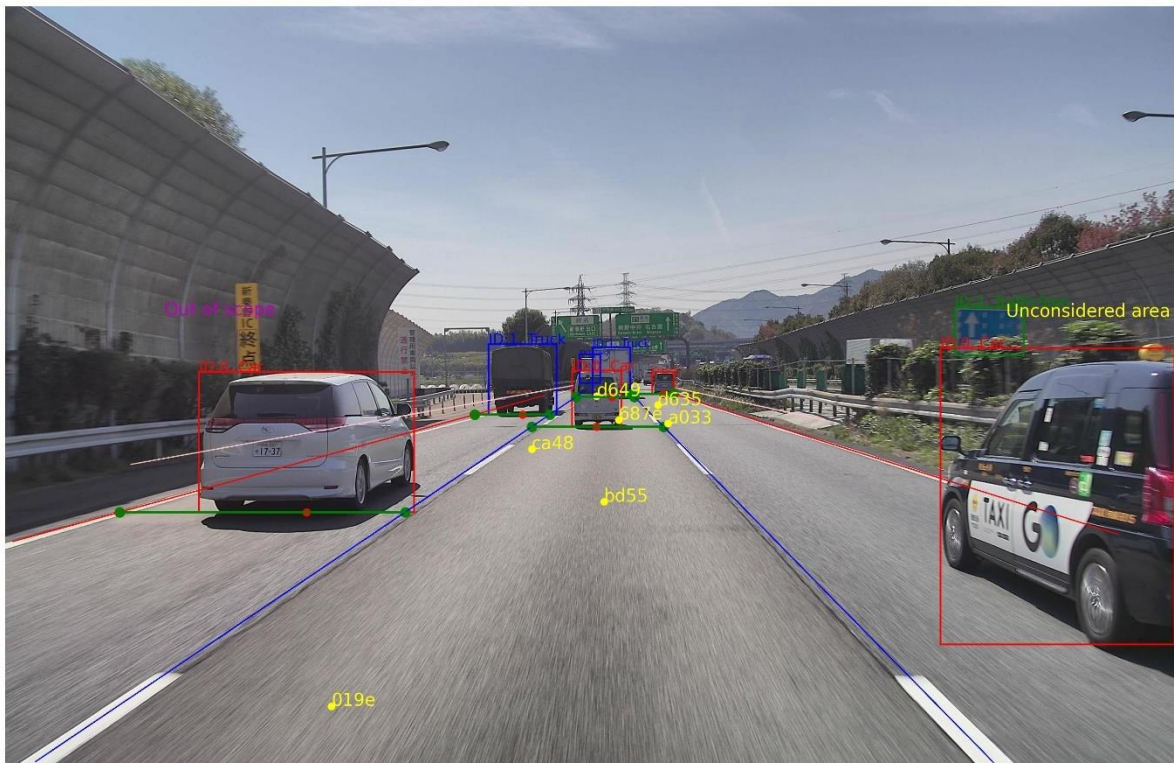


Figure 7.7: Case where object matching is not possible.

When data points from trajectories are wrong, their projection on the image plane is also wrong, and they cannot be matched or are matched incorrectly with detected fellows which results in no trajectory data or incorrect trajectory data.

Trajectories of the fellow vehicles that were created on first 250 images of the dataset using the second approach for lateral offset estimation via „angled“ intersections is shown in Figure 7.8:

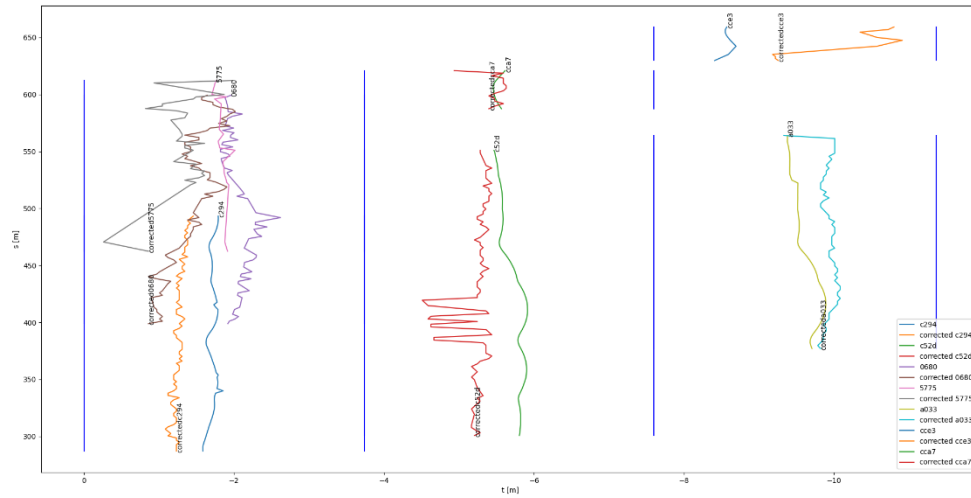


Figure 7.8: Fellow trajectories for lateral offset estimation via „angled“ intersections for first 250 images.

After using Savitzky-Golay filter on corrected trajectories of both methods for lateral offset estimation is shown in Figure 7.9 and Figure 7.10.

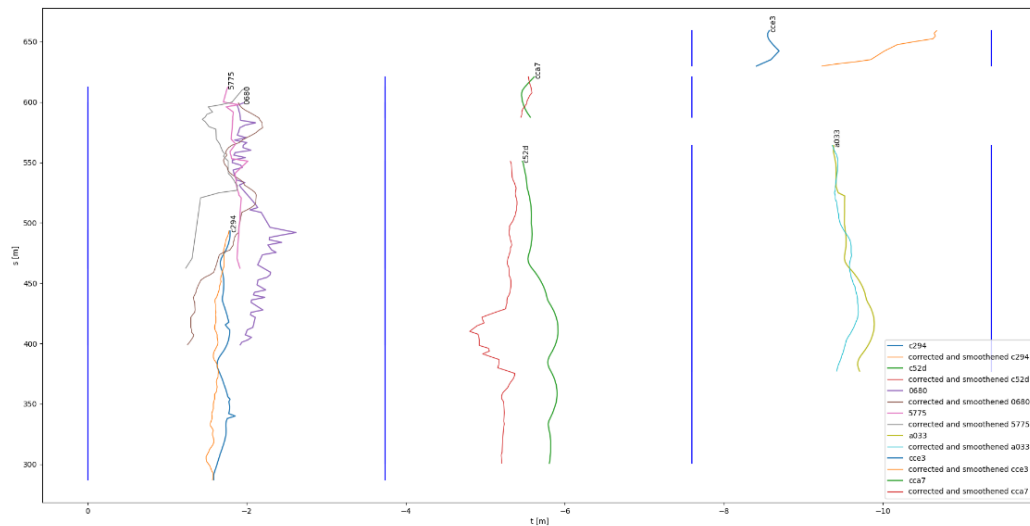


Figure 7.9: Fellow trajectories for lateral offset estimation via „angled“ intersection for first 250 images after using the filter.

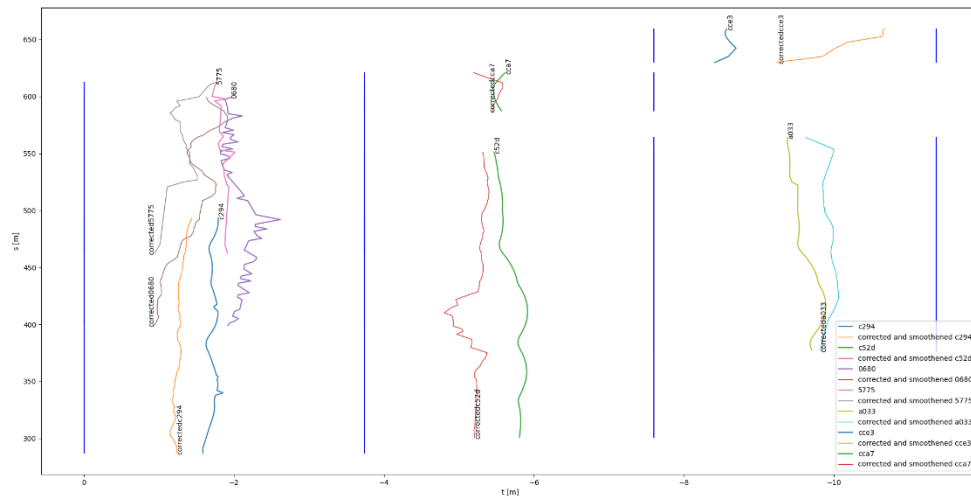


Figure 7.10: Fellow trajectories for lateral offset estimation via „angled“ intersections for first 250 images after filter.

Looking at Figure 7.9 and Figure 7.10 it is visible that fellow trajectories after using the filter are much smoother and give more realistic representation of real-world trajectories.

## 8. CONCLUSION

The main goal of this thesis was to develop an algorithm and implement that algorithm on a real-world data in order to estimate and correct lateral offsets of fellow vehicles. Throughout this master thesis methods that were used in achieving that goal were presented and explained. There are three main methods: lane identification and matching, object matching and lateral offset estimation and correction. Each of those was successfully done, and trajectories were corrected to better match a real-world scenario in some cases.

By visual verification it was concluded that corrected trajectories give more accurate representation of a real-world scenario than original trajectories in some cases where matching is possible. Also, the approach of lateral estimation via „angled“ intersections gives better results than the approach of lateral estimation via horizontal intersections. Another approach to results verification is hardly achievable because there is no ground truth except visual perception and comparing trajectories with the video. Virtual scenarios can be generated using data trajectories and corrected trajectories, and that virtual scenario can be simulated with dSPACE software AURELION, but note that simulation results for original trajectories and corrected trajectories cannot be compared directly because the parameters of the simulator and processes of generating scenarios can introduce changes in trajectories.

By running the algorithm on the whole data set some of the limitations were met. One of the limitations is a situation when there are many vehicles on the highway and one vehicle is behind the other. In that cases coordinates of both vehicles are projected, but one is detected resulting in matching one detected vehicle with two vehicles from the trajectory. That situation is visible in Figure 8.1:



Figure 8.1: Scenario where object matching mistakes occur.

Another case when algorithm could not match detected vehicles with vehicles from the trajectory is when coordinates from the trajectory are wrong and matching is not done meaning trajectories cannot be corrected.

Future improvements of the algorithm could include bounding box rotation to better describe fellow position. Now bounding box represents whole detected object and when that object is not in ego lane bounding box consists of whole vehicle including its side and rear end. Future improvement could find rotation matrix so that bounding box would consist only of the back end of the fellow to estimate lateral position even better. A scene where that could be implemented is visible in Figure 8.2:

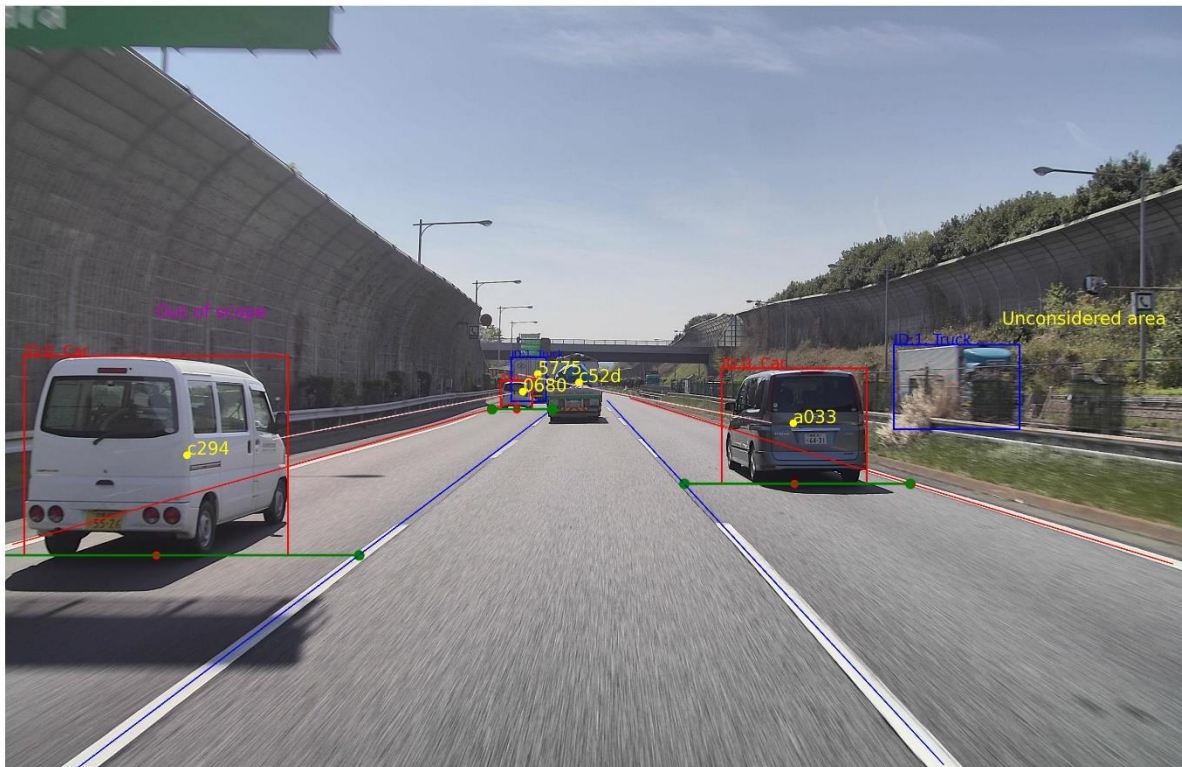


Figure 8.2: Scenario where bounding box rotation could improve lateral position estimation.

Future improvements of the presented approach could also include *tracking* of detected objects in image stream (i.e., across all the recorded frames) for improved object matching, as well as different kind of algorithms for smoothening of the estimated lateral offsets.



**BIBLIOGRAPHY**

- [1] Sievers, Gregor & Seiger, Caius & Peperhowe, Michael & Krumm, Holger & Graf, Sebastian & Hanselmann, Herbert. (2018). Driving Simulation Technologies for Sensor Simulation in SIL and HIL Environments.
- [2] Camera for data collection. <https://www.sensing-world.com/en/pd.jsp?id=24> (accessed: February 3, 2025)
- [3] Odrviewer. <https://odrviewer.io/> (accessed: February 3, 2025)
- [4] ONNX Runtime in Python. <https://onnxruntime.ai/docs/get-started/with-python.html> (accessed: February 3, 2025)
- [5] ONNX format. <https://medium.com/@shivprataprai11/understanding-onnx-an-open-standard-for-deep-learning-models-350a72714660> (accessed: February 3, 2025)
- [6] DBSCAN. <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html> (accessed: February 3, 2025)
- [7] Ransac regressor. [https://scikit-learn.org/dev/modules/generated/sklearn.linear\\_model.RANSACRegressor.html](https://scikit-learn.org/dev/modules/generated/sklearn.linear_model.RANSACRegressor.html) (accessed: February 3, 2025)
- [8] Coordinate system s/t. [https://publications.pages.asam.net/standards/ASAM\\_OpenSCENARIO/ASAM\\_OpenSCENARIO\\_XML/latest/06\\_general\\_concepts/06\\_03\\_coordinate\\_systems.html#\\_road\\_coordinate\\_system\\_st](https://publications.pages.asam.net/standards/ASAM_OpenSCENARIO/ASAM_OpenSCENARIO_XML/latest/06_general_concepts/06_03_coordinate_systems.html#_road_coordinate_system_st) (accessed: February 3, 2025)
- [9] Savitzky-Golay Filter. <https://medium.com/pythoneers/introduction-to-the-savitzky-golay-filter-a-comprehensive-guide-using-python-b2dd07a8e2ce> (accessed: February 3, 2025)
- [10] Scipy library. [https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.savgol\\_filter.html](https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.savgol_filter.html) (accessed: February 3, 2025)
- [11] T. Menzel, G. Bagschik, L. Isensee, A. Schomburg and M. Maurer, "From Functional to Logical Scenarios: Detailing a Keyword-Based Scenario Description for Execution in a Simulation Environment," 2019 IEEE Intelligent Vehicles Symposium (IV), Paris, France, 2019, pp. 2383-2390, doi: 10.1109/IVS.2019.8814099.

- 
- [12] Camera intrinsics and extrinsics. <https://towardsdatascience.com/what-are-intrinsic-and-extrinsic-camera-parameters-in-computer-vision-7071b72fb8ec> (accessed: February 3, 2025)
- [13] Camera intrinsics and extrinsics. <https://leimao.github.io/blog/Camera-Intrinsics-Extrinsics/> (accessed: February 3, 2025)
- [14] Matrix transformations. <https://sigmoidal.ai/en/matrix-transformations-and-coordinate-systems-with-python/> (accessed: February 3, 2025)
- [15] Python point rotation. <https://stackoverflow.com/questions/14842090/rotate-line-around-center-point-given-two-vertices> (accessed: February 3, 2025)

---

**APPENDICES****APPENDIX A: Data examples****Trajectory data example for one key:**

```
{
"1680581868.0": [
{
"object_id": "EGO,6e54d6e8-74c6-41cb-aa31-005a3c925e84",
"road_position": {
"road_ID": 2,
"section_index": 0,
"lane_id": -3.0,
"road_s": 240.95532913104938,
"road_t": -5.822240584283497},
"world_position": {
"x": 0.01,
"y": 0.01,
"z": 0.75,
"yaw": -2.66,
"pitch": -0.01,
"roll": 0.0}},
{
"object_id": "06804c38-6c91-446b-bd7b-9614a44bf865",
"road_position": {
"road_id": 2,
"section_index": 0,
"lane_id": -2.0,
"road_s": 342.1832673695808,
"road_t": -2.066811300018098},
"world_position": {
"x": -87.15,
"y": -52.24,
```

```
"z": -0.83,
"yaw": -2.62,
"pitch": -0.0,
"roll": 0.01}}
```

### Example of Road Network data for road part 2:

```
"2": {
  "sections": {
    "0": {
      "driving_lanes": [-2, -3,-4],
      "lanes": {
        "0": {"type": "none"},
        "-1": {"type": "shoulder"},
        "-2": {"type": "driving"},
        "-3": {"type": "driving"},
        "-4": {"type": "driving"},
        "-5": {"type": "shoulder"}
      }
    }
  }
}}
```

### Example of Camera image data for one frame:

```
{'detected_objects': {'#1': {'bounding_box': (2086, 799, 2503, 1038),
  'class': 'Truck',
  'mapping_data': {'associated_lane_id': "",
    'lane_identification_case': 'unconsidered',
    'matches_to_annotation_object': 'MatchNotFound',
    'matching_road_lanes': {1: -2,
      2: -3,
      3: -4},
    'reference_point': 'ReferencePoint(2295, '
      '1038)'},
    'score': '0.79898787'}},
  '#2': {'bounding_box': (1160, 906, 1266, 1017),
```

```

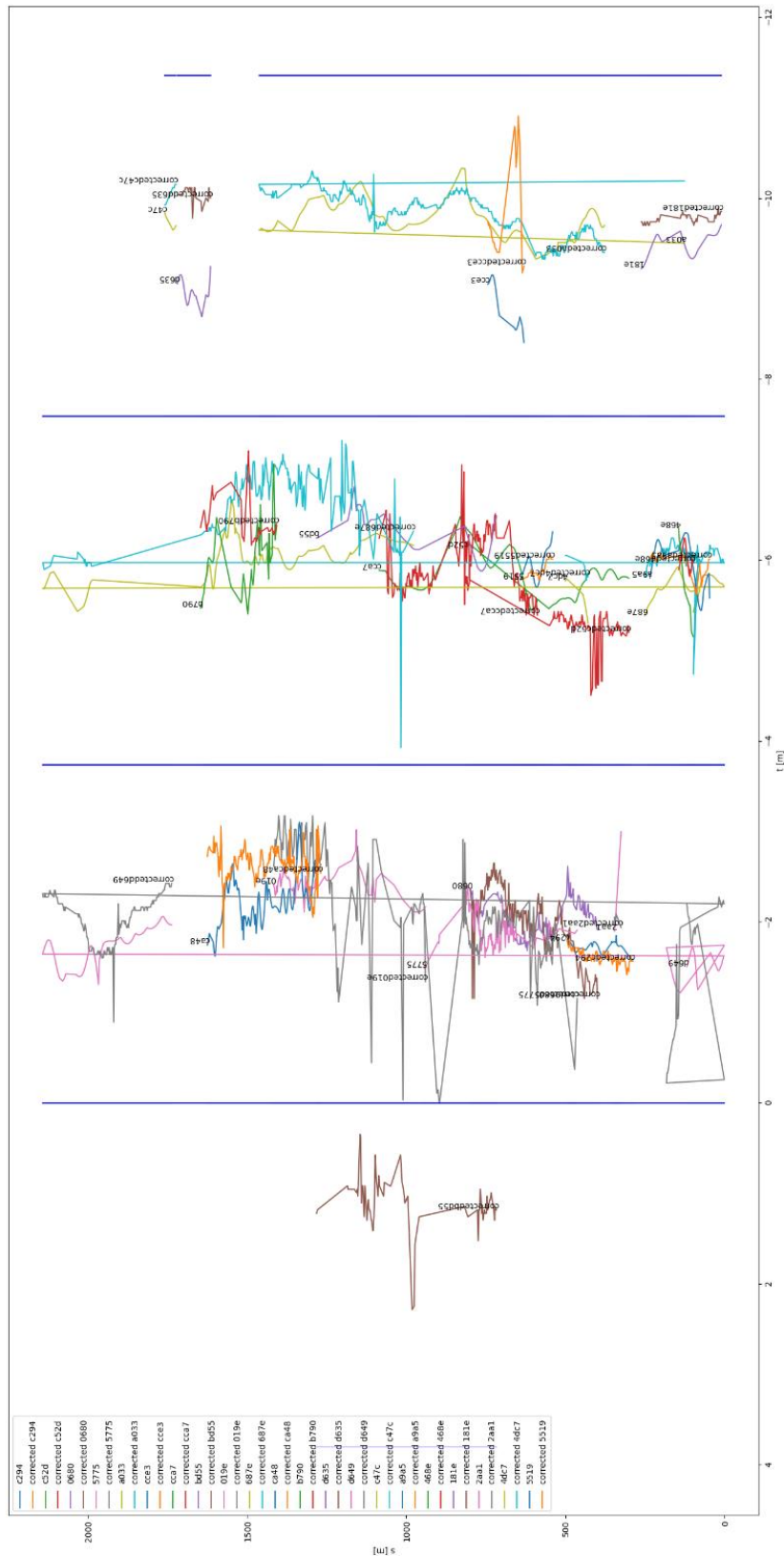
'class': 'Car',
'mapping_data': {'associated_lane_id': 1,
                  'lane_identification_case': 'Identified',
                  'lane_t_max': -3.739450918814552,
                  'lane_t_min': 0.0,
                  'lane_width': 3.739450918814552,
                  'left_lane_boundary_point': (1114,
                                               1016),
                  'left_ratio': 0.42,
                  'matches_to_annotation_object': 'c294421a-7fe5-47d6-81f2-
690a497aa3de',
                  'matching_road_lanes': {1: -2,
                                           2: -3,
                                           3: -4},
                  'reference_point': 'ReferencePoint(1213, '
                                     '1017)',
                  'right_lane_boundary_point': (1350,
                                               1016),
                  'right_ratio': 0.58,
                  'road_network_element': '2_0',
                  'road_s': 287.0148362223681,
                  'road_t': -1.5879434167702944,
                  'road_t_corrected': -1.570569385902112,
                  'road_t_delta': 0.017374030868182455,
                  'xyz_corrected': (-38.9,
                                    -25.37,
                                    0.0)},
'score': '0.7557713'},
'#3': {'bounding_box': (1382, 858, 1506, 996),
       'class': 'Truck',
       'mapping_data': {'associated_lane_id': 2,
                        'lane_identification_case': 'Identified',
                        'lane_t_max': -7.598524542478597,
                        'lane_t_min': -3.739450918814554,

```

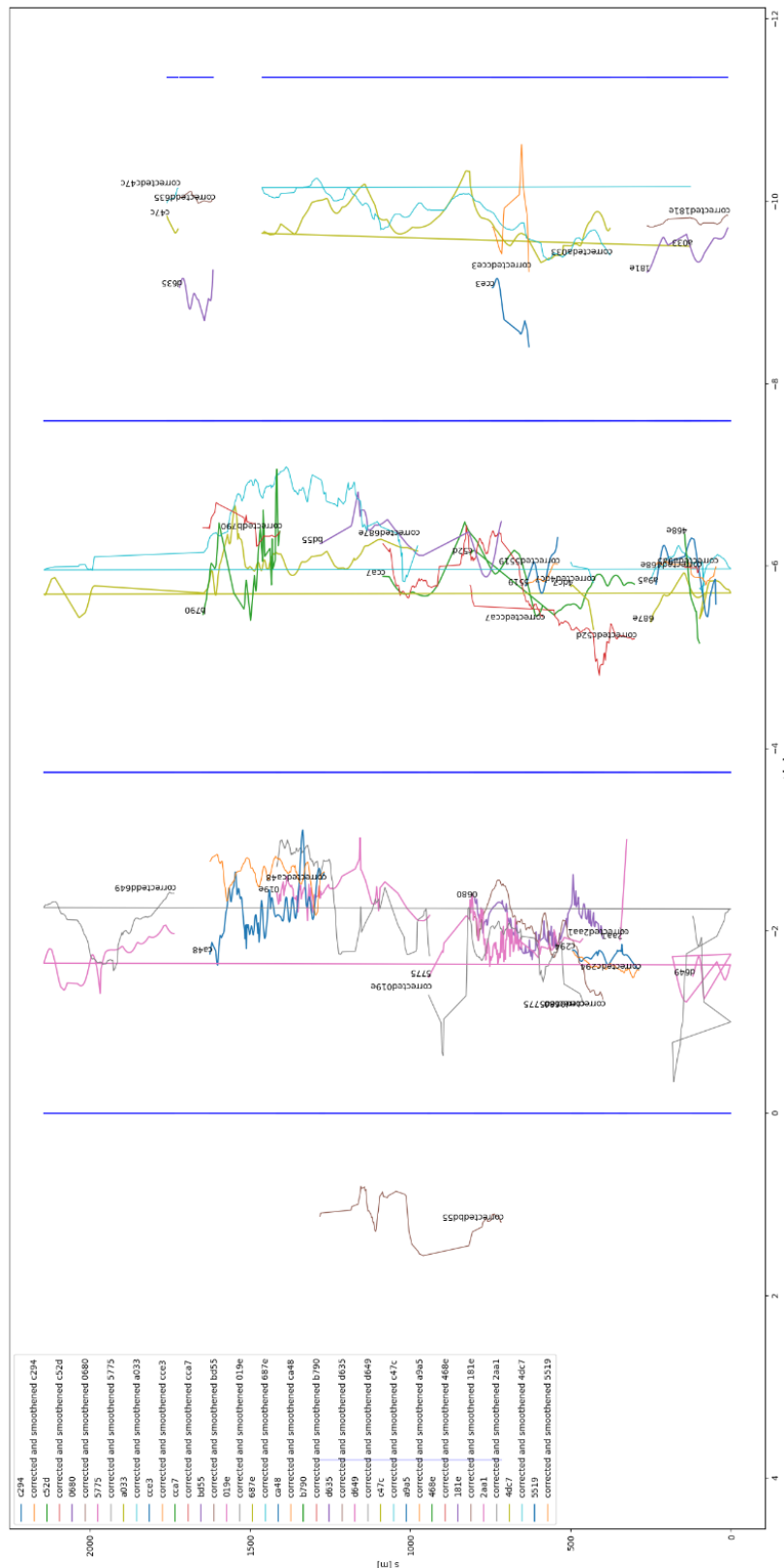
```
'lane_width': 3.8590736236640435,
'left_lane_boundary_point': (1371,
                             996),
'left_ratio': 0.38,
'matches_to_annotation_object': 'c52d1b47-66f6-494f-a81e-
c26727926971',
'matching_road_lanes': {1: -2,
                        2: -3,
                        3: -4},
'reference_point': 'ReferencePoint(1444, '
                   '996)',
'right_lane_boundary_point': (1560,
                              996),
'right_ratio': 0.62,
'road_network_element': '2_0',
'road_s': 300.68463767966875,
'road_t': -5.80533166262987,
'road_t_corrected': -5.20589889580689,
'road_t_delta': 0.5994327668229795,
'xyz_corrected': (-52.92,
                  -28.4,
                  0.0)},
'score': '0.6120546'}},
'image_frame': '1680581868050000.jpg',
'matching_trajectory_timestamp': 1680581868.0,
'observation_case': 'case 3: both barriers are detected'}
```

## APPENDIX B: Trajectories

Corrected trajectories with lateral offset estimation via horizontal intersections:

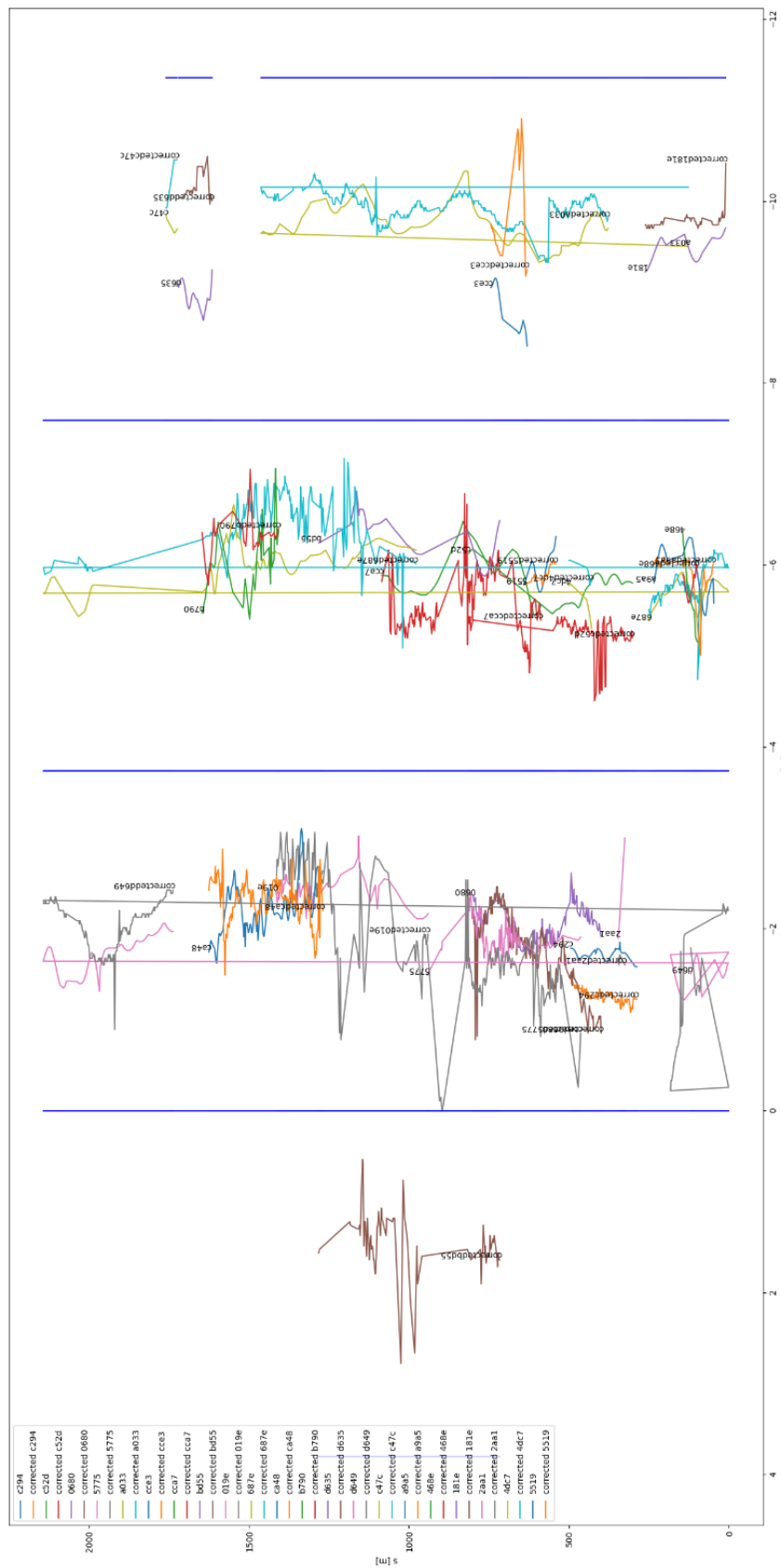


Corrected trajectories with lateral offset estimation via horizontal intersections, corrected with Savitzky-Golay filter:





Corrected trajectories with lateral offset estimation via „angled“ intersections:



Corrected trajectories with lateral offset estimation via „angled“ intersections, corrected with Savitzky-Golay filter:

