University of Khartoum

Faculty of Engineering
Department of Civil Engineering

INFLUENCE OF ROAD DEFICIENCIES ON VEHICLE TRAJECTORIES USING ARTIFICIAL INTELLIGENCE BASED SYSTEM

A thesis submitted in partial fulfilment of the requirement of the B.Sc. degree in Civil engineering

Submitted by

Abdalkhalig Elfatih Salih Ahmed	**163056**
Ahmed Abdalgadir Mohammed Turkman	**163016**
MuhammadAlmujtaba Abdulsalam MuhammadAhmad Yaseen	**163117**

Subervised by

Dr. Shaza Farouk Azhari Hassan
Prof. Montasir Abbas

September 2023

Abstract

Road deficiencies, such as potholes, and cracks, can significantly disrupt traffic flow, causing congestion, accidents, and contributing to environmental problems due to increased emissions, and noise. The estimation of the impact of these deficiencies on a traffic system is of particular importance for countries with limited budgets allocated to infrastructure improvement, because in such case, it is possible to only fix a few of these deficiencies. Artificial Intelligence (AI) systems, which can leverage existing data sources, offer a cost-effective solution to this problem.

In this study, the aim is to design an AI system capable of estimating vehicle trajectories, including variables such as distance, speed, and acceleration, from video data. The primary objective was to use these estimated trajectories to analyze and quantify the impact of road deficiencies on traffic patterns. This was achieved by comparing the trajectories of vehicles under normal road conditions with those resulting from the presence of road deficiencies.

The results indicate that road deficiencies significantly alter vehicle trajectories, leading to inefficient traffic flow and potential safety hazards. The AI system was able to accurately estimate vehicle trajectories and detect changes in these trajectories related to road deficiencies.

However, it is important to note that the success of this system is dependent on the accuracy of the used tracking system. Therefore, it is recommended to employ several trackers, including but not limited to StrongSORT, to compare their performance and identify the most effective. In future work, it would also be beneficial to use instrumented vehicles, which can provide more accurate and consistent data for testing. Moreover, the use of drones for video capturing is advisable to overcome common issues encountered with ground-level video capture, such as perspective distortion and occlusion effects. Using drones could provide a more comprehensive view of the traffic system and allow for more accurate trajectory estimation.

In conclusion, this study demonstrates the potential of AI systems in analyzing the impact of road deficiencies on traffic flow. The findings highlight the importance of ongoing research in this area, particularly in the development and testing of advanced tracking systems and data capture methods. These advancements could significantly improve the ability to identify and rectify road deficiencies, ultimately leading to safer and more efficient roadways.

المستخلص

يمكن أن تؤدي أوجه القصور في الطرق، مثل الحفر والشقوق، إلى تعطيل تدفق حركة المرور بشكل كبير، مما يتسبب في الازدحام والحوادث والمساهمة في مشاكل بيئية بسبب زيادة الانبعاثات والضوضاء. إن تقدير تأثير أوجه القصور هذه على نظام المرور له أهمية خاصة بالنسبة للبلدان ذات الميزانيات المحدودة المخصصة لتحسين البنية التحتية، لأنه في مثل هذه الحالة، من الممكن إصلاح عدد قليل فقط من أوجه القصور هذه. توفِر أنظمة الذكاء الاصطناعي، التي يمكنها الاستفادة من مصادر البيانات الحالية، حلاً فعالاً من حيث التكلفة لهذه المشكلة

الهدف في هذه الدراسة هو تصميم نظام ذكاء اصطناعي قادر على تقدير مسارات المركبات، بما في ذلك المتغيرات مثل المسافة والسرعة والتسارع، من بيانات الفيديو. كان الهدف الأساسي هو استخدام هذه المسارات المقدرة لتحليل وقياس تأثير أوجه القصور في الطرق على أنماط حركة المرور. وقد تم تحقيق ذلك من خلال مقارنة مسارات المركبات في ظل ظروف الطريق العادية مع تلك الناتجة عن وجود عيوب في الطريق

وتشير النتائج إلى أن أوجه القصور في الطرق تغير بشكل كبير مسارات المركبات، مما يؤدي إلى تدفق حركة مرور غير فعال ومخاطر محتملة على السلامة. وتَمكَّن نظام الذكاء الاصطناعي من تقدير مسارات المركبات بدقة واكتشاف التغييرات في هذه المسارات بسبب عيوب الطريق

ومع ذلك، من المهم ملاحظة أن نجاح هذا النظام يعتمد على دقة برنامج التتبع المستخدم. لذلك، يوصى باستخدام العديد من برامج التتبع، بما في ذلك على سبيل المثال لا الحصر، StrongSORT، لمقارنة أدائها وتحديد الأكثر فعالية. وفي العمل المستقبلي، سيكون من المفيد أيضًا استخدام المركبات المجهزة، والتي يمكن أن توفر بيانات أكثر دقة واتساقًا للإختبار. علاوة على ذلك، يُنصح باستخدام طائرات بدون طيار لالتقاط الفيديو للتغلب على المشكلات الشائعة التي تواجه التقاط الفيديو على مستوى الأرض، مثل تشويه المنظور وتأثير الانسداد. يمكن أن يوفر استخدام الطائرات بدون طيار رؤية أكثر شمولاً لنظام المرور ويسمح بتقدير المسار بشكل أكثر دقة

في الختام، توضح هذه الدراسة إمكانات أنظمة الذكاء الاصطناعي في تحليل تأثير عيوب الطرق على تدفق حركة المرور. وتسلط النتائج الضوء على أهمية البحث المستمر في هذا المجال، وخاصة في تطوير واختبار أنظمة التتبع المتقدمة وطرق التقاط البيانات. يمكن لهذه التطورات أن تحسن بشكل كبير القدرة على تحديد وتصحيح أوجه القصور في الطرق، مما يؤدي في النهاية إلى طرق أكثر أمانًا وكفاءة

Table of Contents

List of Figures

List of Tables

Acknowledgements

We would like to sincerely thank our supervisors Dr. Shaza Azhari, and Prof. Montasir Abbas, for their support, and guidance. They did their best to extract the best version of this project.

We also would like to express our deepest gratitude to Dr. Awad Abdelhalim, for his generosity, deliberate instructions, and patience. Without his help, this project definitely wouldn't see the light.

Any remaining imperfections are completely on us.

Chapter 1

Introduction

1.1 BACKGROUND

Today about 55% of the world's population lives in cities, with that number expected to rise over the next few decades. This increase in population density is accompanied by an increase in the number of vehicles on the road.[1]

The ability to provide an accurate estimate of the speed of road vehicles is a key feature of Intelligent Transport Systems (ITS). This requires tackling problems such as synchronized data recording, representation, detection and tracking, distance and speed estimation, etc., and it can be addressed using data from sensors of different nature (e.g. radar, laser, or cameras) and under different lighting and weather conditions. This is a well-known topic in the field with considerable contributions from both academia and industry. The most important application domains are speed limit enforcement, traffic monitoring and control, and, more recently, autonomous vehicles. However, it is important to note that the nature of the problem of speed detection differs in a not insignificant way, depending on whether it is done from the infrastructure (fixed sensors) or a mobile platform (intelligent/autonomous vehicles or mobile speed cameras). Recent advances in computer vision techniques have led to a significant increase in the number of works proposing the use of vision as the only mechanism for measuring vehicle speed. The challenge of making accurate estimations of distances and speeds arises from the discrete nature of video sensors that project the 3D world into a 2D discrete plane. This intrinsic limitation results in a digital representation whose accuracy follows an inverse-square law, that is, its quantity is inversely proportional to the square of the distance from the camera to the vehicle. Despite these limitations, the potential benefits of using video cameras are remarkable, since they are a cost-efficient solution compared with range sensors such as microwave Doppler radars or laser-based devices. The possibility of using already installed traffic cameras without the need to integrate new sensors is another advantage. Also, the visual cues from these sensors can be used to address other problems such as automatic identification of the vehicle (license plate, make, model, and color), re-identification in different places, and if the resolution allows, the identification of occupants, or seat belt use. The vehicles, or some of their representative features, must be detected in all the available images. Tracking the vehicle or its features over time is essential to obtain speed measurements. Different approaches can be applied to address both tasks. Speed estimation intrinsically involves the estimation of distances with associated timestamps. Different methods exist to calculate the relative distance of vehicles from some global reference, as well as different approaches to finally calculate the speed of the vehicle.

1.2 PROBLEM STATEMENT

Sudan deeply suffers from road deficiencies that negatively impact the traffic flow, causing delays, congestion, and sometimes even accidents. And with the limited budget available to fix these deficiencies, the infrastructure department is only capable of fixing a few of them every while. This study estimates the vehicles' trajectories (distance, speed, and acceleration), which can then be used to estimate the degree of impact each deficiency causes to the traffic flow, at a very low cost, using the traffic cameras already installed, or capturing videos of suitable lengths featuring the targeted deficiency, then, this analysis could be used to determine which deficiencies to prioritize their fixation over the others, given their higher degree of impact on the traffic flow.

Given that AI systems normally require low costs to operate, it's very practical for a country like Sudan to start adopting AI to help tackling some of the pressing problems it suffers from.

1.3 AIM AND OBJECTIVES

The aim of this study is to utilize artificial intelligence techniques in order to accurately estimate the trajectories of vehicles. To achieve this, the following objectives were considered:

1. To develop AI system to estimate vehicles' trajectories (distance, speed, and acceleration) from videos, using the computer vision model YOLO with StrongSORT tracker.
2. To determine and analyse the impact of road deficiencies on vehicles' trajectories.

3. To compare vehicle trajectories under ideal road conditions, with those observed on roads with deficiencies.

1.4 SCOPE OF STUDY

The scope of this study focuses on the development of a machine learning system that aims to estimate various aspects of vehicles' trajectories, including distance, speed, and acceleration. The study solely considers video data as the input type for training and testing the model. However, it is important to note that this research does not take into account the potential influence of factors such as video quality and lighting conditions on the accuracy and performance of the developed model. Additionally, it is crucial to highlight that the study's applicability is limited to straight roadways, excluding any horizontal curves or turns in the analysis. It's also noteworthy to mention that this study does not differentiate between various types of vehicles - SUVs, Sedans, Motorbikes, and others are all collectively referred to as "cars" in this content.

1.5 THESIS OUTLINE

This thesis contains five chapters, the rest four chapters are organized as follows: Chapter 2 contains the literature review about the vehicle's speed estimation problem, deep learning, and its applications in transportation, in addition to details about the softwares used in the study. Chapter 3 contains a detailed description of this research problem and the methods and algorithms used for tackling this problem, in addition to the description of the development process of the inference system. Chapter 4 contains the detailed results and findings of the experiments done in chapter 3. Chapter 5 contains an overall conclusion and summary of the thesis in addition to recommendations and suggestions for future work.

Chapter 2

Literature Review

2.1 THE VEHICLE'S SPEED ESTIMATION PROBLEM

Vehicle speed estimation is the process of estimating the speed of a vehicle based on various data sources and sensors. The problem of vehicle speed estimation arises when the vehicle's actual speed differs from the speed estimated by the system.

One of the earliest approaches for vehicle speed estimation was edge detection. Edge detection techniques, such as the Canny edge detector, were used to extract the edges from an image. The edges were then analysed to estimate the vehicle's speed. This approach was limited by its reliance on the accuracy of the edge detection algorithm and the ability to distinguish between edges that correspond to vehicles and those that do not.[2]

another method used was Object proposal generation. it is a method that generates a set of regions of interest (RoIs) in an image, which can be used to estimate the speed of vehicles. The basic idea is to generate a set of RoIs that contain objects (e.g., vehicles) and then use these RoIs to estimate the speed.

Feature extraction is another method that extracts useful information from an image or a video sequence. The extracted features can then be used to estimate the speed of vehicles. Feature extraction methods can be broadly classified into two categories: hand-crafted and deep learning-based methods.

Model-based methods were also used for vehicle speed estimation. Model-based methods involve using a mathematical model of the vehicle and the imaging process to estimate the speed of the vehicle. These methods can be based on physical laws, such as the law of motion, or machine learning models, such as support vector machines (SVMs) or neural networks.

One of the earliest model-based methods for speed estimation is the Kalman filter (KF).[3] The KF is a mathematical method that uses a state-space model of the vehicle and its dynamics to estimate the vehicle's state, including speed. The KF has been widely used in various applications, including vehicle speed estimation, due to its computational efficiency and simplicity. However, the KF relies on a linearized vehicle model and assumes a Gaussian noise distribution, which can limit its accuracy in non-linear driving scenarios.

2.2 DEEP LEARNING

Deep learning (DL) is a subfield of machine learning (ML) that focuses on developing algorithms and models that can learn and improve on their own by automatically extracting features from data.[4] DL algorithms are designed to process large amounts of data, and they can learn complex patterns and relationships in the data, even when the data is unstructured or semi-structured.

2.2.1 Deep Learning for Computer Vision

Deep learning has revolutionized the field of computer vision in recent years. The advent of large-scale datasets, powerful computational resources, and advances in algorithms have enabled the development of sophisticated models that can learn and represent complex patterns in visual data. In this literature review, we will survey some of the key works in deep learning for computer vision, focusing on their contributions, strengths, and limitations.

One of the earliest and most influential works in deep learning for computer vision is the AlexNet model.[5] This model achieved state-of-the-art performance on the ImageNet dataset, a large-scale benchmark for image classification. The AlexNet model is built upon a deep convolutional neural network (CNN) architecture, which leverages the concept of transfer learning to learn hierarchical representations of images. The authors demonstrated that by pre-training the model on a large dataset of images, the model can learn to recognize objects and achieve high accuracy on a variety of downstream tasks.

Since the introduction of AlexNet, numerous deep learning models have been proposed for computer vision tasks, including object detection, segmentation, and generation. One of the most popular and widely-used architectures is the Faster R-CNN (Region-based Convolutional Neural Network).[6] This model extends the basic CNN architecture with a region proposal network (RPN) that generates a set of regions of interest (RoIs) in an image, followed by a CNN that classifies and refines the bounding boxes of the detected objects. The Faster R-CNN model achieved state-of-the-art performance on object detection tasks and has been widely adopted in various applications, including autonomous driving, surveillance, and medical imaging.

Another important contribution to deep learning for computer vision is the Generative Adversarial Networks (GANs).[7] GANs consist of a generator network that produces synthetic images and a discriminator network that tries to distinguish between real and synthetic images. Through adversarial training, the generator learns to produce high-quality images that are indistinguishable from real images, while the discriminator becomes increasingly effective at distinguishing between real and synthetic images. GANs have been applied to various computer vision tasks, including image synthesis, data augmentation, and style transfer.

In addition to these works, there are several other notable deep learning models and techniques that have contributed to the advancement of computer vision. The Convolutional LSTM (Long Short-Term Memory) network introduces the concept of spatial and temporal convolutions, enabling the model to learn both spatial and temporal patterns in video data. [8] The Inception network introduces the concept of multi-scale features, allowing the model to capture features at different scales and aspect ratios. [9] The Residual Network (ResNet) introduces the concept of residual learning, enabling the model to learn much deeper networks with less vanishing gradients. [10]

While deep learning models have achieved remarkable performance in computer vision tasks, they are not without limitations. One of the major challenges is the lack of interpretability, making it difficult to understand why the model is making certain predictions or decisions. Another challenge is the requirement for large amounts of labeled data, which can be time-consuming and expensive to obtain. Finally, there are concerns about the ethical implications of using deep learning models, such as bias, privacy, and security.

In conclusion, deep learning has revolutionized the field of computer vision in recent years. The development of sophisticated models and algorithms has enabled the recognition of complex patterns in visual data, leading to significant advances in various computer vision tasks. While there are challenges and limitations associated with deep learning models, they have shown great promise and potential in various applications, including autonomous driving, surveillance, medical imaging, and more.

As the field continues to evolve, we can expect to see further advances in deep learning for computer vision.

2.2.2 The Use of Deep Learning in Transportation

Deep learning has been increasingly applied to various transportation-related problems in recent years. Here are some of the key ways in which deep learning is being used in transportation:

Traffic prediction

Traffic prediction is a crucial component of intelligent transportation systems, and deep learning methods have gained significant attention in this area. Existing traffic prediction methods can be categorized into traditional methods and deep learning-based methods. Deep learning models can capture complex patterns and relationships in traffic data using neural networks with multiple layers.

State-of-the-art deep learning-based approaches have been applied to various traffic prediction applications, including traffic speed prediction, traffic volume prediction, traffic incident detection, and traffic route planning. These approaches have shown promising results in improving traffic prediction accuracy and reducing computational complexity.

However, there are still several open challenges in this field, such as the lack of high-quality traffic data, difficulty in capturing non-linear relationships between different regions, and the need for better interpretability and explainability of deep learning models. [11]

Autonomous vehicles

Deep learning networks (DLNs) have become a crucial component in the development and implementation of autonomous driving systems. In recent years, there have been significant advancements in both software and hardware tools used for the development and deployment of DLNs in self-driving cars.

One notable software tool is the DGX-1 supercomputer, which is specifically designed for training DLNs. This powerful tool enables the rapid development and testing of complex DLNs, allowing researchers and engineers to iterate and refine their models more quickly and efficiently.

Another important tool is the Drive PX 2, a scalable AI car computer platform for inference. This platform enables the deployment of trained DLNs in real-world driving scenarios, allowing self-driving cars to make decisions in real-time based on the data they collect from their sensors.

Looking to the future, the recently announced Xavier, an AI supercomputer on a chip for future autonomous vehicles, promises to revolutionize the field even further. This powerful tool will enable the integration of DLNs into the vehicle itself, allowing for even faster and more efficient processing of sensor data.

The effectiveness of these tools can be seen in the results of Nvidia's own, end-to-end DLN for autonomous driving. This system, which uses a combination of DGX-1 and Drive PX 2, has demonstrated impressive results in real-world driving scenarios, showcasing the potential of DLNs in self-driving cars.[12]

Transportation safety

Traffic safety is a critical concern for transportation planners and policymakers, as road crashes continue to be a major cause of injury and death worldwide. Analytical Transportation Safety Planning (TSP) is an emerging approach that seeks to integrate safety considerations into the transportation planning process. Recent research has focused on improving the accuracy of crash prediction models, which are a crucial component of TSP. However, many of these models rely on aggregated data, which can lose important details and introduce biases. In a recent study, researchers proposed a deep learning approach to predict traffic crashes using high-resolution data. The approach was found to be more accurate than traditional models using low-resolution data, highlighting the potential of deep learning techniques for transportation safety planning. The study's findings suggest that using detailed data can provide valuable insights into the complex relationships between various factors that contribute to traffic crashes, ultimately leading to better policies and decision-making. As transportation data continues to grow in volume and complexity, the use of deep learning techniques is likely to become increasingly important for transportation safety planning.[13]

These are just a few examples of the many ways in which deep learning is being applied to transportation problems. As the field continues to evolve, we can expect to see even more innovative applications of deep learning in transportation in the years ahead.

2.3 SOFTWARES

2.3.1 YOLO

Object detection is a fundamental task in computer vision that involves locating and classifying objects within images or videos. The You Only Look Once (YOLO) algorithm is a deep learning-based approach [14] that has gained popularity in recent years due to its accuracy, speed, and ease of implementation. In this literature review, the key features, strengths, and weaknesses of YOLO, as well as its applications is explored.

The first version of YOLO was released in 2016. It used a single neural network to predict bounding boxes and class probabilities directly from full images. YOLOv1 achieved high accuracy on the PASCAL VOC dataset, but it was not as fast as other object detection Algorithms. In 2017, the authors of YOLO released YOLOv2, which improved upon the first version by using a more complex network architecture and a new loss function. YOLOv2 achieved state-of-the-art performance on the COCO dataset. Released in 2018, YOLOv3 includes several improvements, such as a larger network architecture, a new spatial pyramid pooling module, and a new loss function. More versions had been released since then. YOLOv8, which is the version that this study had adopted, is the latest and the most powerful version until the time of conducting the study.

YOLO is an open-source algorithm, which means that it is free and available for anyone to use. The source code is available on GitHub.

Key Features of YOLO

Real-time Object Detection: YOLO is designed for real-time object detection, which means that it can detect objects in images and videos in real-time. This is achieved through the use of a single neural network that predicts bounding boxes and class probabilities directly from full images.

End-to-End Learning: YOLO uses an end-to-end learning approach, which means that it learns to detect objects directly from the input data without requiring any pre-processing or feature engineering. This approach allows for faster training times and improved accuracy.

Convolutional Neural Networks (CNNs): YOLO uses CNNs to extract features from images and learn the patterns and relationships between them. CNNs are particularly effective for image recognition tasks because they can learn to detect increasingly complex features and patterns as the layers progress.

Anchors and Predictions: YOLO uses anchors to generate a set of possible bounding boxes for each object in an image, and then predicts the class probabilities and bounding box coordinates for each anchor. This approach allows YOLO to detect objects of various sizes and aspect ratios.

Loss Functions: YOLO uses a combination of two loss functions: the standard cross-entropy loss function and the smooth L1 loss function. The cross-entropy loss function is used to classify objects into different categories, while the smooth L1 loss function is used to refine the bounding box coordinates.

Strengths of YOLO

High Accuracy: YOLO has been shown to achieve high accuracy on several benchmark datasets, including the PASCAL VOC dataset, the COCO dataset, and the KITTI dataset.

Fast Speed: YOLO is designed for real-time object detection, which means that it can process images and videos quickly. This makes it suitable for applications that require real-time performance, such as autonomous driving, surveillance, and robotics.

Easy Implementation: YOLO has a simple architecture and requires minimal computational resources, which makes it easy to implement and deploy.

Flexibility: YOLO can be used for a variety of object detection tasks, including detecting objects in images, videos, and 3D point clouds.

Weaknesses of YOLO

Computational Cost: While YOLO is fast, it still requires significant computational resources, especially for high-resolution images and videos. This can make it difficult to deploy on low-power devices.

Training Time: Training YOLO models can be time-consuming, especially for large datasets. This can be a challenge for researchers and practitioners who need to fine-tune the model for specific applications.

Overfitting: YOLO uses a single neural network to predict bounding boxes and class probabilities, which can lead to overfitting, especially when dealing with large datasets.

Limited Interpretability: YOLO is a black-box model, which means that it can be difficult to interpret and understand the reasoning behind its predictions.

Applications of YOLO

Autonomous Driving: YOLO can be used for detecting objects in images and videos captured by cameras in autonomous vehicles, such as pedestrians, cars, and road signs.

Surveillance: YOLO can be used for detecting objects in surveillance videos, such as people, vehicles, and suspicious activity.

Robotics: YOLO can be used for object detection in robotics, such as detecting objects in a robot's environment and navigating around them.

Healthcare: YOLO can be used in healthcare for detecting objects in medical images, such as tumors, organs, and other anatomical structures.

Security: YOLO can be used in security applications, such as detecting objects in surveillance videos, recognizing faces, and identifying suspicious activity.

Retail: YOLO can be used in retail for object detection, such as detecting products on shelves, tracking inventory, and analysing customer behavior.

Overall, YOLO is a powerful and widely used object detection algorithm that has many applications in computer vision and related fields. Its real-time performance, accuracy, and ease of implementation make it a popular choice for researchers and practitioners.

2.3.2 StrongSORT

Recently, there has been growing interest in multi-object tracking (MOT), which involves detecting and tracking specific objects across frames in a video sequence. However, the existing methods for MOT have some limitations, such as relying on various basic models and using different training or inference techniques, which makes it challenging to compare their performance fairly. To address this issue, this study aims to provide a strong and fair baseline for MOT by revisiting a classic tracker, DeepSORT, and improving it from multiple perspectives.

The proposed tracker, named StrongSort [15], is based on DeepSORT but with several significant improvements, including the use of a strong detector and feature embedding model, and the application of advanced inference techniques. StrongSORT serves as a baseline for fair comparison between different tracking methods, particularly for tracking-by-detection methods. It also demonstrates the effectiveness of using a stronger baseline for demonstrating the performance of methods.

In addition, StrongSORT addresses two inherent "missing" problems in MOT: missing association and missing detection. Missing association refers to the situation where the same object is split into multiple tracklets, while missing detection refers to the failure to detect an object as a background. To address these problems, this study proposes two lightweight and plug-and-play algorithms, appearance-free link model (AFLink) and Gaussian-smoothed interpolation (GSI), which can be easily incorporated into various trackers with minimal computational cost.

AFLink uses spatiotemporal information to predict whether two input tracklets belong to the same ID, without relying on appearance models. GSI uses the Gaussian process regression algorithm to produce more accurate and stable localizations by considering motion information during interpolation. Both AFLink and GSI are model-independent, lightweight, and appearance-free, making them suitable for various trackers.

Extensive experiments demonstrate that StrongSORT, AFLink, and GSI achieve state-of-the-art performance on multiple public benchmarks, including MOT17 and MOT20. In particular, StrongSORT++ which combines StrongSORT with AFLink and GSI, achieves the best performance on these benchmarks. The results show that StrongSORT++ outperforms other state-of-the-art trackers, such as Center-Track, TransTrack, and FairMOT, with a running speed of 1.7 ms and 7.1 ms per image, respectively, on MOT17.

In summary, StrongSORT study provides a strong and fair baseline for MOT by revisiting and improving the classic tracker, DeepSORT, and proposing two lightweight and plug-and-play algorithms, AFLink and GSI, to address the missing association and detection problems. The proposed methods achieve state-of-the-art performance on multiple benchmarks and demonstrate the effectiveness of using a stronger baseline for demonstrating the performance of methods.

2.3.3 FFmpeg

FFmpeg is a free and open-source software project that contains a suite of libraries and programs for handling multimedia data. It includes libavcodec, an audio/video codec library used by many commercial and free software products, libavformat, an audio/video container mux and demux library, and ffmpeg, a command-line program to convert multimedia files between formats.

Since its inception in 2000, FFmpeg has become a fundamental tool in the multimedia industry, with a vast range of applications from video and audio conversion to live streaming. This review aims to explore the various facets of FFmpeg, its applications, and its impact on the multimedia industry.

Codecs and Formats

FFmpeg's strength lies in its wide-ranging support for multimedia formats. It can decode, encode, transcode, mux, demux, stream, filter, and play pretty much anything that humans and machines have created. A study has highlighted the utility of FFmpeg's libavcodec library, noting its comprehensive collection of codecs, which allows for compatibility with nearly all video and audio formats in use today.

Performance

The performance of FFmpeg's libraries and tools has been a subject of much research. A group has conducted a study to assess the performance of the FFmpeg's H.264 decoder. They found that FFmpeg provides efficient multithreaded implementations of popular codecs, contributing significantly to its performance. Another study also endorsed FFmpeg's performance, especially in the context of high-definition video decoding and streaming.

Applications

FFmpeg's application in research and industrial settings is widespread. For instance, in the field of computer vision, FFmpeg is frequently used to extract frames from videos for further analysis. It has also found usage in bioacoustic research, where it has been used for the automated detection and classification of animal vocalizations.

In the broadcasting industry, FFmpeg's streaming capabilities have been utilized for live streaming video content over the internet. Furthermore, many commercial software products, such as VLC, YouTube, and iTunes, rely on FFmpeg for multimedia data handling.

Limitations and Future Directions

Despite its advantages, some challenges and limitations of FFmpeg have been identified. One of these is the steep learning curve associated with its command-line interface, which can be a barrier for novices. Moreover, the lack of a standardized API documentation has also been pointed out as a potential difficulty for developers.

Future research could focus on enhancing FFmpeg's performance further, particularly in the context of emerging codec standards. Additionally, efforts could be made to improve the ease-of-use and accessibility of FFmpeg for beginners.

Conclusion

FFmpeg has proven to be a versatile and powerful tool in the realm of multimedia data handling. Its extensive format support, high performance, and wide range of applications underscore its value in both research and industry contexts. Going forward, continuous development and refinement of FFmpeg are likely to further cement its place as a cornerstone of the multimedia landscape.

2.3.4 Google Colab

Google Colab is a free, web-based platform for data science and machine learning that was first introduced in 2017. Since then, it has gained significant attention in the scientific community due to its ease of use, flexibility, and powerful capabilities. In this literature review, we will explore some of the key scientific research works that have utilized Google Colab, highlighting its applications, strengths, and limitations.

Applications of Google Colab

Data Science and Machine Learning: Google Colab is widely used in data science and machine learning tasks, such as data cleaning, preprocessing, visualization, feature engineering, model training, and evaluation. It provides a variety of tools and libraries, including TensorFlow, PyTorch, Scikit-learn, NumPy, Pandas, Matplotlib, and Seaborn, which are essential for data science and machine learning tasks.

Education: Google Colab is also widely used in education, particularly in teaching data science and machine learning courses. It provides a free, interactive, and collaborative environment for students to learn and practice data science and machine learning concepts. Instructors can create and share Colab notebooks with students, who can then work on them individually or in groups.

Research: Google Colab has been used in various research works, particularly in the fields of computer vision, natural language processing, and bioinformatics. It has been used for tasks such as image classification, object detection, speech recognition, text classification, and protein structure prediction.

Strengths of Google Colab

Ease of Use : Google Colab is very easy to use, even for users who are new to data science and machine learning. It provides a user-friendly interface, and users can start working on their projects without installing any software or setting up any environments.

Flexibility : Google Colab is highly flexible and can be used for a wide range of tasks, from data cleaning to model deployment. It supports a variety of programming languages, including Python, R, and Julia, and users can switch between languages seamlessly.

Collaboration : Google Colab supports collaboration, allowing users to share their work with others and work together in real-time. This feature is particularly useful for education and research, where collaboration is essential.

Scalability : Google Colab provides scalable computing resources, allowing users to run their jobs on large machines with powerful GPUs. This feature is particularly useful for training deep learning models, which require significant computational resources.

Limitations of Google Colab

Limited Resources: While Google Colab provides scalable computing resources, it still has limitations on the amount of resources that can be used. Users may encounter issues with memory, CPU, or GPU usage, particularly when working with large datasets or training deep learning models.

Security: Google Colab stores data and code in the cloud, which may raise concerns about data privacy and security. Users may need to take additional steps to ensure the security of their data, such as encrypting sensitive data or using virtual private networks (VPNs).

Dependence on Internet: Google Colab requires a stable internet connection, which may be a limitation for users who do not have reliable internet access. Users may need to plan their work accordingly and use alternative tools or environments when internet connectivity is an issue.

Conclusion

Google Colab is a powerful tool for data science and machine learning that has been widely adopted in various fields, including education, research, and industry. Its ease of use, flexibility, collaboration features, and scalability make it an attractive choice for many users. However, it also has limitations, such as limited resources, security concerns, and dependence on internet connectivity. Users should carefully evaluate these factors when deciding whether to use Google Colab for their work.

Chapter 3

Methodology

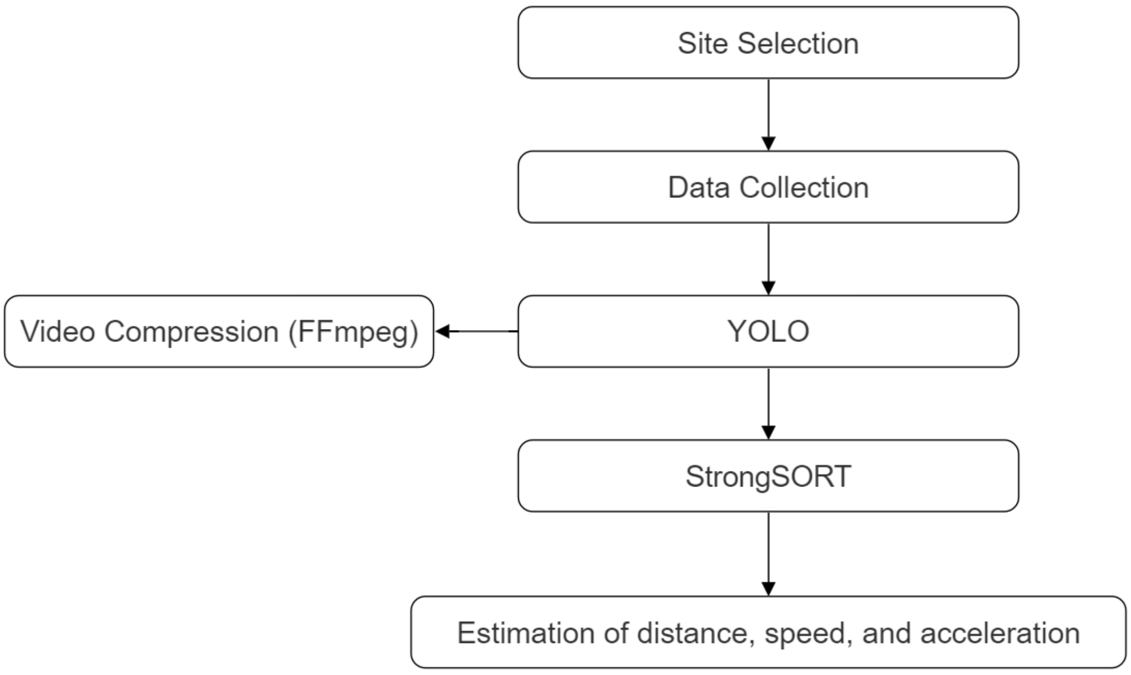

Site Selection
Data Collection
Video Compression (FFmpeg)
YOLO
StrongSORT
Estimation of distance, speed, and acceleration

Figure 3.1 Method Flowchart Illustration

3.1 SELECTION OF STUDY AREA

The road chosen for this study is “*Al-Mantiga Al-Sina’iya street*”, it’s located in Khartoum North, Kobar. It is a dual carriageway road, with two lanes in each direction. The road has a width of 10 meters, and a length of 55 meters (see fig 3.2 below).

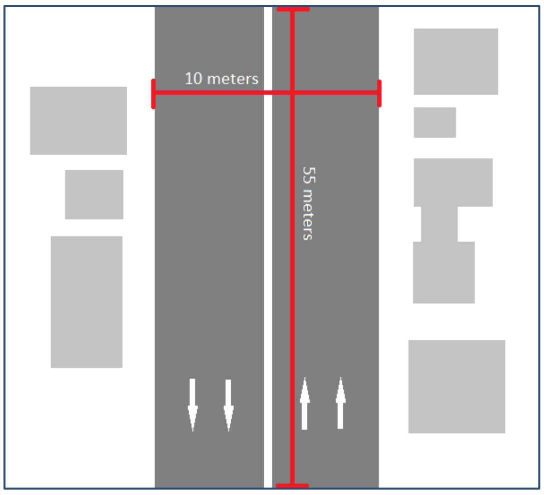


Figure 3.2 Illustration of the dimensions of the road used for this study

This road was characterized by a medium flow of traffic and featured a specific anomaly - a hole of moderate size that induced observable alterations in the trajectories of vehicles, at one carriageway.

3.2 ACQUISITION OF DATA

Data was acquired using a smartphone camera to record video footage. To avoid occlusion issues where vehicles might obscure each other, which will affect the tracking performance, the recording was made from an elevated vantage point, specifically a tall building adjacent to the road.

3.3 EXPERIMENTAL DESIGN

A video clip of 30 seconds duration was chosen for the experimental process. This relatively short duration was selected to reduce the time required for inference during the experimental phase.

For the purpose of executing training and inference experiments, Google Colab, a complimentary cloud-based platform, was employed.

3.4 MODEL SELECTION

For this study we utilized YOLO, a pre-trained machine learning model within the computer vision domain, for the task of vehicle detection in the video feed. To ascertain the X and Y coordinates of each detected vehicle in each frame, StrongSORT, a tracking software was employed.

The model employed for this study was the eighth iteration of the YOLO series, known as YOLOv8. This model is available in five distinct sizes, each designed to strike a balance between inference accuracy and speed. The available versions include YOLOv8n (Nano), YOLOv8s (Small), YOLOv8m (Medium), YOLOv8l (Large), and YOLOv8x (Extra Large).

Smaller versions of the model tend to produce quicker results but often sacrifice accuracy. Conversely, larger versions deliver high accuracy but require more processing time. For this study, the medium-sized YOLOv8m was chosen for the initial experimental phase. This size strikes a balance between acceptable inference speed and satisfactory results.

For the final output, the extra-large YOLOv8x was selected to maximize the accuracy of the results, even though it required a longer processing time.

3.5 ESTIMATION OF DISTANCE, SPEED, AND ACCELERATION

The detection and tracking of vehicles were performed in the original camera space.

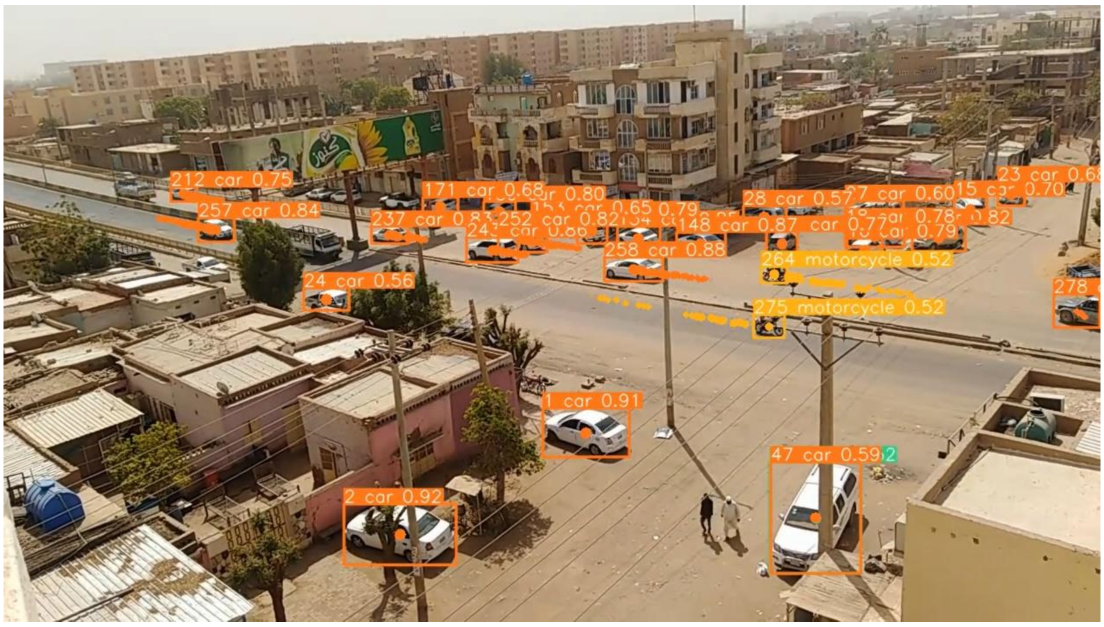


Figure 3.3 A shot from the inference video from the tracker.

To calculate the meters per pixel (MPP) ratio, a reference object was identified in the video. In this case, a car of known length (4.47 meters) was utilized. Due to perspective distortion resulting from the camera angle, which was not parallel to the roadway, it was necessary to account for changes in the MPP when calculating distances for speed calculation. This was achieved by selecting two points along the road and manually determining the pixel-length of the reference vehicle at these two points relative to their x-coordinates. The actual distance travelled by a vehicle anywhere along the roadway was calculated by first determining the MPP at the end point reached by the vehicle through linear interpolation between the vehicle's pixel-length at this endpoint, and the pixel-lengths at the two reference points, given the current x-coordinate of the vehicle. Multiplying this MPP ratio by the pixel-length travelled gives the actual distance in meters.

The Euclidean distance equation was employed to calculate the distance:

$$S_{point1,point2} = \sqrt{\Delta x^2 + \Delta y^2}$$

Where:

S ≡ Euclidean distance in pixels.

X and Y = Pixel-coordinates of two points (upper-left corners of bounding boxes of detected vehicles).

Vehicle speed was subsequently computed using the formula:

$$V_{i,k} = \frac{(S_{corner\ i,k\ ,\ \ corner\ i,k-x})\ \times\ MPP}{x/fps}$$

Where:

$V_{i,k}$ ≡ Estimated speed of vehicle i in frame k (m/s).

Corner i, k and corner i, k-x ≡ Corners of bounding box for tracked vehicle i in current frame and frame X frames prior.

Frame k and frame k-x = Current frame, and frame X frames prior where vehicle i was detected.

fps $\equiv$ Frame rate (30 fps for this study).

Finally, vehicle acceleration was derived using the equation:

$$a_{i,k} = \frac{V_{i,k}}{x/fps}$$

Where:

$a_{i,k} \equiv$ Estimated acceleration of vehicle i in frame k (m/s^2).

Vehicle trajectories were determined every X frames to mitigate noise in trajectory graphs that would arise if trajectories were calculated on a frame-by-frame basis.

The detailed python codes for all experiments are shown in Appendix A.

3.6 PROCESSING OF INFERENCE VIDEO

Beyond the trajectories derived from the study, an output video featuring uniquely tracked vehicles was generated. Each vehicle was assigned a unique track index, as displayed in Fig 3.3.

In an effort to conserve storage space and minimize data usage, the output video file was compressed using FFmpeg, an open-source suite of libraries and tools designed for handling multimedia data, including audio, video, and related file types. This software is capable of reducing the file size significantly, making it more manageable for storage and transfer purposes. Detailed information about FFmpeg and its functionalities is provided in section 2.3.3.

Chapter 4

Results

4.1 OVERVIEW

This chapter provides a detailed analysis and interpretation of the experimental outcomes described in the preceding chapter (chapter 3). The following sections delve into the results under two distinct scenarios: one where the lane is in perfect condition (termed as the "normal case") and another where the lane suffers from a deficiency (referred to as the "deficiency case").

Both cases - normal, and presence-of-deficiencies - are located at one road, with one carriageway being normal (without deficiencies), and the other is with a deficiency. The featured section's length is approximately 55 m. The type of the deficiency is a series-of-potholes type.

To aid in the understanding and interpretation of the results, tables and graphical representations are interspersed throughout the text. Also, a summary of findings is provided in section 4.4.

4.2 TRAJECTORIES ESTIMATION - NORMAL CASE

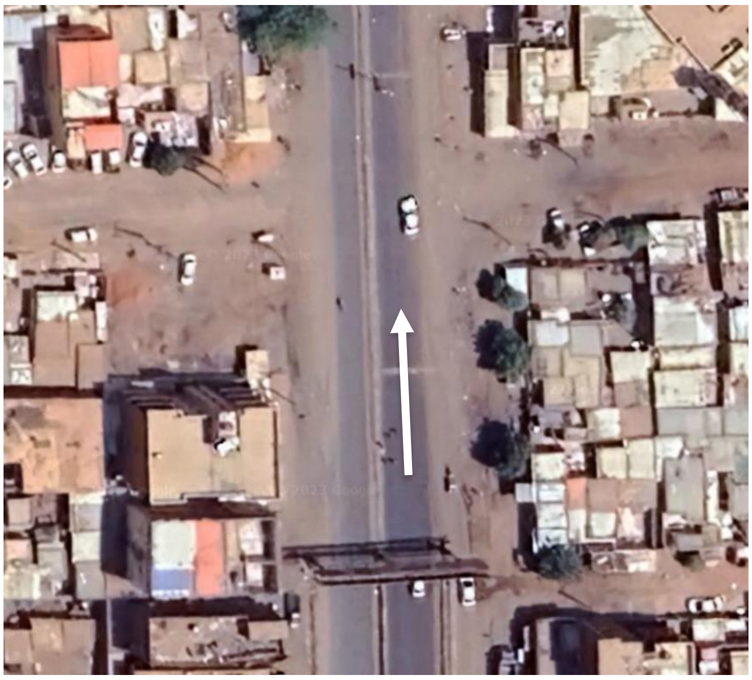
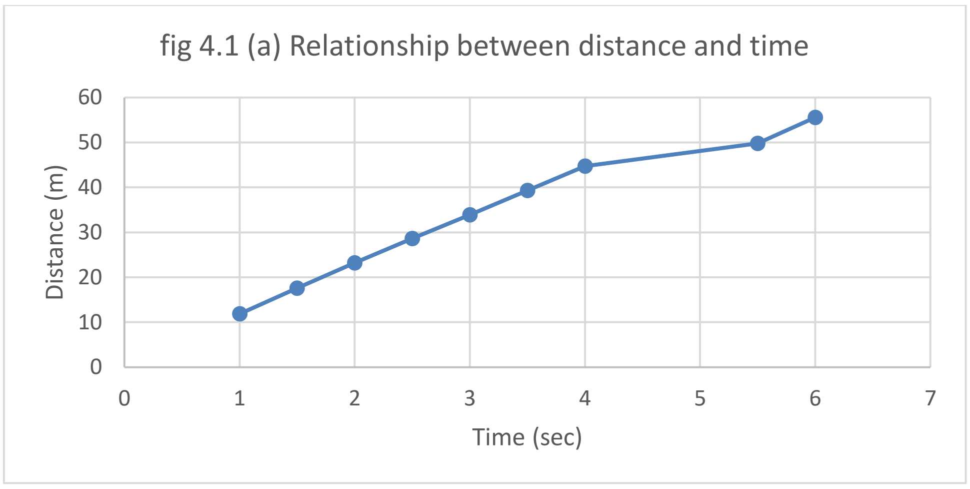


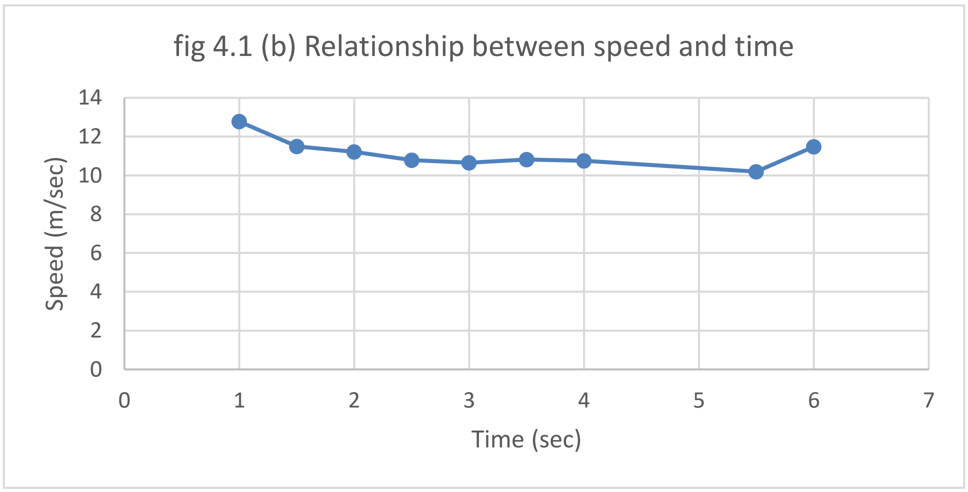
Figure 4.1 Trajectories Estimation - Normal Case.

This section focuses on the outcomes related to the estimated trajectories of a vehicle traveling on a lane without any deficiencies. As evidenced by table 4.1 and figure 4.2, the speed and acceleration parameters appear to be consistent across the entire stretch of the road. This implies that the vehicle neither significantly accelerated nor decelerated, as there was no need for the driver to do so. It's also noteworthy that the data was collected during off-peak hours, with free flow speed, which vehicles can travel on a roadway without encountering any significant hindrances or congestion. It represents the ideal or optimal speed at which vehicles can move safely and efficiently under normal traffic conditions.

Table 4.1 Estimated Trajectories of a Vehicle - Normal Case

time(sec)	distance(m)	speed(m/sec)	acceleration(m/sec^2)
1	11.85652653	12.77541105	-1.793282898
1.5	17.60861481	11.50417656	-2.542468978
2	23.21526841	11.2133072	-0.581738719
2.5	28.61011166	10.7896865	-0.847241405
3	33.9377656	10.65530787	-0.268757264
3.5	39.34769844	10.81986569	0.329115637
4	44.7241037	10.75281052	-0.134110331
5.5	49.82226435	10.19632129	-1.112978459
6	55.56594594	11.48736317	2.582083762





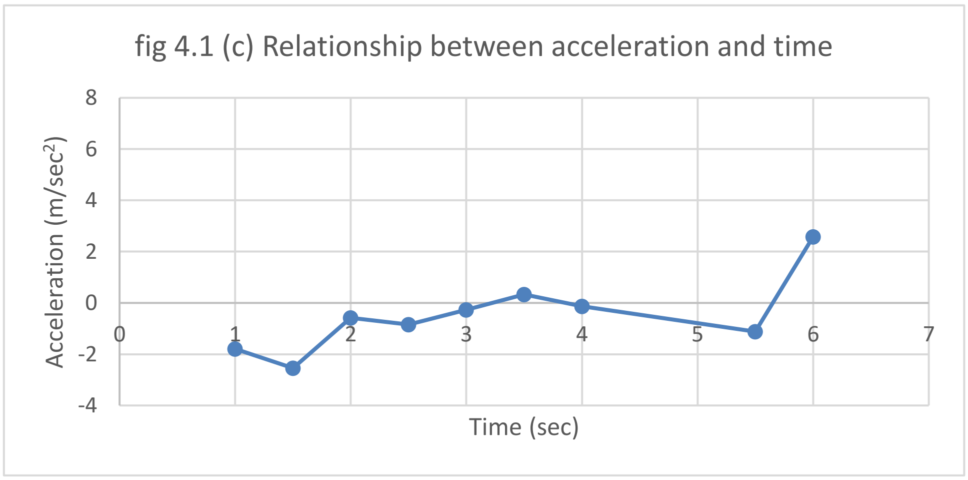


Figure 4.2 Estimated Trajectories of a Vehicle - Normal Case

4.3 TRAJECTORIES ESTIMATION - CASE WITH DEFICIENCIES

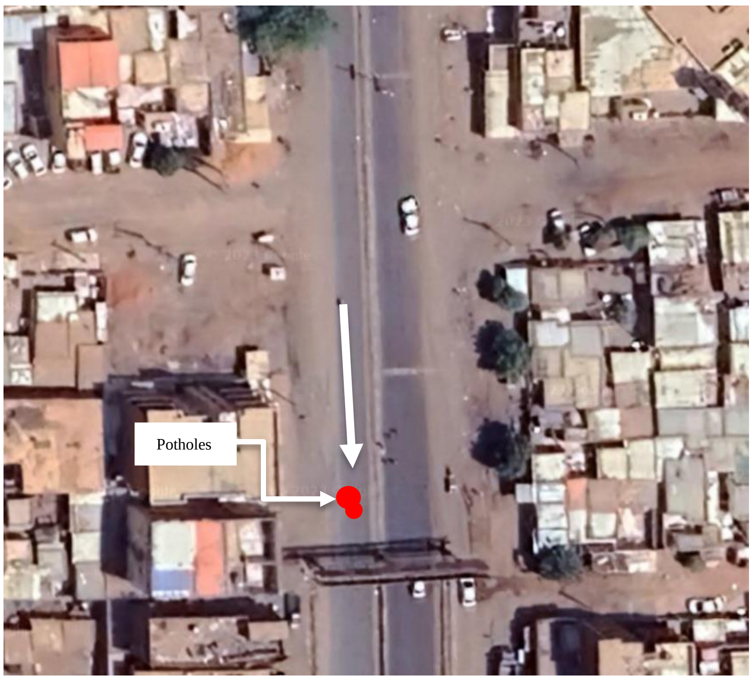


Figure 4.3 Trajectories Estimation - Case with Deficiencies

This section delves into the scenario where a vehicle traverses a lane exhibiting a deficiency. As anticipated, there was a discernible reduction in the speed of the vehicle as it approached the deficiency (refer to Fig 4.2). Thus, during off-peak hours, drivers tend to maintain a high speed and then progressively decelerate upon encountering a deficiency, resuming acceleration once they have navigated past the deficiency.

Table 4.2 Estimated Trajectories of a Vehicle - Case with a Deficiency

Time (sec)	Distance (m)	Speed(m/sec)	acceleration(m/sec^2)
18,5	2,964730645	5,743731425	5,915566887
19	5,74467874	5,559896189	-0,367670471
19,5	8,633451536	5,777545593	0,435298807
20	11,44906728	5,631231496	-0,292628193
20,5	14,14139503	5,38465549	-0,493152012
21	16,67879498	5,074799901	-0,619711178
21,5	19,0343668	4,711143646	-0,72731251
22	21,22312887	4,377524136	-0,66723902
22,5	22,76722664	3,088195546	-2,57865718
23	24,28284563	3,031237964	-0,113915164
23,5	25,70974918	2,853807105	-0,354861718
24	27,23250183	3,045505311	0,38339641
24,5	28,8335596	3,20211554	0,313220458
25	30,88955431	4,111989422	1,819747764
25,5	32,99661298	4,214117335	0,204255826
26	35,277295	4,561364033	0,694493396
26,5	37,44436548	4,334140967	-0,454446131
27	39,82812234	4,767513716	0,866745496
27,5	42,62776856	5,599292433	1,663557435

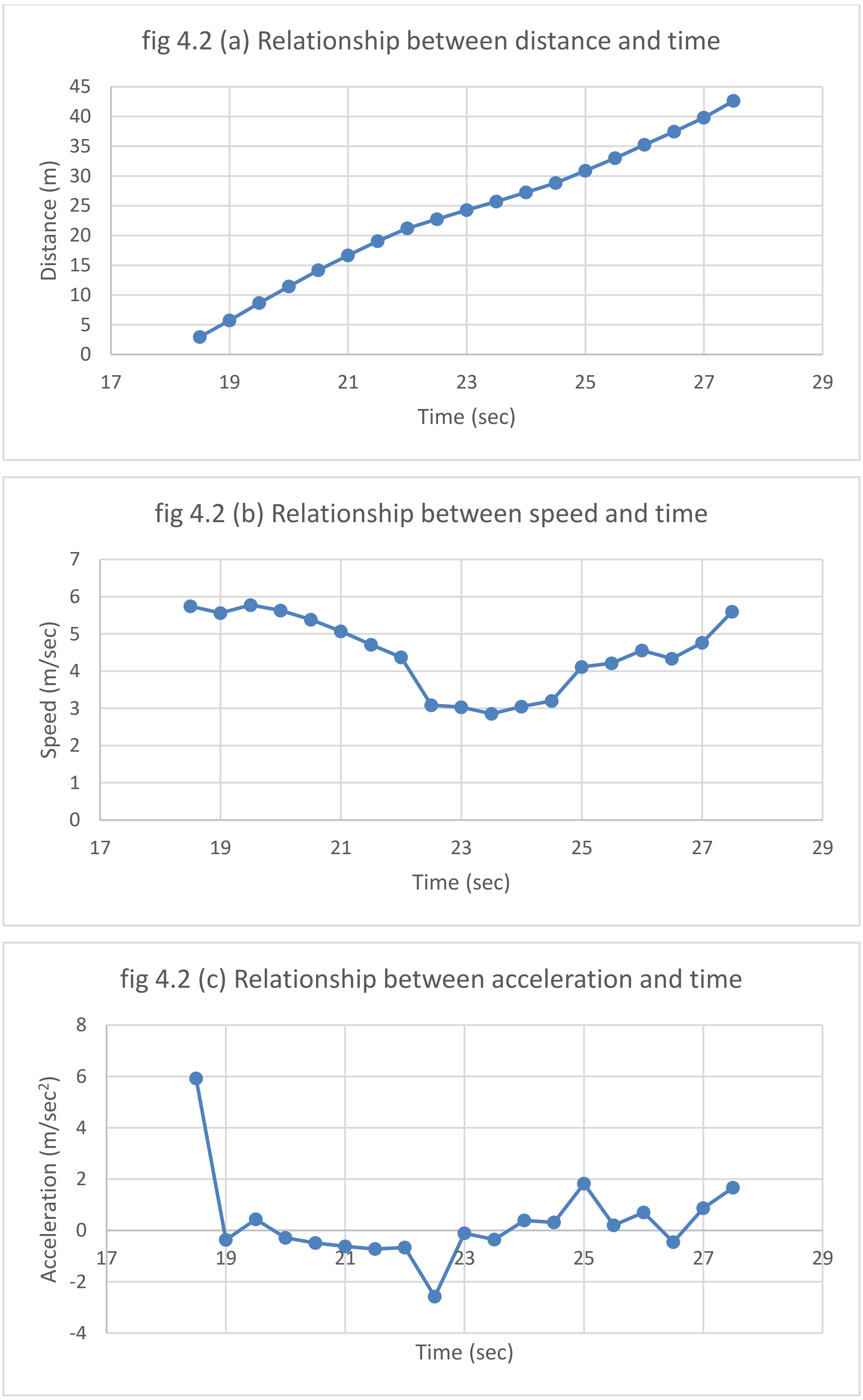


Figure 4.4 Estimated Trajectories of a Vehicle - Case with a Deficiency

Figure 4.5 below outlines the speed of multiple vehicles traversing a lane with a deficiency plotted against the distance. The plot clearly illustrates the considerable impact of the deficiency on the overall traffic flow on this road. The mean speed was also computed and incorporated into the plot (depicted as the bold line in the figure).

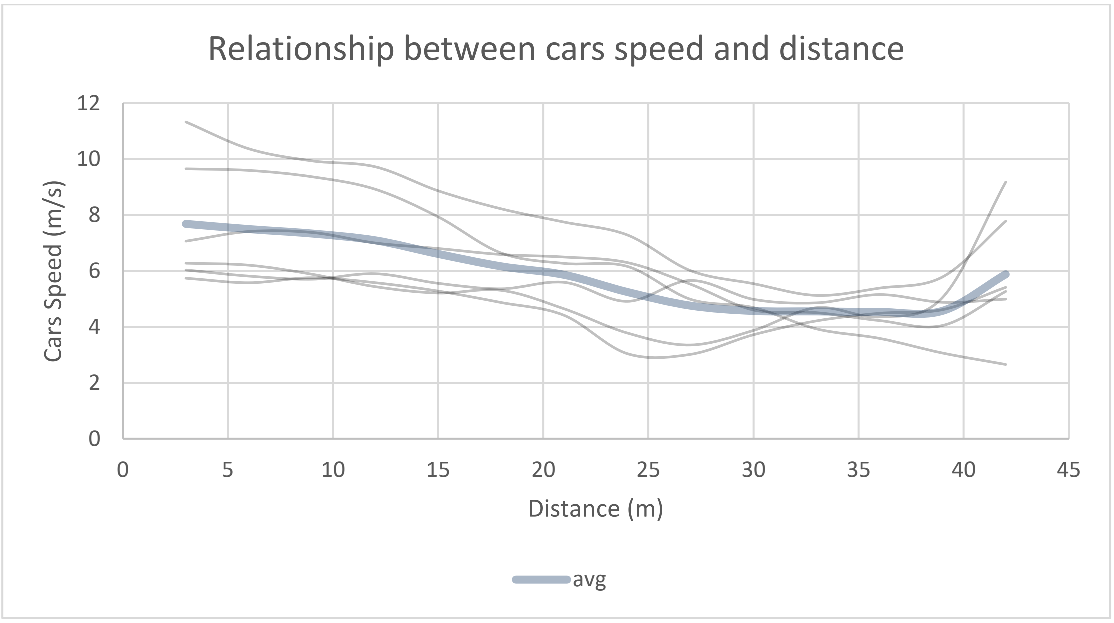


Figure 4.5 Speeds of Multiple Vehicles and The Mean Speed

4.4 SUMMARY OF FINDINGS

In this section, the experimental results from two different scenarios - the normal case and the case with present deficiencies - are summarized. By estimating and graphically portraying the trajectories of vehicles under each scenario, the profound influence of road deficiencies on traffic flow becomes evident. Such analysis could be instrumental for departments of Transportation and Infrastructure. They can leverage these insights to understand the extent of impact of specific deficiencies, and prioritize their remediation efforts in the context of a network of roads exhibiting multiple deficiencies.

Chapter 5

Conclusion and Recommendations

5.1 CONCLUSION

In this study, the complex problem of vehicle trajectory estimation from video data was explored under two distinct scenarios: (1) vehicles moving on roads without deficiencies, and (2) vehicles moving on roads with deficiencies. The differences in the estimated trajectories under these conditions were then compared and analyzed.

The practical implications of this research are substantial. By contributing to the body of knowledge on vehicle trajectory estimation, we can better understand and evaluate traffic efficiency and safety. Furthermore, this study stands out for its cost-effectiveness, as it uses readily available data or data that can be generated at a minimal cost. This approach makes it particularly appealing for jurisdictions with limited budgets for traffic and infrastructure repairs.

5.2 RECOMMENDATIONS

Based on the findings of this study, several recommendations are proposed to enhance the accuracy and applicability of vehicle trajectory estimation from video data in future research:

Incorporate Instrumented Vehicles: Using instrumented vehicles, equipped with onboard sensors to automatically measure vehicle trajectories, can significantly increase the accuracy of the testing and validation stages of the developed AI system. Instrumented vehicles can provide ground truth data against which the AI predictions can be compared.

Optimize Video Capture: The angle of video capture can significantly influence the accuracy of trajectory estimation. Capturing video from an angle more parallel to the road can reduce the impact of perspective distortion. Ideally, the use of drones for video capturing is recommended. Drones can eliminate issues associated with occlusion and perspective distortion. By increasing their altitude, drones can cover larger road lengths in the video, enabling the analysis of vehicle trajectories across greater distances and durations.

Invest in High-Quality Cameras: The quality of video data directly impacts the precision of the AI system. Therefore, investing in high-specification cameras can contribute to an increase in the overall accuracy of vehicle trajectory estimation.

Experiment with Various Trackers: There are various tracking algorithms available, each with its strengths and weaknesses. Future research should explore and compare the performance of different trackers to identify the most accurate and robust method for vehicle trajectory estimation from video data.

In summary, these recommendations aim to build upon the findings of this study and further advance the field of vehicle trajectory estimation. By implementing these suggestions, future researchers can improve the quality of their data and the accuracy of their results, leading to more reliable assessments of traffic efficiency and safety.

Bibliography

[1] Amr Khalifa, The Use of Computational Intelligence for Traffic Light Control.

[2] Canny, J. (1986). A computation approach to edge detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, 8(6), 679-698.

[3] R. E. Kalman, "A new approach to linear filtering and prediction problems," Journal of Basic Engineering, vol. 82, no. 1, pp. 35-45, 1960.

[4] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436-444.

[5] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems, 3061-3069.

[6] Ren, S., He, K., & Sun, J Ren, S., He, K., & Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. Advances in Neural Information Processing Systems, 9985-9993.

[7] Goodfellow, I., Pouget-Abadie, J., Mirza, M., & Xu, B. (2014). Generative adversarial nets. Advances in Neural Information Processing Systems, 2672-2680.

[8] Simonyan, V., Vedaldi, A., & Zisserman, A. (2014). Two-stream convolutional networks for spatial and temporal processing. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2791-2799.

[9] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2015). Rethinking the Inception Architecture. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2898-2906.

[10] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 770-778.

[11] Yin, X., Wu, G., Wei, J., Shen, Y., Qi, H., & Yin, B. (2021). Deep learning on traffic prediction: Methods, analysis, and future directions. *IEEE Transactions on Intelligent Transportation Systems*, *23*(6), 4927-4943.

[12] Kisačanin, B. (2017, May). Deep learning for autonomous vehicles. In *2017 IEEE 47th International Symposium on Multiple-Valued Logic (ISMVL)* (pp. 142-142). IEEE.

[13] Cai, Q., Abdel-Aty, M., Sun, Y., Lee, J., & Yuan, J. (2019). Applying a deep learning approach for transportation safety planning by using high-resolution transportation and land use data. *Transportation research part A: policy and practice*, *127*, 71-85.

[14] Redmon J., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.

[15] Du, Y., Zhao, Z., Song, Y., Zhao, Y., Su, F., Gong, T., & Meng, H. (2023). Strongsort: Make deepsort great again. *IEEE Transactions on Multimedia*.

Appendices

Appendix A

Code Listings

Backup Code

Code for connecting Google Colab with Google Drive to automatically save edits and results.

```
from google.colab import drive
drive.mount("/content/gdrive")

!ls

%cd gdrive/My Drive/Graduation_Project_YOLO_tracker_trajectories_extracter
```

Model Clonning

Code for Clonning the YOLOv8 with the tracker, repository from GitHub. Then, downloading the requirements needed for the model to run properly.

```
!git clone --recurse-submodules https://github.com/mikel-brostrom/yolov8_tracking.git
# clone recursively
%cd yolov8_tracking
!pip install -r requirements.txt  # install dependencies
```

Tracking

Code takes the captured traffic video as input, and generates the track file.
For each frame, the track file contains the x and y coordinates of all detected vehicles in this frame.

```
# The classes (and their indecies) we are going to track: car: 2, motorcycle: 3, bus:
5, truck: 7
# Using the (--save-txt) flag to save the trajectories
# Using the (--save-vid) flag to save the output video
# To get the size of the input video (So you can make sense of the output track.text
file) see: https://www.linuxshelltips.com/find-video-resolution-linux/
# To do mapping from pixels to meters, take a screenshot from the input video, resize
(to the same size of the video) it in krita, choose a reference object, then do the
mapping
# Always resize (using the --imgsz flag) to the size of the video (just my own
observation)
```

```
!python track.py --yolo-weights yolov8x.pt --tracking-method strongsort --source vlc-
record-2023-02-15-19h39m38s-VID_20230215_134159.mp4-.mp4 --classes 2 3 5 7 --imgsz
1280 --save-vid --save-txt
```

Compression of the Inference Video

Code for compressing the inference video.

```
# Compress the output video


!ffmpeg -i
/content/gdrive/MyDrive/Graduation_Project_YOLO_tracker_trajectories_extracter/yolov8_
tracking/runs/track/exp82/vlc-record-2023-02-15-19h39m38s-VID_20230215_134159.mp4-.mp4
-vf fps=30 -vcodec libx264
/content/gdrive/MyDrive/Graduation_Project_YOLO_tracker_trajectories_extracter/compres
sed_output_videos/comp_output82.mp4
```

Track File Reading

Code for the reading the track file as Pandas data frame.

```
import pandas as pd
import numpy as np
import math

df =
pd.read_csv('/content/gdrive/MyDrive/Graduation_Project_YOLO_tracker_trajectories_extr
acter/yolov8_tracking/runs/track/exp80/tracks/vlc-record-2023-02-15-19h39m38s-
VID_20230215_134159.mp4-.txt', header = None, sep = " ", index_col = False)
```

Editing of the Original Data Frame

Code for removing the unnecessary columns from the track file. Then, only leaving the x and y coordinates in the range of interest from the road.

```
df.drop(df.columns[[6, 7, 8, 9, 10]], axis=1, inplace=True) # Removing the unnecessary
columns

df.columns = ['frame', 'car', 'x', 'y', 'bbox_w', 'bbox_h']

# hole_cords = [620, 300]
involve_trajs_till_x_cord_ = 400
starting_x_cord = 1140 # the starting x coordinates, before which the bbox is trimmed

# To envolve the trajectories between the starting point and the hole:
df = df[df['x'] > involve_trajs_till_x_cord_]
df = df[df['x'] < starting_x_cord]
display(df)
```

Interpolation

Code for doing interpolation to estimate the pixel-length of the reference vehicle at any point across the road given its x coordinates.

```
def ref_px_len(x, carriageway):
  if carriageway == 'far':
    ref_px_len = ( (x - 555) * (115 - 40) / (1280 - 555) ) + 40
  else:
    ref_px_len = ( (x - 1280) * (85 - 142) / (830 - 1280) ) + 142

  return ref_px_len

ref_px_len(1003, 'close')
```

Trajectories Generation

Function for generating a vehicle's trajectories, taking as input: the vehicle's index, the cars' tracker dataset, the carriageway the vehicle is moving across, the desired time step, and the fps of the captured video.

```
def give_me_trajs(car_index, cars_dataset, carriageway, delta_t, fps):
  car = cars_dataset[cars_dataset['car'] == car_index]
  time_steps = []
  distance_from_last_cord = []
  speed = []
  acceleration = []
  x_cord = []
  last_cord = [car.iloc[0][2], car.iloc[0][3]]

  if carriageway == 'far':
    ref_real_len = 4.47 # the real length of the reference car of the 'far'
carriageway (The white tucson)
  else:
    ref_real_len = 4.84

  count = 0

  for i in car['frame']:
    if i % (delta_t * fps) == 0:
      time_steps.append(i / fps)
      present_cord = [car.iloc[count][2], car.iloc[count][3]]
      x_cord.append(present_cord[0])
      refpxlen = ref_px_len( present_cord[0], carriageway )
      px_to_m = ref_real_len / refpxlen
      if carriageway == 'far':
        b = 10
      else:
        b = 15 # not sure about this
      delta_dist = ( math.sqrt( (present_cord[0] - last_cord[0])**2 + (present_cord[1]
- last_cord[1])**2 ) - b) * px_to_m
      if len(distance_from_last_cord) == 0:
        if carriageway == 'far':
          a = 5
        else:
          a = 10 # not sure about this
        delta_dist = ( math.sqrt( (present_cord[0] - last_cord[0])**2 +
(present_cord[1] - last_cord[1])**2 ) - a) * px_to_m
        if delta_dist < 0: # don't use negative distances (negativity happens due to
subtracting a small number from a distance)
          delta_dist = ( math.sqrt( (present_cord[0] - last_cord[0])**2 +
(present_cord[1] - last_cord[1])**2 )) * px_to_m
          print(delta_dist)
        distance_from_last_cord.append(delta_dist)
        sp = delta_dist / ( (i - car.iloc[0][0]) / fps)
        speed.append(sp) # m/sec
        acceleration.append( np.nan )
      else:
        distance_from_last_cord.append(delta_dist + distance_from_last_cord[-1])
        sp = delta_dist / delta_t
        acceleration.append( ( (sp - speed[-1]) ) / delta_t ) # m/sec^2
        speed.append(sp) # m/sec
      last_cord = present_cord
    count += 1

  car_trajs = pd.DataFrame(
      {'time(sec)': time_steps,
       'x': x_cord,
       'distance(m)': distance_from_last_cord,
       'speed(m/sec)': speed,
       'acceleration(m/sec^2)': acceleration
      })

  return car_trajs
```

Vehicle trajectories to CSV

Code for converting the trajectories of a vehicle to a csv file, and determining the csv file directory in Google Drive.

```
# save the dataframe as a csv file - in the drive

car_trajs_54 = give_me_trajs(54, df, 'far', 0.5, 30)

car_trajs_54.to_csv('/content/gdrive/MyDrive/Graduation_Project_YOLO_tracker_trajector
ies_extracter/csv_trajectories_files/exp80_car54.csv')
```

Speed vs Distance

Function for determining a vehicle's speed at any point (distance) a cross the road.

```
def new_distances_to_new_speeds(car_trajs, lower_limit, upper_limit, interval):
    # this function assumes that the largest distance in new_distances is smaller than
the largest distance in old_distances
    # It also assumes that there's at most one new_distance less than the smallest
old_distance

    old_distances = car_trajs['distance(m)']
    old_speeds = car_trajs['speed(m/sec)']

    new_distances = list(range(lower_limit, upper_limit + 1, interval))
    new_speeds = []
    skip = 1

    if(lower_limit < old_distances[0]):
        sp = old_speeds[0] - ( (old_distances[0] - lower_limit) * (old_speeds[1] -
old_speeds[0]) / (old_distances[1] - old_distances[0]) )
        new_speeds.append(sp)

    else:
        skip = 0

    for i in range(len(new_distances)):
        if skip:
            skip = 0
            continue

        for j in range(len(old_distances)):
            if old_distances[j] < new_distances[i] and old_distances[j + 1] >
new_distances[i]:
                sp = ( (old_speeds[j + 1] - old_speeds[j]) * (new_distances[i] -
old_distances[j]) / (old_distances[j + 1] - old_distances[j]) ) + old_speeds[j]
                new_speeds.append(sp)

    return new_speeds
```

Lists concatenation

Function for concatenating two lists at the columns level.

```
def add_two_lists(list1, list2):
  sum_list = []
  for i in range(len(list1)):
    sum_list.append(list1[i] + list2[i])

  return sum_list
```

Many Vehicles' Speeds

Function for generating a list containing the speeds of a number of vehicles for the determined distance steps.

```
def give_me_many_speeds_with_chosen_distance(df_of_cars, list_of_wanted_cars,
lower_limit, upper_limit, interval, carriageway, delta_t, fps):
    output = pd.DataFrame()
    output['new_distances'] = list(range(lower_limit, upper_limit + 1, interval))
    sum_list = [0] * len(output['new_distances'])

    for car in list_of_wanted_cars:
        car_trajs = give_me_trajs(car, df_of_cars, carriageway, delta_t, fps)
        new_speeds = new_distances_to_new_speeds(car_trajs, lower_limit, upper_limit,
interval)
        output[str(car)] = new_speeds
        sum_list = add_two_lists(sum_list, new_speeds)

    n = len(list_of_wanted_cars)

    avg_list = [x / n for x in sum_list]

    output['avg'] = avg_list

    return output
```

Generating the speeds of many cars…

```
list_of_wanted_cars = [54, 98, 149, 157, 212, 237]

output = give_me_many_speeds_with_chosen_distance(df, list_of_wanted_cars, 0, 42, 3,
'far', 0.5, 30)
```

Saving the speeds of many cars as a csv file…

```
output.to_csv('/content/gdrive/MyDrive/Graduation_Project_YOLO_tracker_trajectories_ex
tracter/csv_trajectories_files/exp80_a_lot_of_cars_3.csv')
```