Steering towards robust velocity estimation: Regressing velocity of moving objects in real-world scenes using 3D long-range point tracking methods

Student: HMZD2
Supervisor: Prof. Gabriel J. Brostow

MSc Data Science and Machine Learning

September 2024

This report is submitted as part requirement for the MSc Degree in Data Science and Machine Learning at University College London. It is substantially the result of my own work except where explicitly indicated in the text.

Department of Computer Science

University College London

Abstract

Real-world velocity estimation of moving objects is an often overlooked, but challenging and crucial task in various domains such as autonomous driving, sports analytics, and wildlife conservation. The complexity of this problem arises from the need to accurately estimate the speed and direction of objects from visual data under varying and dynamic conditions. Traditional methods relied heavily on prior assumptions and optimization models, which are now being replaced by deep learning approaches. Despite significant progress, these newer methods often struggle with maintaining accuracy across diverse scenarios and lack robustness in challenging environments.

In this research, we developed a novel approach to real-world velocity estimation by integrating state-of-the-art 3D point tracking methods within a monocular imaging system. Specifically, we employed SpatialTracker, ZoeDepth combined with CoTracker, and DepthAnythingV2 combined with CoTracker, and tested these methods on a subset of the DriveTrack dataset. We evaluated our approach using benchmarks integrating these methods to test its applicability in various scenarios for vehicle velocity estimating, highlighting its strengths and limitations. Our findings show that SpatialTracker outperformed other methods in our benchmark integration for real-world velocity estimation. While our approach achieves promising results in standard conditions, its performance is challenged by high-speed objects and low-visibility scenarios, emphasizing the need for further refinement. This work contributes to the field by offering a new perspective on video-based velocity estimation, particularly in dynamic and real-world environments where the camera is moving as well.

Code for the project is available at the anonymised repository: `https://uclstudentano.github.io./`

Acknowledgements

I would like to express my deepest gratitude to: My supervisor, Professor Gabriel Brostow, for his passionate guidance and invaluable expertise. His support made this learning journey both enriching and unforgettable; Skanda Koppula, for his endless patience, support and helpful resources; Gizem Unlu and the Prism group, for their consistent guidance.

A special thanks to Alon and my family, whose support made this work possible.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction and Background

In video analysis, visual signals can be categorized into appearance and motion. Appearance refers to the spatial layout of a frame, whereas motion refers to the temporal dynamics of frames. In physics, motion is defined as the relative change in an object's position over time, regardless of the environment, or background in video context. Understanding motion is highly important for knowledge representation learning in video-based research. This facet of video analysis, called motion analysis, remains an active area of research.

Motion analysis used to be formulated as an optimization problem, relying on prior assumptions about the motion model representing a specific problem. Today, these traditional methods are increasingly being replaced by approaches that regress directly from observational data, with minimal or no prior assumptions about the models. Additionally, the field of motion analysis tends to focus on two main extremes: high-level tasks like action recognition and low-level tasks such as frame-to-frame optical flow. Between these extremes lies the intermediate task of physical estimation.

Physical estimation involves understanding and interpreting the physical properties and dynamics of objects within a scene. Examples of such problems are 3D reconstruction [2] and pose estimation [78]. These tasks are particularly challenging because it requires the model to comprehend the 3D structure of objects in the video, as well as their relationships to each other and the camera.

One often overlooked but equally challenging sub-task within physical estimation is velocity estimation.

1.1 Real-world velocity estimation

In computer vision, velocity estimation involves determining the speed and direction of moving objects within a sequence of images or video frames. Velocity estimation is crucial for tasks such as motion prediction, which aims to predict the future positions of objects after observing part of their motion, known as context motion.

Current prevalent methods for real-world velocity measurement integrate LiDAR (Light Detection and Ranging), a remote sensing technology that utilizes laser light to measure distances to an object, which is a common method used for autonomous driving [33]. LiDAR generates highly accurate and precise 3D information about the shape of objects and environments. Another popular sensor that shares

the same purpose is radar [42]: a sensor that with the help of radar transmitters, radio waves are sent and receive the reflected waves with the help of radar receivers. However, while LiDAR and radar sensors are generally accurate, these methods are expensive and can be unreliable in adverse weather conditions or when obstructed by other objects. An alternative cost-effective method for velocity estimation is the use of video cameras. Their affordability and versatility make them suitable for diverse applications such as the automotive industry [16], sports analytics [21], surveillance [24], and wildlife conservation [54]. While video cameras address some limitations associated with traditional sensors, they also present their own set of challenges. A key difficulty lies in relying solely on visual information to estimate the structure and motion of a moving object. This issue is particularly present when using a video made from a single camera, as the estimation becomes more complex when the object is observed from a moving camera.

Video-based velocity estimation can be approached in two ways: measuring the velocity of an object in image space and measuring the real-world velocity of an object. Image velocity estimation focuses on tracking the movement of pixels between frames to derive velocity, while real-world velocity estimation translates these pixel displacements into actual distances and speeds, incorporating the camera's properties and the scene's geometry. This conversion is essential for applications where accurate physical motion is critical, such as in autonomous vehicles.

There are various approaches for video-based real-world velocity estimation, which we will explore in detail in Section 2.1.1. Current dominating methods that are commonly used for this purpose are deep learning (DL) based methods, which can learn richer visual features and motion patterns from image sequences. DL-based approaches have significantly improved speed estimation performance [18], but typically excel only on their training datasets, struggling to maintain accuracy in different scenes or scenarios. This overlooked task of vision-based real-world velocity estimation requires more exploration to find methods that are more robust and reproducible in different scenes or scenarios.

1.2 Point-wise motion understanding

Instead of relying on traditional methods, which often involve specific shapes or forms, a broader approach to motion analysis views the world as composed of individual particles. Each particle moves along its own three-dimensional path through space and time. By tracking the movement of these particles, it is possible to analyze 3D motion without needing prior knowledge of any specific 3D shapes. This concept is referred to as point-wise motion understanding.

Previous research on 2D motion understanding has highlighted the significant value of this point-wise motion understanding. A notable example is the Track Any Point (TAP) task [13], which extends optical flow into a temporal framework aware of occlusions. Optical flow also has a 3D counterpart called scene flow. While scene flow is useful, it shares the same limitations as optical flow in a sense that it captures instantaneous motion but fails to understand pixels over long sequences.

The field has witnessed numerous new methods to achieve 3D and long-range motion understanding from videos. The recently introduced TAPVid-3D benchmark for evaluating the TAP task in 3D [32] shows the quality of current state-of-the-art methods by combining recent 2D point trackers and monoc-

ular depth estimators. As stated by its authors an example of a well recognized task within this area is monocular depth estimation.

1.3 Monocular imaging system

Point-wise motion understanding can be achieved through analyzing a sequence of images. To grasp the relationships between these images, we need to extract detailed information from each individual image consecutively. This necessity has led to the rising popularity of monocular vision systems in computer vision: a system that uses a single camera to capture and analyze visual information. This is in contrast to binocular or stereo vision systems, which use two cameras to obtain depth information.

Despite their advantages, monocular vision systems face significant challenges in applications requiring precise depth information. Accurately reconstructing 3D objects from a single image is inherently difficult due to the lack of depth cues, which are essential for spatial understanding. Additionally, identifying and distinguishing between overlapping objects in cluttered scenes is challenging without depth information, leading to potential inaccuracies in object recognition. Although monocular systems lack the inherent depth perception of binocular systems and face their own set of challenges, they present several advantages, such as cost-effectiveness, compactness, and reduced computational complexity.

1.4 Objectives and contributions

The goal of this research is to explore new methods for estimating real-world velocity estimation of moving objects by combining the benefits of a monocular system (Section 1.3) with the flexibility of point-wise tracking (Section 1.2) which can potentially lead us to an ideal solution: a system that can accurately regress velocity from monocular video input by measuring point-wise 3D trajectories over space and time, even when the camera itself is in motion. As a starting point for building such a system, we hypothesize that combining state-of-the-art tracking and depth estimation models, particularly advanced 3D long-range point tracking methods, are able to estimate velocity of moving objects in videos.

To achieve this, we will create benchmarks that assess velocity estimation in a point-wise 3D framework within a monocular imaging system. This involves combining state-of-the-art 2D tracking, monocular depth estimation, and 3D tracking methods.

The motivation behind developing these benchmarks is to find the most reliable approach for point-wise velocity tracking of objects in video footage which can be reproducible out of existing 3D long-range point tracking methods. It is important to note that our focus is not on the engineering aspects, such as optimizing inference time, but rather on evaluating whether this method is viable and suitable for the intended purpose. A detailed discussion of the objectives and research questions will be presented in Section 3.1, following the background information to ensure a comprehensive understanding.

1.5 Organisation

Chapter 2 provides an overview of the relevant literate in velocity estimation examining their application in vehicle velocity estimation to date, camera pose estimation, 2D and 3D tracking, monocular depth estimation and the combination of these areas. The methods used in the present research are detailed

fully in Chapter 3, and Chapter 4 subsequently reports our experiments, evaluations and results. We discuss the conclusions, strengths and limitations of the research alongside future directions in Chapter 5.

Chapter 2

Related work

This chapter provides a comprehensive review of the relevant literature and background information. We begin by exploring recent methods for vehicle velocity estimation (Section 2.1. This is followed by a discussion of the essential background information required for our study. To meet our objectives, it is crucial to first understand the principles of scene reconstruction. We cover both static scene reconstruction and the prevalent methods used in this area (Section 2.2). Next, we discuss dynamic scene understanding, focusing on tasks that are closely related to our work, particularly 3D tracking, while highlighting the leading methods in this field (Section 2.3). Finally, we address monocular per-pixel depth estimation, a key component of our study. We explore recent advancements in this area, assessing their applicability to the challenges posed by both real-world and synthetic data in depth estimation, with a focus on the most prominent methods (Section 2.4).

2.1 Current state of velocity estimation

While the task of velocity estimation may often be overlooked, significant progress has been made in the field. This section will review recent literature and methodologies to provide a comprehensive understanding of the current state of velocity estimation.

2.1.1 Existing methods

One of the earliest significant contributions for the task of real-world velocity estimation came from Kampelmühler et al. [29] in 2018. Their research focused on predicting the relative velocity of other vehicles using only monocular camera footage, even when the ego-vehicle was in motion. Their approach involved translating RGB video into a set of features, which were then used to train a multilayer perceptron (MLP). These features, including tracks, depth, and motion, were all derived from video sequences. The researchers used the TuSimple dataset, comprising 1074 driving videos of freeway traffic captured by a single HD camera. Their light-weight approach was able to estimate real-world velocity well and has won the CVPR2017 vehicle velocity estimation challenge.

Building on Kampelmühler's work, McCraith et al. [41] achieved comparable results with a simplified two-step approach. They utilized only a tracker and bounding box annotations to train an MLP, employing pre-trained networks for monocular depth estimation (MonoDepth [22]) and optical flow estimation (FlowNet2 [27]), rather than deriving this information from the video sequences such as Kam-

pelmühler et al. The outputs from these streams were passed through the multilayer perceptron which simplified their approach. An interesting aspect of McCraith's research was the generation of synthetic training data using bounding boxes. Interestingly, training solely on this synthetic dataset yielded excellent velocity estimations when evaluated on real data from the TuSimple dataset. Jain et al. [28] tackled the same problem using a similar methodology: Their approach involved feature extraction from data using optical flow calculation and employing a tracker to input data into an MLP.

Interestingly, these early approaches share a common trait: they use pre-trained models solely for feature extraction and then train their own MLPs afterwards.

Hayakawa and Dariush [26] introduced a method that does not involve training a new network and, therefore, diverges from the common thread of prior approaches. Instead, they integrate three pre-trained neural networks for feature extraction, using simple derivations to calculate the relative velocity of surrounding cars and ego-motion. To improve the accuracy of 3D position and orientation estimation for a vehicle, this method combines monocular depth estimation with the RANSAC algorithm [19] to dynamically estimate and correct the ground plane. By relying solely on pre-trained models, their framework avoids the need for additional training, increasing its robustness compared to the previously described methods. Even though the building blocks in their method can easily be substituted by novel algorithms. Their ground plane estimation method still makes the whole framework tailored to the specific environment of vehicles on the road.

A more generalizable framework has been recently introduced by Liebe et al. [36] in 2023. They argue that while research in the field mainly compete on accuracy levels, most solutions lack robustness and reproducibility across different datasets. Their speed estimation pipeline consists out of five consecutive steps: vehicle tracking, frames per second (FPS) estimation, depth estimaton, scaling factor estimation and speed estimation. This pipeline outputs the average speed in km/h. While FarSec may not outperform existing models in terms of accuracy, it contributes an end-to-end framework demonstrating compatibility with real-world datasets due to its modular design. This approach closely aligns with our research goals of building robust pipelines. However, while FarSec is limited to static cameras and vehicle velocity estimation, our approach advances robustness further by taking into account moving cameras as well.

These recent velocity estimation methods can be categorized based on several factors. First, we can distinguish between methods that train their own network and those that use pre-trained models. Second, we can differentiate between approaches that account for a moving camera and those that assume a static viewpoint. Lastly, we can categorize methods based on whether they operate in real-time or prioritize accuracy over speed.

Although significant progress has been made in vehicle velocity estimation, the research in this area remains limited, with considerable opportunities for further improvement and exploration. Developing flexible and reproducible methods, such as those by Liebe et al. [36], and robust approaches that account for challenges like moving cameras, as demonstrated by Kampelmühler et al. [29], are essential steps toward creating a more resilient system. Such a system could potentially be adapted for use in environ-

ments beyond vehicle applications, broadening its applicability. It is noteworthy that, with the exception of the method by Hayakawa and Dariush [26], all other described methods are capable of real-time operation. Real-time performance is crucial for a functional vehicle velocity estimation system, particularly in the context of autonomous vehicles. However, our research prioritizes evaluating the feasibility of our proposed velocity estimation approach over achieving real-time performance. As a result, we will not focus on optimizing for real-time operation at this stage, leaving that aspect for future engineering refinements once our approach has been thoroughly tested.

2.1.2 Why is vehicle velocity estimation so prevalent?

A review of the literature [18][17] reveals a significant concentration of velocity estimation research in automotive contexts. This raises the question why velocity estimation in automotive environments attracts such intense focus and what specific demands drive this concentration in the field.

As clearly explained by Llorca et al. [18], vehicle speed estimation serves multiple crucial functions. It enables traffic monitoring, forecasting, and control, which are essential for urban planning and real-time traffic management. These capabilities facilitate adaptive traffic signal control, navigation systems that respond to current conditions, and the detection of traffic jams or accidents. These different application domains are depicted in Figure 2.1.

Figure 2.1: Visual examples of speed detection scenarios across various contexts. The left panel showcases traffic monitoring and aerial drone imagery. The center panel illustrates speed enforcement camera views. The right panel depicts the perspective from autonomous vehicle systems. This example figure is taken from Llorca et al. [18]

Furthermore, speed detection plays an important role in road safety. High car speed is a large cause of both the occurrence and severity of traffic accidents[70]. Effective speed enforcement, often achieved through technologies like speed cameras, has been shown to significantly reduce speeding violations and crash rates [18].

Our task discussed in this paper, monocular velocity estimation, finds its most significant application in autonomous vehicles, especially when cameras are mounted on moving cars. In the context of autonomous vehicles, accurate estimation of surrounding vehicles' speeds and distances is fundamental.

This information is crucial for various driver assistance systems, from basic adaptive cruise control to advanced autopilot features that can handle complex maneuvers like overtaking, and also prevent collisions. Autonomous vehicles are not widely implemented yet, but they can become a big factor in saving time and space reducing road accidents and road congestion. In addition to saving time and space, and as explained by [48] they also facilitate the mobility of the elderly, and disabled.

2.2 Static scene reconstruction

Static scene reconstruction is a computer vision technique that creates 3D digital models of motionless environments. It works by analyzing multiple two-dimensional images or depth measurements taken from various viewpoints around a scene. These images are processed to identify common points, which are then used to calculate the 3D structure of the environment. This process effectively transforms flat, 2D information into a rich, 3D representation of the space. The static aspect is very important as the scene must remain unchanged during capture to ensure accuracy. Several popular methodologies for static scene reconstruction include Structure-from-Motion (SfM) [65] and Multi-View Stereo (MVS). SfM, a photogrammetric range imaging technique, estimates 3D structures from 2D image sequences through feature matching, camera pose estimation, and 3D point triangulation. Complementing SfM, MVS techniques employ multiple images with known camera poses to reconstruct 3D scene geometry via dense correspondence matching and depth map computation. Traditional multi-view approaches for depth estimation are increasingly being overshadowed by advancements in monocular depth estimation. These improvements allow us to infer depth information from the features of a single image, complementing and substituting traditional multi-view methods. We will explore this topic in more detail in Section 2.4.

2.2.1 Camera pose estimation

Camera pose estimation, a fundamental task in computer vision, becomes essential when the camera positions and orientations for static scene reconstruction described in the previous section are unknown. This process involves determining a camera's position and orientation in three-dimensional space relative to a known coordinate system or reference frame. Camera pose estimation techniques can be broadly categorized into two main approaches: batch-mode (offline) and online (real-time) estimation. Batch-mode methods analyze all frames simultaneously, offering higher accuracy through global optimization but at the cost of increased computational intensity, making them unsuitable for real-time applications. COLMAP [57] is a leading example of offline estimation and is considered one of the standard approaches for camera pose estimation, alongside Bundler [59] and VisualSfM [10] [71]. In contrast, online estimation methods process frames sequentially as they arrive, providing lower latency and real-time performance. While these methods may sacrifice some accuracy for speed, they excel in handling dynamic scenes and moving cameras. Visual SLAM systems such as ORB-SLAM [43] and LSD-SLAM [15] exemplify real-time estimation approaches, making them examples of online estimation approaches. The choice between these methods depends on the specific requirements of the application, balancing the need for accuracy against the demands for real-time performance.

As previously mentioned, our research prioritizes evaluating the feasibility of our proposed velocity estimation approach over achieving real-time performance. Consequently, our focus will be on obtaining more accurate results rather than optimizing for speed. Therefore, we have decided to concentrate on a batch-mode method. Given the exceptional performance of COLMAP, we will explore this model in greater detail to provide context and explain why it stands out.

2.2.2 COLMAP

COLMAP is a comprehensive Structure-from-Motion (SfM) and Multi-View Stereo (MVS) pipeline designed to create 3D reconstructions from collections of images. It aims to provide a general-purpose solution that improves upon the current state-of-the-art in 3D reconstruction. The COLMAP pipeline begins with feature extraction, which identifies distinctive points in each image that can be reliably matched across multiple images. COLMAP uses the SIFT (Scale-Invariant Feature Transform) algorithm [38] by default, but it also supports other feature detection methods. Following feature extraction, COLMAP performs feature matching to identify correspondences between images, aiming to find the same physical points across different images. COLMAP offers several matching strategies, including exhaustive matching, which compares every image against every other image; sequential matching, which compares each image with its neighbors in a sequence; vocabulary tree matching, which uses a hierarchical structure to speed up matching in large datasets; and spatial matching, which utilizes known GPS locations of images to guide the matching process. Once potential matches are found, COLMAP performs geometric verification to filter out incorrect matches, using epipolar geometry constraints to ensure that the matched features are consistent with a valid 3D scene structure. The core of COLMAP's functionality is its Structure-from-Motion algorithm, which simultaneously estimates camera poses (position and orientation) for each image, 3D positions of the matched features (creating a sparse point cloud), and camera intrinsic parameters (if not provided). COLMAP uses an incremental SfM approach, starting with a carefully selected subset of images and gradually adding more images to the reconstruction. After the sparse reconstruction from SfM, COLMAP can perform dense reconstruction using Multi-View Stereo techniques, creating a denser point cloud or a mesh representation of the scene. The MVS step includes depth map computation for each image, depth map fusion to create a dense point cloud, and optional meshing of the point cloud. Throughout the reconstruction process, COLMAP performs bundle adjustment to refine the estimated camera parameters and 3D point positions. This optimization step minimizes the reprojection error, improving the overall accuracy of the reconstruction. Due to COLMAP's performance and user-friendly interface, we have selected it as the foundational tool for camera pose estimation in our study.

2.3 From 2D to 3D Tracking

We have discussed the reconstruction of static scenes, but how do we approach dynamic scenes, as required in our study? The field of motion analysis contains a variety of methods for dynamic scenes. We will now narrow down to the methods similar to our approach and explain why and how our approach relates to them.

2.3.1 Similar tasks of sparse 3D tracking

Optical flow is a technique in computer vision used to analyze motion in video sequences. The concept was first introduced by Gibson in the early days of the field [1]. It estimates the apparent motion of pixels between consecutive frames in a 2D image sequence, resulting in a 2D vector field that represents how each pixel moves from one frame to the next in the image plane. Optical flow typically works with RGB images.

Scene flow [61] extends the concept of optical flow to 3D space, estimating the actual 3D motion of points in a scene. This results in a 3D vector field that describes how each point in 3D space moves between frames, incorporating depth information for a more complete representation of motion. While optical flow provides 2D motion information, scene flow offers a richer understanding of motion in a scene by accounting for depth. This allows scene flow to better handle occlusions and depth discontinuities. However, scene flow requires additional depth information, leading to greater computational complexity and the need for more detailed input data compared to optical flow. Despite its advantages in providing detailed 3D motion information, scene flow is not suitable for long-range motion tracking. Both optical flow and scene flow are considered dense methods, as they attempt to estimate motion for every pixel or point in the scene. This distinguishes them from sparse tracking methods, which focus on specific features or points of interest and are generally less computationally expensive. Sparse methods are particularly beneficial when the objective, as in our study, is to track a specific object within the environment, as they require fewer resources and can be more efficient. Koppula et al. [32] explain that their prior research has demonstrated that having sparser point coverage is adequate for tracking applications such as in our study.

Another task similar to our approach is dynamic scene reconstruction. This task aims to create 3D models of scenes that change over time. This is more challenging than static reconstruction because it needs to account for moving objects and changing geometry, meaning the alterations in the shapes, positions, and spatial relationships of objects within a scene over time. An exciting relatively new method here is 3D Gaussian splatting. This approach represents a scene using a collection of 3D Gaussian primitives, each with properties like position, size, and color. These Gaussians are dynamically updated and optimized to adapt to scene changes, allowing for efficient and high-quality reconstruction of dynamic environments. Koppula et al. [32] explains that traditional dynamic scene reconstruction methods typically require expensive and time-consuming optimization processes for each video. These methods often also rely on assumptions about object motion within the scene. To address these limitations, Koppula et al. introduce a new dataset and benchmark called Tracking Any Point in 3D (TAPVid-3D) [32], which aims to advance the field by enabling sequential processing of data without the need for iterative optimization. This approach removes the necessity for video-specific optimization and prior motion information, as emphasized by the authors. Following the solution of velocity estimation by measuring point-wise 3D trajectories over space and time, introduced in Chapter 1, is strongly related to this approach. This dataset consists out of 4000+ real-world videos combining three existing datasets: PointOdyssey [79], DriveTrack [3], and Aria Digital Twin [47]. Motion analysis in videos can be ap-

proached with a variety of methods, particularly when it comes to velocity estimation. The authors of TAPVid-3D provide a comprehensive comparison of the methods explained above and situating their approach within this broader context. It is important to note that while methods such as scene flow, optical flow, and dynamic scene reconstruction provide useful information for motion analysis, they have already been thoroughly investigated in the context of TAPVid-3D. Consequently, our research focuses on leveraging the TAPVid-3D benchmark, as it specifically addresses the challenges of long-range point tracking in 3D space. This decision allows us to build upon the existing comparative analysis and concentrate on advancing the state-of-the-art in 3D trajectory estimation for extended temporal sequences. To achieve this, we can approach 3D long-range trajectory estimation as either a factorized estimation problem or a joint estimation problem. We define a factorized estimation problem as appending a 2D estimated trajectory by a 2D tracker with an estimated depth by a monocular depth estimation model. We define a joint estimation problem as an estimation where we use a model that does these two tasks jointly and therefore uses depth as leverage for the 2D trajectories.

2.3.2 SpatialTracker

An example of a state-of-the-art joint estimation method, and therefore direct 3D trajectory estimation model is SpatialTracker [69], introduced at CVPR 2024. SpatialTracker tracks 2D pixels in 3D space. 3D tracking comes with complexities, especially in scenes under occlusion or estimated complex 3D motions. This difficulty often comes from tracking in only 2D image space and therefore disregarding the 3D nature of motion. SpatialTracker's method seems to work well with these edge cases. They do this by leveraging geometric prior from monocular depth estimators and using that to lift 2D pixels into 3D and tracking in 3D space. This way they omit the issues caused by image projection and even enforces 3D motion priors such as the As-Rigid-As-Possible (ARAP) constraint. In this way, the model learns which points move rigidly together. The authors have proposed to represent each frame's 3D scene with triplane feature maps [9]. This representation enables to cover the 3D space densely making it possible to track any 3D point. Their model achieves state-of-the-art performance on benchmarks including TAP-vid [12] and PointOdyssey [79]. The aim of their model is to track any given query pixel throughout an entire monocular video by leveraging 3D space, allowing it to access and utilize more 3D contextual information for accurate tracking.

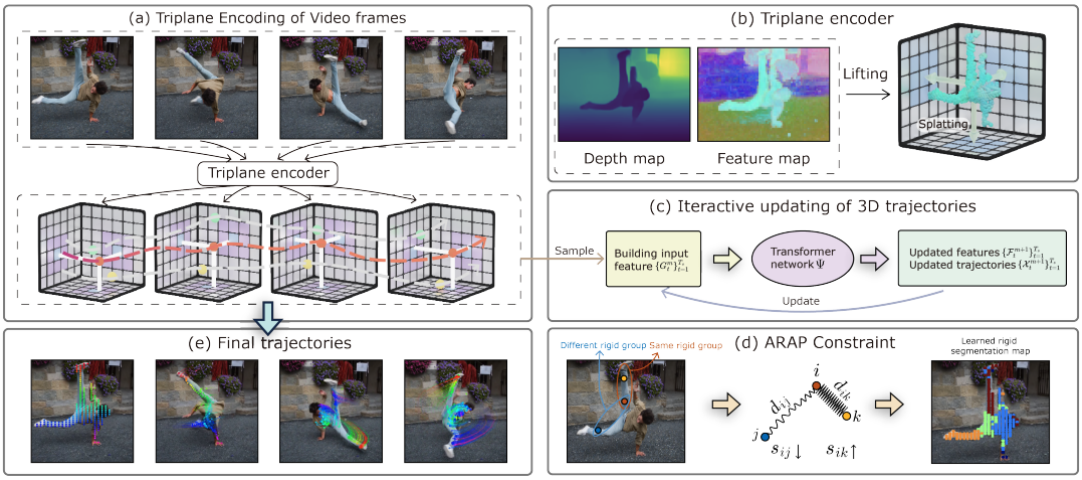


Figure 2.2: High-level overview of SpatialTracker's pipeline. Taken from Xiaou et al. [69]

Figure 2.2 illustrates the complete pipeline of SpatialTracker. The process begins with a monocular depth estimator, which converts 2D pixel coordinates into 3D coordinates. These coordinates are then projected onto triplane feature maps for easier processing. The raw images are transformed into feature representations, which are used by transformers to iteratively predict the trajectory. To maintain rigidity, clusters of pixels that move together are identified, enforcing the As-Rigid-As-Possible (ARAP) constraint. The total loss [69] has used in training their model consists out of the 3D trajectory loss shown in 2.1, the visibility loss shown in 2.2 and the ARAP loss shown in 2.3. The 3D trajectory loss is supervised by the ground truth 3D trajectories and represented in the paper as following:

$$\mathcal{L}_{\text{traj}} = \sum_{m=1}^{M} \sum_{i=1}^{N} \sum_{t=1}^{T_s} w^m \|\mathbf{X}_i, t^m - \hat{\mathbf{X}}_i, t^m\|_1 \tag{2.1}$$

where $\mathbf{X}_i, t^m$ and $\hat{\mathbf{X}}_i, t^m$ represent the predicted and ground-truth 3D positions, respectively. The term w^m corresponds to the weight associated with the m-th step. This is followed with the visibility loss proposed as:

$$\mathcal{L}_{\text{vis}} = \sum_{i=1}^{N} \sum_{t=1}^{T_s} \text{CE}(v_{i,t}, \hat{v}_{i,t}), \tag{2.2}$$

where $v_{i,t}$ and $\hat{v}_{i,t}$ represent the predicted and ground-truth visibility, respectively. The function CE denotes the cross-entropy loss.

The ARAP loss encourages distances between pairs of points that show high rigidity over time. This is proposed in their paper as:

$$\mathcal{L}_{\text{arap}} = \sum_{m=1}^{M} \sum_{t=1}^{T_s} \sum_{\Omega_{ij}} w^m s_{ij}^m \|d(\mathbf{X}_i, t^m, \mathbf{X}_j, t^m) - d(\mathbf{X}_i, 1, \mathbf{X}_j, 1)\|_1, \tag{2.3}$$

where Ω_{ij} is the set of all pairwise indices, and $d(\cdot, \cdot)$ is the Euclidean distance function. The weight w^m applies to the m-th step. Adding these losses together gives the total loss:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{traj}} + \alpha \mathcal{L}_{\text{vis}} + \beta \mathcal{L}_{\text{arap}}, \tag{2.4}$$

where α and β are weighting coefficients.

Due to the state-of-the-art performance of this model and its unique approach to velocity estimation

integrating it as a joint method rather than a factorized one like CoTracker, we have selected SpatialTracker as a baseline for further research.

2.3.3 CoTracker

Further then a joint estimation method such as SpatialTracker, a factorized estimation method can be explored as well. For this type of method, we need a 2D tracker. A current popular method that surpasses most existing methods in performance across various benchmarks in the field is CoTracker [30]. CoTracker is a transformer-based model that accounts for the relationship and correlation between points, thus combining ideas from optical flow and tracking. Therefore CoTracker tracks dense points jointly in a frame across a sequence of videos. The goal of CoTracker is to track 2D points over the length of a video sequence. This process involves generating a trajectory for each of the N points within sequential RGB frames. Each point's trajectory begins at a specific time within the video and is annotated with a visibility flag which indicates whether the point is visible or occluded in each frame. A point can be tracked starting from any frame, as long as it is visible in the initial frame where tracking begins. The task is to estimate each point's trajectory and predict its visibility across the video. Initially, only the starting trajectory and visibility are known to the tracker; the remaining trajectory and visibility must be predicted.

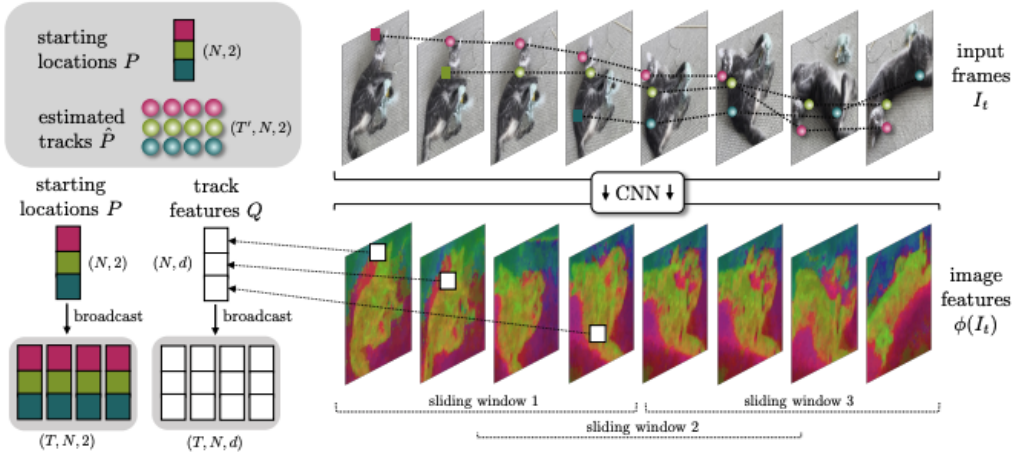


Figure 2.3: Shows the high-level overview of the training procedure of CoTracker's transformer model. Taken from Karaev et al. [30]

CoTracker utilizes a transformer network built upon a foundation of convolutional neural networks. Tracks are encoded as input tokens, which include position, visibility, appearance and correlation of the tracks. The transformer processes these tokens iteratively, progressively refining the track predictions for context assimilation. The reason why CoTracker can handle long videos is there windowed inference approach: If videos are longer than the maximum window length, the video is divided into overlapping windows. A downside of this approach is the limitation of the ability to track points through long-term occlusions that exceed the size of a single window. Therefore, the primary loss that [30] has introduced

for track regression is the following:

$$\mathcal{L}_1(\hat{P}, P) = \sum_{j=1}^{J} \sum_{m=1}^{M} \gamma^{M-m} \|\hat{P}(m, j) - P(j)\| \tag{2.5}$$

where γ compensates for early transformer updates. $P(j)$ contains the ground-truth trajectories restricted to window j. For the visibility flags, cross-entropy is used as a second loss function during training. Another challenge with CoTracker is the computational complexity of the transformer, which scales with four times the number of points being tracked. This makes it less suitable for dense prediction tasks. However, this limitation is not a concern for this thesis, as our focus is on sparse tracking. Given that the model's limitations do not hinder our objectives and its strong performance relative to other benchmarks, we have chosen to use CoTracker as a baseline in our research.

As previously explained, we define a factorized estimation problem as combining a 2D trajectory estimated by a 2D tracker with depth information estimated by a monocular depth estimation model. To incorporate CoTracker into a factorized estimation approach, we also need to estimate depth separately. We will now explore how we can achieve this.

2.4 Monocular per pixel depth estimation

We are now faced with the challenge of determining depth. This task is referred to as depth estimation. As observed by Liu et al. [37] current depth estimation methods can be divided into active and passive methods. While active methods refer to obtaining accurate depth from controlled beams and their reflections, passive methods refer to the use of imaging to obtain depth information. As active methods have been in use for a long time let's take a deep dive. [37] categorizes active methods in Structured light and laser scanning methods and Time-of-Flight (ToF) methods. An example of a ToF method, also traditionally used in vehicle velocity estimation, is a light detection and ranging (LiDAR) system as mentioned in Chapter 1. This system models the distortion of light patterns and from there infer light. LiDAR sensors, specifically, emit laser pulses towards objects in the environment and measure the time it takes for these pulses to bounce back, referred to as Time of Flight. Accurate depth estimations are derived from variations in these time measurements.

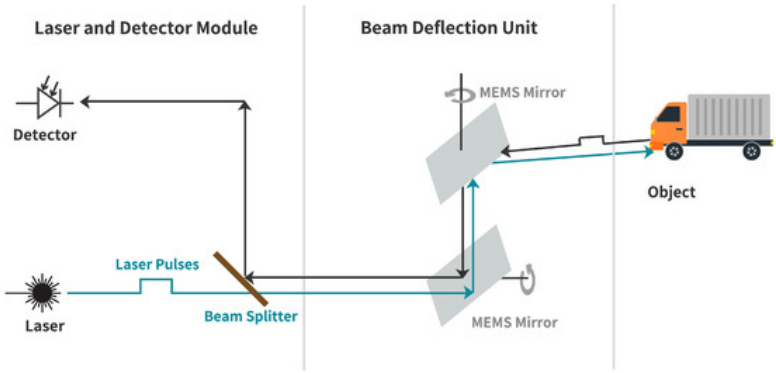


Figure 2.4: Schematic of a LiDAR system illustrating the Laser and Detector Module alongside the Beam Deflection Unit. The laser emits pulses that are directed towards the object via mirrors in the Beam Deflection Unit. Reflected light from the object is detected by the detector, allowing for the calculation of distance based on the time of flight of the laser pulses. This overview is taken from Haas et al. [23]

Although these sensors have become more affordable, their cost and energy consumption still are challenging factors for widespread deployment in mass-market autonomous systems, such as self-driving vehicles [37].

Although cameras surpass LiDAR in capturing denser and more detailed representations of the environment, LiDAR has traditionally been more effective at extracting distance information due to the challenges of depth estimation using only a single camera in a standard perspective view [49]. However, recent advancements in vision-based depth estimation have significantly improved accuracy, leading to software-driven methods. Using cameras for depth estimation is falls under passive methods which require lower software and hardware requirements, are more convenient for widespread deployment, but at the cost of lower accuracy [49]. To understand this better, we will dive into the various types of depth estimation techniques. Depth estimation (DE) can be classified in three categories: Monocular depth estimation (MDE), binocular depth estimation (BDE), and multi-view depth estimation (MVDE). Monocular estimation determines the depth, or distance to the camera, of objects in a scene using only a single image. BDE uses two images from different viewpoints to estimate depth, where image disparity is leveraged to calculate the distance between objects. The principle behind this type of depth estimation is similar to human 3D perception, relying on the triangulation of rays from two viewpoints. MVDE uses more than two viewpoints to estimate depth and is valued for its accurate depth information that can be obtained.

As mentioned previously, monocular depth estimation has become increasingly popular. This method is valued for its ease of interpretation and decent accuracy. Cameras are cost-effective and provide textured and rich information. Monocular depth estimation traditionally focused on metric depth in specific domains, such as vehicles or sports. However, there is now a shift towards zero-shot relative monocular depth estimation, which aims to create more generalizable models.

This problem referred to as single-image depth estimation (SIDE) can be classified in metric depth

estimation (MDE) and relative depth estimation (RDE). RDE aims to predict the relative order of objects in a scene based on depth. Since it does not account for scale, it is possible to train these models on diverse scenes and datasets. RDE models' lack of reliance on consistent camera information allows for greater generalizability across various environments. However, a downside is that RDE models lack the ability to provide meaningful metric depth, making them unsuitable for applications requiring precise measurements. Conversely, MDE models are designed to estimate depth in physical units, making them popular for tasks requiring physical measurements, such as physical estimation tasks. MDE models are typically trained on specific datasets, which limits their generalizability to new environments, reducing their zero-shot capabilities. A significant challenge for MDE models arises from the issue known as scale ambiguity, a common problem in monocular depth estimation where a single camera setup does not inherently provide absolute scales. This ambiguity makes it difficult to determine the actual physical distance to objects from a single image, complicating efforts to apply these models outside of their training datasets without additional calibration or data. Recently, several state-of-the-art depth estimation models focusing on these deep learning techniques have been released. These include PatchFusion [34], ZoeDepth [5], Marigold [31] and DepthAnyhting V2 [74]. The trend began with MiDAS [50], released in 2019, which set the benchmark for robust monocular depth estimation. Over time, MiDAS became a standard in the field, leading to the development of three additional versions, ending with MiDAS v3.1 [6]. Despite its limitations in providing consistent estimations as it is a RDE model, MiDAS remains a significant backbone for many other state-of-the-art models. We will now explore two of these state-of-the-art models that are used in our paper.

2.4.1 ZoeDepth

A novel model that goes against MDE and RDE's limitations, and therefore makes a step toward universal depth estimation is ZoeDepth [5]. Built on top of MiDAS [50] focusing on solving the spatial consistency problem of MiDAS, ZoeDepth combines the tasks of relative depth estimation and metric depth estimation. With a design focused on producing metric depth estimates, ZoeDepth is able to differentiate between the foreground, midground, and background elements within an image in metric units. The model performs well in even challenging scenarios such as regions with low texture or occlusions and also unseen environments. Their proposed two-stage framework works as following: Firstly they have trained the relative depth estimation tasks using a common encoder-decoder architecture, pre-training on a diverse set of 12 datasets, including BlendedMVS [75], TartanAir [64], ReDWeb [67], NYU Depth V2 [44], ApolloScape [62], MegaDepth [35], DIML-Indoor [11], HRWSI [68], KITTI [20], and 3D Movies [58]. The diversity in this selection leads to good generalization. This stage is followed by the fine-tuning part, here is where their improvement comes into place. Inspired by LocallBins [4], they improve the pre-training architecture by adding heads, specifically for the metric depth estimation. This comes from the idea of estimating a set of depth values instead of single depth value per pixel. These heads are finetuned for metric depth estimation, with a primary focus on KITTI (outdoor) [20] and NYU Depth V2 (indoor) [44] dataset, where they use one light-weight metric head tailored per domain.

The training utilizes the scale-invariant log loss for pixel-level supervision as seen before in Local-

Bins [4], introduced by Eigen et al. [14]:

$$\mathcal{L}_{\text{pixel}} = \alpha \sqrt{\frac{1}{T}\sum_i g_i^2 - \frac{\lambda}{T^2}\left(\sum_i g_i\right)^2} \tag{2.6}$$

where $g_i = \log \tilde{d}_i - \log d_i$ and the ground truth depth d_i and T denotes the number of pixels having valid ground truth values.

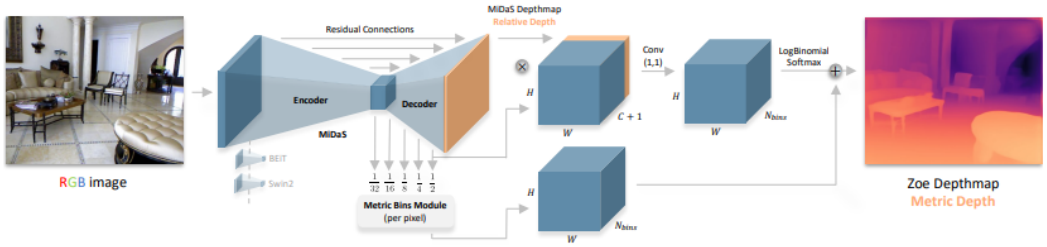


Figure 2.5: Shows the high-level overview of the model architecture of ZoeDepth. Taken from Bhat et al. [5]

The full model takes an RGB image as input and outputs a predicted depth map. A key improvement in ZoeDepth is its use of a lightweight head with a novel bin adjustment Due to the mechanism known as the metric bins module, which is tailored for different domains the model automatically routes each input image to the appropriate head using a latent classifier. As mentioned in their paper [5], the approach of their flagship model ZoeD-M12-NK significantly outperforms the state-of-the-art on the trained datasets, achieving a 24.3% improvement in absolute relative error. Additionally, their paper [5] demonstrates zero-shot capabilities by outperforming SOTA on 7 untrained metric datasets. These remarkable results, aligned with the goal of robust and reproducible velocity estimation in this paper, makes a the model a good fit for our methods.

2.4.2 Synthetic data vs Real data

When building depth estimation models, researchers utilize various datasets that can be broadly categorized into real scenes and synthetic scenes. Real scene datasets such as KITTI [20], NYU Depth V2 [44], and Make3D [56] provide authentic environmental data, while synthetic datasets like SYNTHIA [53] and SceneNet RGB-D [40] offer artificially generated scenes. The selection of an appropriate dataset is crucial and should align with the specific research objectives. For instance, synthetic data not only addresses the scarcity of real data, especially in fields like autonomous vehicles, but is also essential in scenarios where obtaining real data is impractical or dangerous. Consider the task of training a network to prevent car accidents. To accomplish this, the network requires extensive training data on crashes. However, collecting large datasets of real car crash data is both prohibitively expensive and poses significant safety risks. Therefore, simulating car crashes to generate the necessary training data becomes a practical and beneficial solution.

While synthetic data offers clear advantages, it also presents challenges, particularly in minimizing the sim-to-real gap, as synthetic data often exhibits statistical properties that differ from real-world data.

This makes the success of McCraith et al. [41] as highlighted in Section 2.1.1, who achieved impressive results using only synthetic data, particularly noteworthy. The authors of DepthAnythingV2 [74] recognize this issue, referring to it as a distribution shift between synthetic and real images, which can lead to discrepancies. They emphasize the importance of effective synthetic-to-real transfer in monocular depth estimation, with their methods specifically designed to bridge this gap. By doing so, they aim to overcome the limitations associated with synthetic labeled data, such as label noise and the potential to overlook fine details, while addressing the inherent challenges of using real labeled data.

We will now take a detailed look at this model to understand how the authors have achieved these goals.

2.4.3 DepthAnything V2

DepthAnything V2 [74] is a recent metric depth estimation model that leverages both real and synthetic data. DepthAnything V2 is a follow-up of DepthAnything [73] which is built on top of the DinoV2 architecture [46]. In contrast to ZoeDepth, which has focused on improving their model architecture by incorporating new components, the authors of DepthAnything V2 argue that the biggest bottleneck for model performance, is not necessarily the model architecture itself, but rather the quality of the data on which the model is trained. Therefore, they solely make use of synthetic data, since synthetic datasets have direct access to all the 3D information needed, made by the graphics engine itself which creates the scene. Therefore DepthAnything V2 is exclusively trained on 5 precise synthetic datasets and 8 pseudo-labeled real datasets including: BlendedMVS [75], Hypersim [52], IRS [63], TartanAir [64], VKITTI2 [7], BDD100K [76], Google Landmarks [66], ImageNet-21K [51], LSUN [77]. These are all a mix of indoor and outdoor scenes and give DepthAnything V2 a strong generalization capability. The model can be divided in three main steps:

1) Teacher training
2) Pseudo-labeling real images
3) Student training

The teacher model is trained for supervised depth estimation on the precise synthetic datasets. There are two loss functions used for training here which have been proposed by MiDaS [50]. Consider an image with valid ground truth, containing M pixels. Let $\boldsymbol{\theta}$ represent the prediction model's parameters. The predicted disparity for these pixels can be denoted as $\mathbf{d} = \mathbf{d}(\boldsymbol{\theta}) \in R^M$, and the corresponding ground-truth disparity is given by $\mathbf{d}^* \in R^M$. The individual pixel values are indexed. $\hat{\mathbf{d}}$ and $\hat{\mathbf{d}}^*$ are the scaled and shifted versions of the predictions and ground truth, respectively. The function ρ specifies the type of loss function used. The scale- and shift-invariant loss for a single image is defined as:

$$\mathcal{L}_{\text{ssi}}(\hat{\mathbf{d}}, \hat{\mathbf{d}}^*) = \frac{1}{2M} \sum_{i=1}^{M} \rho\left(\hat{d}_i - \hat{d}_i^*\right) \tag{2.7}$$

The gradient matching loss, where $R_i = \tilde{d}_i - \tilde{d}_i^*$, and R^k represents the difference of disparity maps at the k-th scale, is formulated as:

$$\mathcal{L}_{\text{reg}}(\hat{\mathbf{d}}, \hat{\mathbf{d}}^*) = \frac{1}{M} \sum_{k=1}^{K} \sum_{i=1}^{M} \left(\left| \nabla_x R_i^k \right| + \left| \nabla_y R_i^k \right| \right) \tag{2.8}$$

Combining the scale- and shift-invariant loss, which encourages learning general relative depth relationships, together with the gradient loss, which encourages preserving sharp edges, we get to the final loss where N_l represents the size of the training set:

$$\mathcal{L}_l = \frac{1}{N_l} \sum_{n=1}^{N_l} \mathcal{L}_{\text{ssi}}(\hat{\mathbf{d}}^n, (\hat{\mathbf{d}}^*)^n) + \alpha \mathcal{L}_{\text{reg}}(\hat{\mathbf{d}}^n, (\hat{\mathbf{d}}^*)^n) \tag{2.9}$$

The weight ratio of the scale- and shift-invariant loss and gradient matching loss is set to 1:2, respectively.

With the trained teacher model, millions of unlabeled images can now be annotated in step 2. This results in a large pseudo-depth label dataset. In step 3, the student model is further trained exclusively on pseudo-labeled images. This does not include real labeled datasets or synthetic datasets. This full method makes DepthAnything V2 capable of handling transparent and reflective surfaces well leveraging synthetic data only.

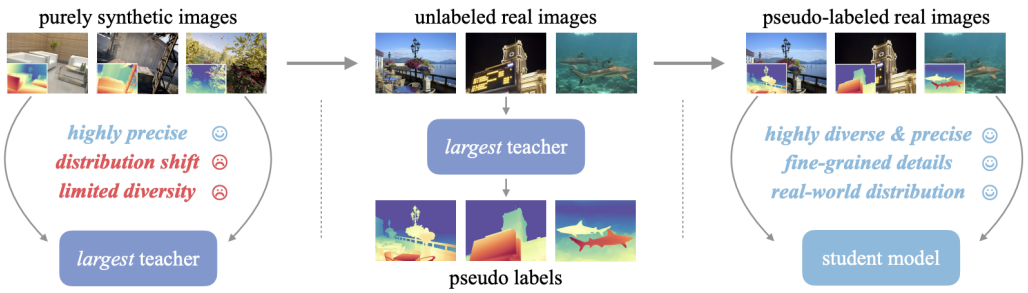


Figure 2.6: Shows the high-level overview of the training procedure of DepthAnyhting V2. Taken from Yang et al. [74]

This approach, which emphasizes dataset quality to create a more generalized model, provides a strong and reliable baseline for our research next to ZoeDepth.

Chapter 3

Method

This chapter provides a detailed description of the methods employed in this study. We begin with an overview of the investigations conducted (Section 3.1), followed by a formal definition of the problem (Section 3.2). We then discuss the dataset (Section 3.3) and go through the velocity derivation using the dataset's features (Section 3.4). that are needed for this research. The chapter concludes with a detailed description of the three proposed methods, their implementation details (Section 3.5), and how they are integrated within established benchmarks (Section 3.6).

3.1 Overview of investigation

Further details on the objectives outlined in Section 1.4 are provided below. This study aims to conduct a comprehensive evaluation of sparse 3D point tracking methods for estimating the real-world velocity of moving objects in video sequences. The research will:

1. Critically compare existing state-of-the-art approaches in long-range 3D point tracking for point velocity in image space.
2. Integrate these approaches into benchmarks for real-world velocity.
3. Analyze the limitations of the best performing benchmark for real-world velocity.

Through this investigation, we seek to:

a) Identify the current constraints and challenges in different state-of-the-art 3D tracking methods for point velocity in image space and real-world velocity.

b) Establish the scope and effectiveness of the top performing developed benchmark in the domain of real-world velocity estimation.

By doing so, we seek to answer the following main research question, while also addressing its scope:

Can current long-range 3D point tracking methods provide accurate real-world velocity estimations of moving objects in videos?

To answer the main research questions, the following subquestions are addressed:

(1) How do the different state-of-the-art 3D point tracking methods compare for point velocity in image

space?

(2) How can these methods be integrated into benchmarks for real-world velocity estimation of moving objects in a video?

(3) What are the current limitations of state-of-the-art 3D tracking methods for real-world velocity error?

3.2 Problem Definition

We define the problem of 3D velocity estimation where the objective is to estimate the velocity vector $\mathbf{v} = [x, y, z]^T$ per frame in a video, with x and y representing the trajectory in the image plane, and z representing the depth component. This estimated vector $\mathbf{v}$ is to be compared against the ground truth velocity vector $\mathbf{V} = [X, Y, Z]^T$ per frame in a video, where X, Y, and Z correspond to the true velocities in world coordinate system. Given a video sequence containing $\mathcal{T}$ frames, our goal is to evaluate whether the estimated 3D trajectory $\mathbf{v}(t)$ aligns with the ground truth $\mathbf{V}(t)$ at each time step t, meaning each frame in a video. This is achieved using different methods to estimate the trajectory and comparing the resulting vectors to the ground truth across time. The used methods and their implementation are described in Section 3.5. Formally, we aim to verify the accuracy of the estimated velocity vectors by comparing:

$$\mathbf{v}(t) \quad \text{vs} \quad \mathbf{V}(t)$$

for each time step t, where $\mathbf{v}(t)$ is derived and estimated from the methods being evaluated. These approaches produce velocity estimates in the camera frame, which must then be transformed into the global frame using a coordinate transformation function $\mathbf{T}(\mathbf{v})$. The detailed calculation of this transformation is explained in Section 3.4. Formally, we check whether:

$$\mathbf{T}(\mathbf{v}(t)) \approx \mathbf{T}(\mathbf{V}(t))$$

The objective is to assess whether the methods provide estimates that are consistent with the ground truth velocities in practice. This problem definition sets the stage for to answer our research questions.

3.3 Dataset

In this paper, we utilize a subset of DriveTrack [3], a benchmark and data generation framework, annotated on the Waymo Open Dataset [60] published by Waymo and Google. Although their data generation workflow is compatible with other autonomous driving datasets, our focus is on the Waymo dataset on which they also published their annotations. The DriveTrack benchmark includes 1000 scenes captured across three different cameras, resulting in approximately 10,000 videos with a resolution of 1280×1920 pixels and a frame rate of 10 fps. Each video captures a single object with an average of 100,000 trajectories per video. The dataset is split into 800 scenes for training, 100 scenes for validation, and 100 scenes for testing, translating to about 8,000, 1,000, and 1,000 videos respectively, with an average video length of 84 frames.

The features annotated by DriveTrack benchmark on the Waymo dataset are provided in Table 3.1.

Due to computational limitations, we use only a subset of the DriveTrack benchmark for the purpose of this research. The first step involves selecting a subset of video sequences from the DriveTrack dataset.

Matrix	Description	Per Frame/Per Video
Camera Intrinsic Matrix	This matrix contains the internal parameters of the camera. It describes how 3D points in the camera's coordinate system are projected onto the 2D image plane. When the inverse of this matrix is applied, it allows the projection of 2D image points back into the 3D camera coordinate system.	Per Video
Camera Extrinsic Matrix	This 4x4 transformation matrix represents the camera's position and orientation relative to a fixed world coordinate system for each video. Therefore, this is a single transformation for the whole video sequence. When applied directly, this extrinsic matrix transforms coordinates from the camera sensor frame to the vehicle frame.	Per Video
Vehicle Pose Transform Matrix	This 4x4 transformation matrix describes the vehicle's position and orientation in the world coordinate system for each individual video frame. When applied, it transforms coordinates from the vehicle frame to the global frame, effectively mapping them to real-world coordinates.	Per Frame

Table 3.1: Relevant coordinate transformation matrices extracted from the Waymo Open Dataset

Specifically, the first 50 video sequences were chosen based on the following criteria:

1. Each sequence contains fewer than 100 frames.
2. COLMAP is capable of accurately estimating the extrinsic parameters for these sequences.

This results in a small evaluation dataset referred to the mini-eval-dataset in the rest of this paper. The videos in this dataset range from 24 to 97 frames each. These videos maintain a resolution of 1280×1920 pixels at 10 fps, with 55k to 100k tracked points per clip. To align the ground truth data from the Waymo Open Dataset with the selected DriveTrack sequences, corresponding frames between the two datasets are identified. This alignment is highly important for accurate synchronization, because we need the camera intrinsics and extrinsics, but these are not avalaible in the annotated features in the DriveTrack dataset. Therefore, these are sourced from the Waymo Open Dataset, once the selected videos from DriveTrack are matched with the right frame from The Waymo Open Dataset. The Waymo Open Dataset includes five cameras per car, and we consider each camera on the ego vehicle separately. Therefore, we generated DriveTrack ourselves using only the 50 videos as our mini-eval-dataset and sourced additional information from the Waymo Open Dataset. The first 50 videos were chosen based on two criteria: No more than 100 frames and COLMAP was able to make an estimation of the camera poses. The Huggingface Filesystem is used to retrieve the Drivetrack data, since the data is stored only there. These videos from the resulting mini-eval-dataset are in RGB format and directly usable as input in the frameworks described in 3.5. Therefore no further data preparation is needed.

#clips	#trajectories per clip	#frames per clip	#videos	resolution	fps
50	55k-100k	24-97	50	1280×1920	10

Table 3.2: Overview statistics of the subset of DriveTrack dataset used for this research referred to as mini-eval-dataset

3.4 Velocity derivation

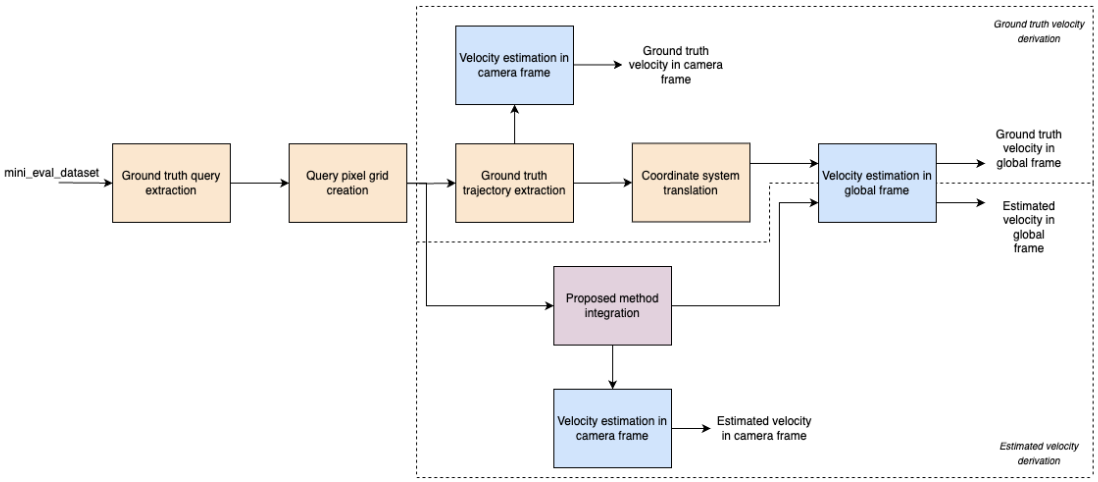


Figure 3.1: High-level procedure for velocity derivation and method integration. The process starts with the mini-eval-dataset, from which ground truth queries and trajectories are extracted. These are used to create a query pixel grid and perform coordinate system translation, resulting in ground truth velocity estimations in both camera and global frames. The methods used in this pink block are detailed in Section 3.5. The estimated velocities from these methods are eventually compared with the ground truth velocities within both camera and global frames.

Figure 3.1 shows the complete procedures of obtaining ground truth velocity as well as the estimated velocity tailored to the DriveTrack dataset. The steps will be explained in the same order.

Ground truth query extraction

For each video sequence in the mini-eval-dataset, we extract ground truth data related to selected query pixels from the Waymo Open Dataset. The process begins by linking the Waymo Dataset with the Drivetrack dataset, enabling the extraction of several critical data sources. From Drivetrack, we obtain all available ground truth trajectories by extracting the 2D image tracks, referred to as *Tracks* in Table 3.1, which serve as the reference points for tracking and analysis. Additionally, from the Waymo Open Dataset, we extract the camera intrinsic matrix, camera extrinsic matrices per video, and vehicle pose transform matrices for each frame. A detailed explanation of these matrices will be explained later in this section.

Query pixel grid creation

Given the variability in the number of available ground truth tracked pixels per frame, ranging from 55.000 to 100.000, an evenly spaced grid of query pixels is generated for each video sequence. This grid standardizes the number of pixels used in the analysis to ensure consistency across different video sequences, as the number of ground truth tracked pixels per video vary. The grid has been evenly spaced using the K-means algorithm [25]: the closest points to each cluster center are used as the grid points. The function to generate the grid points contains of $\mathcal{X}$ points with the corresponding point index and the first visible frame index. The process for obtaining this optimal quantity $\mathcal{X}$ is detailed in Section 3.6.1. This is done using the same set seed (42) for all frameworks to ensure consistency. The pixels within this grid are then referred to as query pixels, and subsequently used to calculate the trajectories of the moving points and make these estimations comparable with the ground truth. To ensure correct tracking, the method also incorporates the ground truth visibility until the query pixel becomes visible in the video. This ground truth is extracted from the DriveTrack annotations in Table 3.1. This ensures that the point is only tracked from the moment it first becomes visible. This is needed, because tracking a point when it is not visible would mean that it would not track a point on the car leading to unnecessary errors.

Ground truth trajectory extraction

The ground truth trajectories of the identified query pixels are extracted from the *Points3d* feature from DriveTrack in Table 3.1. These trajectories provide a baseline, allowing for a direct comparison with the trajectories estimated by our proposed methods which we will dive in later in this chapter. Initially, the extracted trajectories are now represented within the camera coordinate system. We will now dive into how we derive the velocity of these trajectories in camera coordinate system as well as world coordinates coordinate system to also obtain the real-world velocity further than the point velocity in image space.

Before going into the procedure of ground truth velocity derivation from DriveTrack, it is essential to first understand how the relevant camera information is stored and utilized for coordinate transformations. These relevant transformation matrices are given in table 3.1.

Velocity estimation in camera frame

To calculate the ground truth point velocity in camera frame, we follow a systematic process involving several steps:

1. As the coordinates of these query pixels are in image space, we apply the camera intrinsics matrix to transform the coordinates to a camera coordinate system. Let q_i represent the query pixels, and K denote the camera intrinsic matrix. The transformation can be represented as:

$$q_i' = K \cdot q_i \tag{3.1}$$

   This calculation is repeated for every frame in the video.

2. Finally, the displacement of the pixels over time is used to calculate the velocity. The calculation takes into account the last frame where a point was visible compared to the current frame. Let Δd represent the displacement, f the frame rate (in frames per second), and n the number of frames since the point was last visible. The velocity v is calculated as:

$$v = \frac{\Delta d}{n \cdot f} \tag{3.2}$$

By systematically transforming all query pixels accordingly from image pixels to camera coordinates and then calculating the displacement over time, we obtain the ground truth velocity for each pixel in the camera frame of the video. This approach allows for accurate velocity estimation even when points temporarily disappear from view, by using the last known position and the number of frames since it was last visible. The use of the frame rate (f) in the calculation ensures that the velocity is correctly scaled to real-world time, regardless of the video's frame rate. This approach provides a robust method for velocity estimation in scenarios where object occlusion or temporary disappearance from the frame is common.

Velocity estimation in global frame

To calculate the ground truth point velocity in the global frame (real-world velocity), we build upon step 2 from the previous section. It's crucial to understand the DriveTrack's method for transitioning from image space to world coordinates, as we adopt this approach for consistency. We utilize DriveTrack's *Points3d* data, which provides 3D points relative to the vehicle frame. We apply the inverse of the *pose_transform* to convert these points to the world coordinate system. This *pose_transform* represents the transformation of the vehicle frame relative to the world frame. Finally, we track the movement of these world-coordinate points over time, calculating velocity based on their positional changes between consecutive time steps. This process involves the following two steps: The transformed pixels are further multiplied by the camera extrinsics. Let E denote the camera extrinsic matrix. This transformation is given by:

$$q_i'' = E \cdot q_i' \tag{3.3}$$

Next, we apply the vehicle frame pose transformation. Let V denote the vehicle frame pose transform. The transformation for each query pixel is:

$$q_i''' = V \cdot q_i'' \tag{3.4}$$

By transforming all query pixels obtained from *Points3d* and applying the inverse of the*pose_transform* and then calculating the displacement over time as in 3.2 we obtain the ground truth real world velocity for each frame in the video. This transformation effectively cancels out the ego-motion velocity of the camera, isolating the motion of the tracked points relative to the world coordinate system. Examples of DriveTrack video sequences and their ground truth velocities are shown in Figure 3.2.

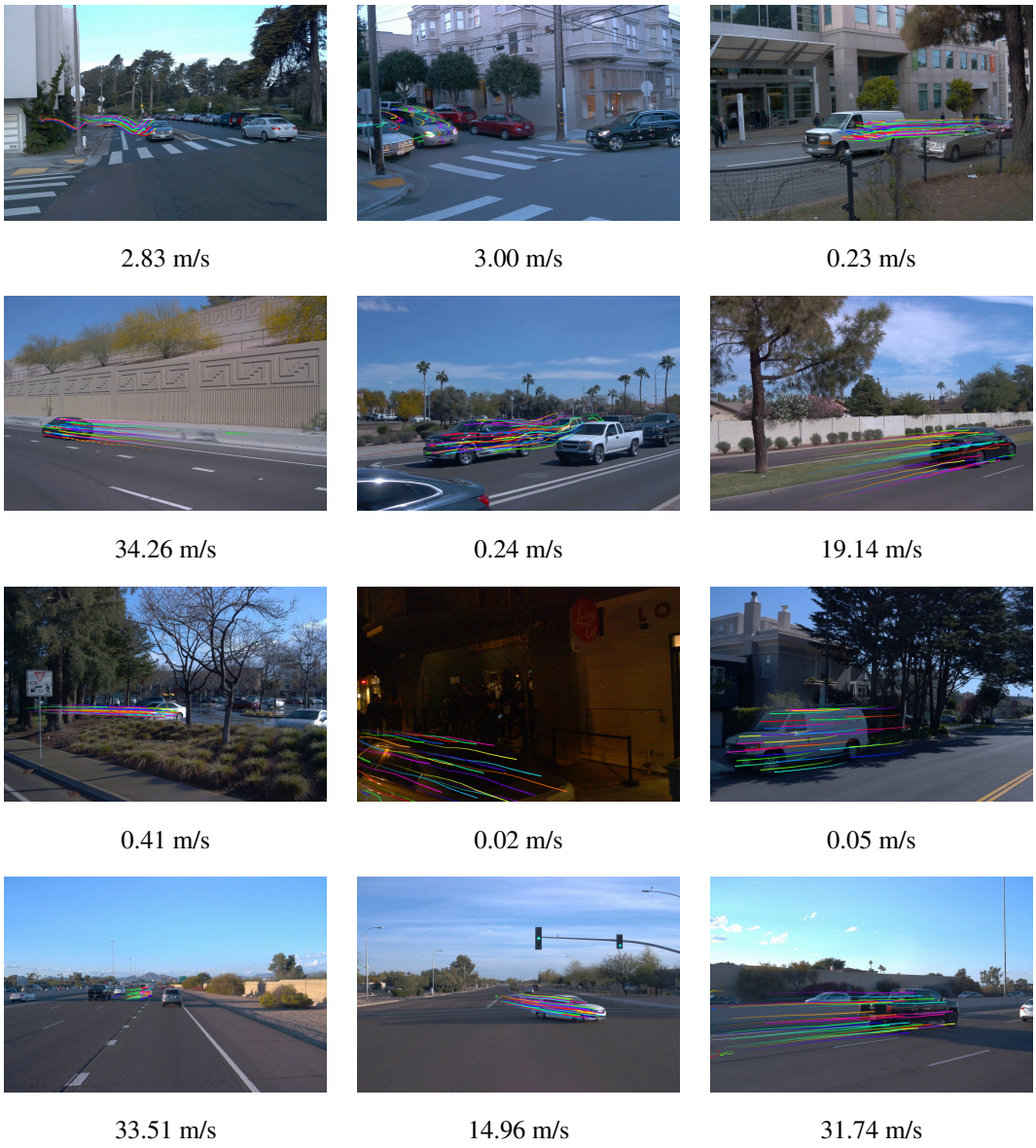


Figure 3.2: This figure shows a frame from various arbitrary video sequences, each labeled with its ground truth real-world velocity as outlined in Section 3.4. Observing the environments in which the vehicles are situated, the depicted velocities appear realistic. It is also noted that stationary vehicles still show some noise.

3.5 Implementation details of methods

There are a total of three methods built combining several state-of-the-art models and eventually compared. The first method is a factorized method with a combination of ZoeDepth and CoTracker. Secondly, we have a method existing of the same 2D tracker, but a different monocular depth estimation method DepthAnythingV2. Thirdly, we re-implement a joint method using SpatialTracker. All the details and history of these methods can be found in Chapter 2: Related Works.

All approaches have run on A100 GPU through Google Colab as we need Google Authentication to

access the Waymo Dataset [1]. These methods output image coordinates per frame. To eventually calculate the real world velocity of moving objects we need to transform these pixels to world coordinates in global frame following the calculations explained in Section 3.4

3.5.1 ZoeDepth + CoTracker

For the first method we utilized the pretrained model and PyTorch code from the official CoTracker codebase, running inference with bi-directional tracking mode enabled, while keeping all default parameters unchanged. Inference is carried out at a resolution of 512 × 384 for DriveTrack on a A100 GPU.

For ZoeDepth we used the pretrained their largest ZoeD_M12_NK model trained on the NYU Depth v2 indoor dataset [45] and the KITTI dataset [20]. We used their PyTorch code from the official codebase and run inference with the default parameters. Inference is performed on a resolution of 720 × 480 for DriveTrack, where the original resolution was too large for running infrence in this model on a standard GPU. Inference was performed on A100 GPU. Inference of this approach on a clip takes on average 9.1 seconds.

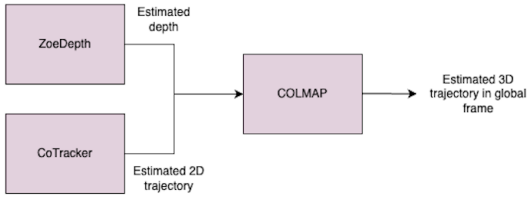


Figure 3.3: The ZoeDepth + CoTracker method. The method integrates depth estimation from ZoeDepth and 2D trajectory estimation from CoTracker. These outputs are fed into COLMAP to derive an estimated 3D trajectory in the global frame.

3.5.2 DepthAnythingV2 + CoTracker

For the second method we utitlize CoTracker the same way, but use the pretrained and fine-tuned model on the Virtual Outdoor KITTI2 dataset [8] of DepthAnythingV2 has been used. Specifically, we use the base model Depth_Anything_V2_Metric_Large. This is their largest base model made available with a size of 335,3M parameters. There are pre-trained models for both indoor and outdoor images, where in this paper the outdoor model is used. We re-implemented the model with its default parameters. Inference of this approach on a clip takes on average 8.8 seconds.

[1] Since the specific GPU cluster that was made available by UCL did not have the ability to authenticate with Google, Colab was used in this case. This is because the Google SDK is needed to authenticate to Waymo, because we are not the owner to this dataset. This issue was discussed on their Github page as well. As mentioned there, gsutil is the only way to connect to the dataset from a Python script which is not available on the UCL server

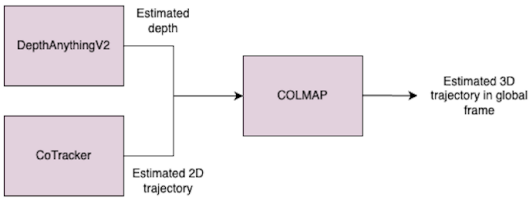


Figure 3.4: The DepthAnythingV2 + CoTracker method. The method integrates depth estimation from DepthAnythingV2 and 2D trajectory estimation from CoTracker. These outputs are fed into COLMAP to derive an estimated 3D trajectory in the global frame.

3.5.3 SpatialTracker

As a third method we use the 2-in-1 model SpatialTracker, specifically the one built on ZoeDepth as depth estimator which supports both RGB/RBGD videos input. SpatialTracker provides methods to downsample images, but we give it the full video without cropping or downsampling. Inference of this approach on a clip takes on average 4.3 seconds.

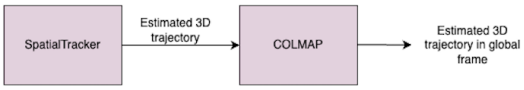


Figure 3.5: The SpatialTracker method. The method uses SpatialTracker to estimate a 3D trajectory, which is then processed by COLMAP to produce an estimated 3D trajectory in the global frame.

3.6 Benchmarks

To accurately calculate real-world velocity, it is essential to account for camera extrinsics. Our objective is to estimate this real-world velocity using a method that is as straightforward as possible, relying solely on camera information. To this end, we estimate these extrinsics while utilizing the ground truth intrinsics provided by the Waymo Open Dataset for each video. For this camera pose estimation we use COLMAP. COLMAP is implemented by first matching features of the frames given the ground truth intrinsic matrix using the OpenCV Camera Model. The next step is to perform Sequential Matching. Sequential Matching is used because we have Video frames, meaning that the frames are ordered. This is followed by a Sparse mapper to create the Camera Poses. Since COLMAP only estimates images it was able to map, the other frame poses are interpolated using the estimated poses from COLMAP. This returns for each image a camera pose. COLMAP has multiple functionalities, but these specific functions were used because they were best suited for the given task. Given our use of a sparse 3D tracking method, it is crucial to be critical in selecting the number of points to track. This selection process involves finding a balance between computational efficiency and accurate velocity estimation. To achieve this, we employ the procedure outlined in Section 3.6.1.

The detailed implementation of our method for real-world velocity estimation in the world coordinate system is outlined in Algorithm 1 for the joint method and in Algorithm 2 for the factorized

method.

Algorithm 3.1 Vehicle velocity estimation algorithm benchmark based on a joint method

Data: video, GT intrinsics, vehicle query pixels

Result: Mean of average car velocity

```
Load video
Load camera intrinsic
Create COLMAP estimation
Create grid of points on vehicle using k-means clustering on query pixels
3D_image, pred_visibility ← Predict 3D tracks and visibility (frame, points, 3)
3D_camera ← Convert 3D tracks to CC system using GT intrinsic
3D_world ← Convert 3D CC system to WC system using Colmap estimated extrinsics
average_car_velocity ← [ ]  for each frame in video do
    frame_velocity ← [ ]  for each point in grid points do
        if frame is first then
        end
        if not pred_visibility[frame, point] then
        end
        x, y, z ← 3D_world[frame, point]
        frames_ago_last_visible ← FindLastVisibleFrame(3D_world, frame, point, pred_visibility)
        prev_x, prev_y, prev_z ← 3D_world[frame - frames_ago_last_visible, point]

        time ← 1 / (fps × frames_ago_last_visible)
        displacements ← (x - prev_x, y - prev_y, z - prev_z)
        displacements_time ← displacements / time
        velocity ← ||displacements_time||_2

        frame_velocity += velocity
    end
    average_car_velocity += mean(frame_velocity)
end
return mean(average_car_velocity)
```

Algorithm 3.2 Vehicle velocity estimation algorithm benchmark based on factorised method

Data: video, GT intrinsics, vehicle query pixels

Result: Mean of average car velocity

```
Load video
Load camera intrinsic
Create COLMAP estimation
Create grid of points on vehicle using k-means clustering on query pixels
2D_tracks, pred_visibility ← Predict 2D tracks and visibility (frame, points, 2)
depth_estimates ← Predict metric depth for each point in 2D tracks
3D_camera ← Convert 2D tracks to 3D using depth estimates and GT intrinsic
3D_world ← Convert 3D camera coordinates to world coordinates using COLMAP estimated extrinsics
average_car_velocity ← [ ]  for each frame in video do
    frame_velocity ← [ ]  for each point in grid points do
        if frame is first then
        end
        if not pred_visibility[frame, point] then
        end
          x, y, z ← 3D_world[frame, point]
          frames_ago_last_visible ← FindLastVisibleFrame(3D_world, frame, point, pred_visibility)
          prev_x, prev_y, prev_z ← 3D_world[frame - frames_ago_last_visible, point]

          time ← 1 / (fps × frames_ago_last_visible)
          displacements ← (x - prev_x, y - prev_y, z - prev_z)
          displacements_time ← displacements / time
          velocity ← ||displacements_time||_2

          frame_velocity += velocity
    end
    average_car_velocity += mean(frame_velocity)
end
return mean(average_car_velocity)
```

3.6.1 Optimal sparse pixel density for object velocity representation

For each video v in the dataset, we computed the velocity $\mathbf{v}_q$ for all query pixels q. We then re-evaluated these velocities using grids with varying numbers of points n, where n ranges from 5 to 200 in increments of 5. Specifically, for each grid size n, we recalculated the velocity $\mathbf{v}_{q,n}$. Let $\mathbf{V}_q$ denote the ground truth

velocity for each query pixel q. We calculated the RMSE for each grid size n across all query pixels and all videos, defined as:

$$\text{RMSE}(n) = \sqrt{\frac{1}{|Q||V|} \sum_{v \in V} \sum_{q \in Q} \|\mathbf{V}_q - \mathbf{v}_{q,n}\|^2}$$

where $|Q|$ is the total number of query pixels, and $|V|$ is the total number of videos. This analysis was conducted using the mini-eval-dataset. Finally, we computed the average RMSE for each grid size n across all videos, yielding:

$$\text{Avg RMSE}(n) = \frac{1}{|V|} \sum_{v \in V} \text{RMSE}_v(n)$$

where $\text{RMSE}_v(n)$ represents the RMSE for video v at grid size n. Based on this analysis, we determine the optimal number of pixels required in a grid for tracking an object.

Chapter 4

Experiments and evaluation

Before diving into the details of these experiments and their outcomes, we first outline the consistency checks performed in Section 4.1 to ensure that our results are grounded in a reliable implementation of the methods. Following this, in order to answer our research questions, we introduce our success metrics (Section 4.2) and provide a comparative analysis of our three methods, evaluated in both the camera and global frames (Section 4.3 and Section 4.5). Finally, we conduct a comprehensive sensitivity analysis of the best-performing benchmark (Section 4.6 and Section 4.7) to thoroughly investigate its scope of effectiveness.

4.1 Cycle consistency checks

To maintain cycle consistency and ensure reliable results, we implemented and performed several checks.

1. **Camera coordinate system:** To validate our approach with the Waymo Dataset's diverse camera poses within the camera coordinate system, we implemented one primary check and one supplementary check.

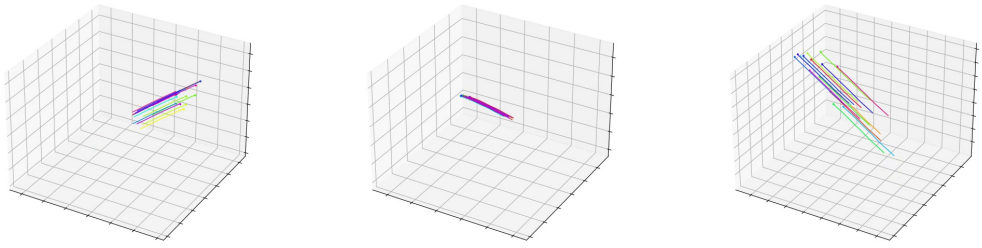
   1) The primary validation focuses on ensuring that the estimated and ground truth trajectories in the camera frame closely align. We verify this by visualizing both trajectories within the same coordinate system. As illustrated by the three arbitrary examples in Figure 4.1, the trajectories indeed follow a similar path, demonstrating that the implementation is successful.

   2) The supplementary check involved analyzing the consistency between two factorized frameworks, both of which utilize the same tracker, CoTracker. By substituting depth estimates from specific models with ground truth depth, we anticipated identical results across both frameworks. This consistency was expected to manifest in the RMSE velocity and position error measurements. Any variation in results would indicate either stochastic behavior in CoTracker's performance or potential implementation errors. However, our findings, as presented in Table 4.1, do not show discrepancies, thus validating both the consistency of our approach and the reliability of the implementation.

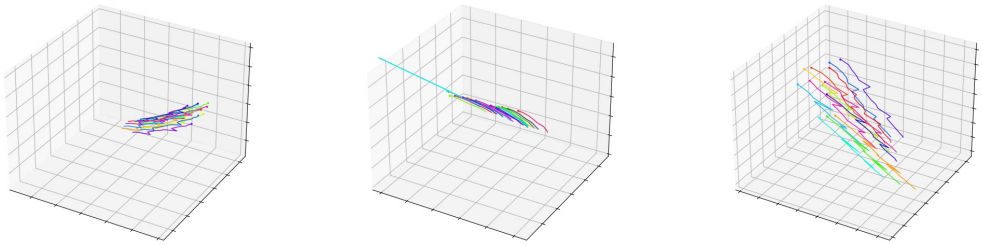
2. **World coordinate system:** To validate our approach using the Waymo Open Dataset's diverse camera poses within the world coordinate system, we conducted one primary validation check along with an additional observational assessment.

1) First, we verified that stationary vehicles registered zero real-world velocity, confirming correct application of coordinate transformations. Examples are shown in Figure 4.2. The minor error in the GT velocity is likely due to noise caused by for example bumps on the road. As described in Table 3.2, the videos have a frame rate of 10 FPS, making them susceptible to noise from small, sudden movements like bumps. Additionally, the Waymo Open dataset includes timestamps for each frame at 10 FPS. They also include in their data the timestamps of each frame. The difference between each timestamp is 0.1 second (10 FPS) but contains little noise on the microsecond level. This can lead to small errors in the GT Velocity.

2) Secondly, we compared linear velocities of the ego-vehicle extracted from the pose transform matrix with those provided by the Waymo Open Dataset, finding matching values. These linear velocities contain the velocity for the camera for on the X-axis, Y-axis and Z-axis. These can be compared to the relative pose of the vehicle on the X-axis, Y-axis and Z-axis, derived from the vehicle pose transform matrix. From these consecutive poses the velocity of each of the three can be calculated. These should be in line with the linear velocity values provided in the dataset. This is indeed the case. The checks ensured accurate handling of extrinsics matrices and data integration across different sources.



i. GT 3D trajectory ii. GT 3D trajectory iii. GT 3D trajectory



i. Est 3D trajectory ii. Est 3D trajectory iii. Est 3D trajectory

Figure 4.1: This figure presents three pairs of frame tracks comparing ground truth (top row) and predictions (bottom row) for a sanity check in CC system. The goal is to validate if the predicted tracks follow roughly the same trajectory as the ground truth. GT = ground truth, Est = estimated.

(A) (B)

Figure 4.2: This figure presents two arbitrary frames (A and B) from different videos with stationary vehicles as tracked objects. The goal is to validate that the velocity is indeed zero for these cars, so that the we know that the correct coordinate transformations are used as described in section 3.7

4.2 Measures of success

According to Llorca et al. [17] the most commonly used metric to measure speed error in traffic cameras is the Mean Absolute Error (MAE), either in its absolute or relative form. However, given that DriveTrack consists of videos of moving cars filmed with cameras mounted on other moving cars, the velocities of the subject and the ego camera could cancel each other out. Therefore, we use the Root Mean Squared Error (RMSE), a metric that has also been proposed in several studies [72] [39] [55]. Root Mean Squared Error (RMSE) is the standard deviation of the residuals, which are the prediction errors. Residuals indicate how far data points deviate from the regression line, and RMSE measures the spread of these residuals. In essence, RMSE indicates how concentrated the data is around the line of best fit. In our study, we calculate the RMSE for the estimated velocity as well as for the estimated velocity using the ground truth depth (Z). Additionally, we calculate the RMSE for the Z positional error and XY positional error to separately evaluate depth estimation and tracking estimation. The RMSE for the estimated velocity is given by:

$$RMSE_v = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(v_i - \hat{v}_i)^2} \tag{4.1}$$

where v_i is the true velocity, $\hat{v}_i$ is the estimated velocity, and n is the number of frames in a video. The RMSE for the Z positional error is calculated as:

$$RMSE_Z = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Z_i - \hat{Z}_i)^2} \tag{4.2}$$

where Z_i is the true position in the Z axis, and $\hat{Z}_i$ is the estimated position in the Z axis, and n is the number of frames in a video. The RMSE for the XY positional error is:

$$RMSE_{XY} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(X_i - \hat{X}_i)^2 + (Y_i - \hat{Y}_i)^2} \tag{4.3}$$

where X_i and Y_i are the true positions in the X and Y axes, respectively, and $\hat{X}_i$ and $\hat{Y}_i$ are the estimated positions in the X and Y axes, and n is the number of frames in a video.

Furthermore, we calculate the position error (PE) in 3D space to measure the quality of the positional accuracy of the 3D trajectory:

$$PE_{XYZ} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left((X_i - \hat{X}_i)^2 + (Y_i - \hat{Y}_i)^2 + (Z_i - \hat{Z}_i)^2\right)} \tag{4.4}$$

where X_i, Y_i, Z_i are the ground truth positions in the X, Y, and Z axes respectively. $\hat{X}_i$, $\hat{Y}_i$, $\hat{Z}_i$ are the estimated positions in the X, Y, and Z axes respectively. This equation calculates the average Euclidean distance between the estimated position $(\hat{X}_i, \hat{Y}_i, \hat{Z}_i)$ and the ground truth position (X_i, Y_i, Z_i), averaged over n frames, which represents the position error.

The error metrics employed in this study are calculated as the mean error per video. Subsequently in this paper, when conducting a comprehensive evaluation across the mini-eval-dataset rather than on individual videos, we denote aggregate measures such as average $RMSE$ and average MAE to represent the mean values of these metrics across all videos in the dataset.

4.3 Comparative analysis in camera coordinate system

In the camera frame, applying only the intrinsics without accounting for camera motion, our results in Table 4.1 shows performance variations among the tracking methods. SpatialTracker demonstrates the best performance with an average $RMSE_v$ of 11.84 for the fully estimated 3D trajectory. This is followed by the factorized method combining DepthAnything V2 and CoTracker, which achieves a score of 13.62. ZoeDepth paired with CoTracker shows the highest RMSE at 14.79, indicating comparatively lower accuracy in velocity estimation. Interestingly, when we substitute ground truth depth values instead of estimating depth alongside the trajectory, we observe an unexpected increase in velocity error across all methods. Most notably, SpatialTracker, despite having the lowest average $RMSE_v$ in the fully estimated scenario, exhibits the highest increase in average $RMSE_v$ with GT Z, approximately 68,24% higher when using ground truth depth. This method also shows the highest average $RMSE_Z$ estimation when depth is computed, surpassing both ZoeDepth + CoTracker and DepthAnything V2 + CoTracker combinations. Additionally, SpatialTracker displays the highest average $RMSE_{XY}$. These findings present an interesting paradox: SpatialTracker simultaneously achieves the lowest velocity error yet the highest trajectory error. This discrepancy asks for further investigation to understand the underlying factors of this.

Method	**Avg** $RMSE_v \downarrow$	**Avg** $RMSE_v$ **with GT Z** $\downarrow$	**Avg** $RMSE_Z \downarrow$	**Avg** $RMSE_{X,Y} \downarrow$
ZoeDepth + CoTracker	14.79	**16.95**	5.96	**3.27**
DepthAnything2 + CoTracker	13.62	**16.95**	**0.55**	7.67
SpatialTracker	**11.84**	18.64	0.85	10.45

Table 4.1: Comparison of tracking methods performance in camera coordinate system. The table presents average RMSE values for velocity (both estimated and with ground truth Z), Z-axis estimation RMSE , and XY-plane estimation RMSE across three different tracking approaches: ZoeDepth + CoTracker, DepthAnything2 + CoTracker, and SpatialTracker. The best results among all methods are highlighted in bold.

To better understand the relationship between average $RMSE_v$ and the trajectory quality, we compare average $RMSE_v$ against the average position error average PE_{XYZ} across the three different tracking approaches. This comparison is presented in Figure 4.3.

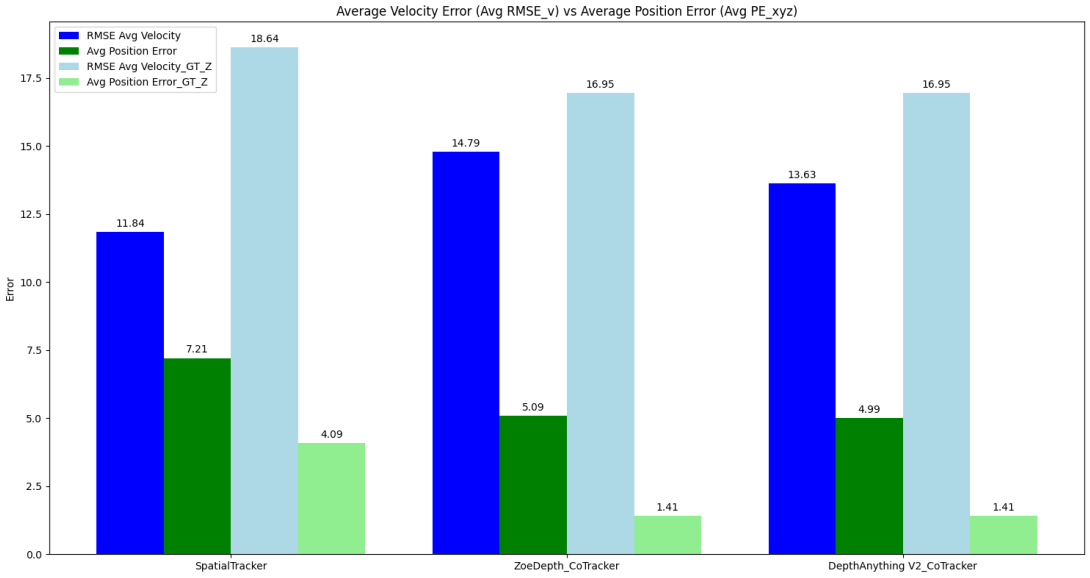


Figure 4.3: Comparison of RMSE velocity and position errors across three tracking methods in camera coordinate system: SpatialTracker, ZoeDepth + CoTracker, and DepthAnything V2 + CoTracker. The chart displays both average and ground truth (GT Z) measurements for each method, highlighting differences in performance for velocity estimation and position accuracy.

Figure 4.3 shows that within the camera coordinate system, an increase in average $RMSE_v$ is associated with a decrease in average PE_{XYZ} across all three methods. Specifically, for SpatialTracker, the average $RMSE_v$ rose from 11.84 to 18.64, marking a 57.43% increase, while average PE_{XYZ} concurrently dropped from 7.21 to 4.09, reflecting a 43.27% reduction. In contrast, ZoeDepth + CoTracker showed a relatively lower 14.6% increase in average $RMSE_v$ but achieved a significant 72.30% reduction in average PE_{XYZ}. DepthAnything V2 + CoTracker showed a 24.36% increase in average $RMSE_v$ alongside a 71.74% decrease in average PE_{XYZ}. These findings indicate that while Spatial-

Tracker has the highest increase in average $RMSE_v$ when using ground truth depth, it results in the lowest decrease in average PE_{XYZ}. Among the methods, ZoeDepth + CoTracker demonstrates the most difference in percentage change, with a relatively modest increase in average $RMSE_v$ paired with a substantial decrease in average PE_{XYZ}. This suggests that ZoeDepth + CoTracker achieves the most reduction in position error relative to the increase in velocity error compared to the other approaches.

4.4 Grid density analysis

A crucial step in constructing our benchmark, as outlined in 3.6.1, involves determining the optimal grid size for the query pixels.

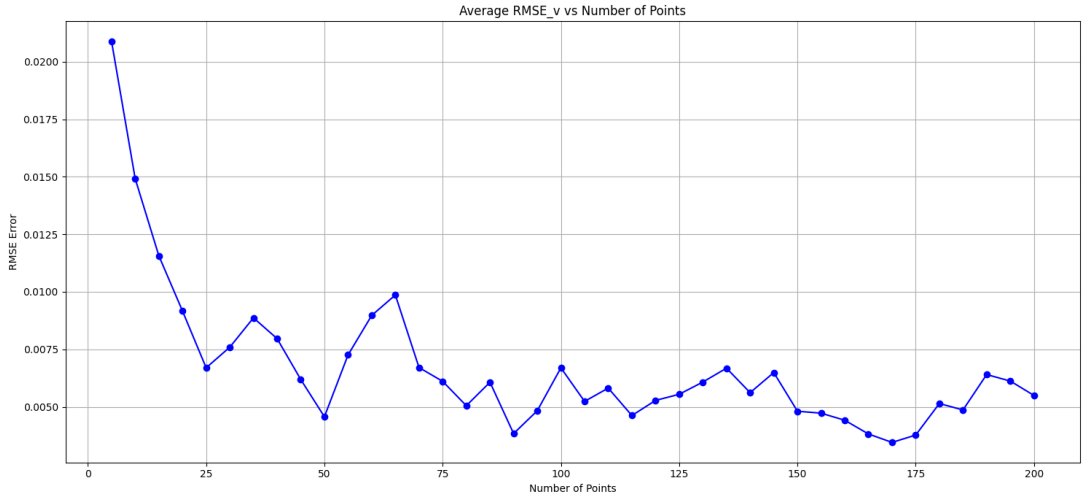


Figure 4.4: This figure represents the error of the ground truth velocity using different amount of points. The points were extracted by making a grid the same way used in the benchmark. With few points it has a GT velocity error of 0.02 m/s, but goes down starting from 25 points.

From Figure 4.4, it can be observed that the RMSE stabilizes with an increasing number of points. However, the overall reduction in error with additional points is minimal. Based on this observation and our resources for the research, we adopt a 30-point grid for tracking all objects. These 30 points correspond to the query pixels provided by DriveTrack, which consistently align with the vehicle being tracked. This choice balances computational efficiency with the trade-off between accuracy and resource use, allowing us to capture a realistic worst-case scenario. Coming back to our research goals: our primary focus is to evaluate whether these methods produces reasonable velocity estimations rather than optimizing for maximum accuracy. After building the benchmarks as further described in Section 3.6

4.5 Comparative analysis in world coordinate system

We find our results in some examples video sequences in Figure 4.7 and a complete overview in Table 4.2.

Method	**Avg** $RMSE_v \downarrow$	**Avg** $RMSE_v$ **with GT Z** $\downarrow$	**Avg** $RMSE_Z \downarrow$	**Avg** $RMSE_{X,Y} \downarrow$
COLMAP + ZoeDepth + CoTracker	22.07	**11.99**	161.00	9096.29
COLMAP + DepthAnything2 + CoTracker	21.73	**11.99**	161.21	**9096.05**
COLMAP + SpatialTracker	**11.91**	11.74	**160.87**	9096.66

Table 4.2: Comparison of tracking methods performance in world coordinate system. The table presents average RMSE values for velocity (both estimated and with ground truth Z), Z-axis estimation RMSE, and XY-plane estimation RMSE across three different tracking approaches: ZoeDepth + CoTracker, DepthAnything2 + CoTracker, and SpatialTracker. The best results among all methods are highlighted in bold

Table 4.2 shows the analysis of tracking methods in global frame, where we observe several trends that both align with and diverge from our findings in the camera coordinate system. SpatialTracker continues to demonstrate the lowest average $RMSE_v$, maintaining its superior performance even when ego-motion of the camera is taken into account. Interestingly, using ground truth depth values for the 3D trajectories yields more significant impacts in world coordinates compared to the camera coordinate system results in Table 4.1. This difference is expected, as the application of extrinsic parameters in world coordinates enhances the influence of Z-axis measurements on overall performance metrics. Examining the average $RMSE_Z$, we find a pattern that mirrors our observations in camera coordinates. SpatialTracker exhibits the lowest error at 160.87, followed closely by DepthAnything V2 + CoTracker and ZoeDepth + CoTracker with values of 161.21 and 161.00, respectively. This consistency across coordinate systems suggests inherent characteristics in each method's approach to depth estimation. The average $RMSE_{XY}$ ratios also maintains a similar relationship to those observed in camera coordinates. However, it's noteworthy that these values are substantially larger in world coordinates. One potential explanation for these elevated error magnitudes could be the unknown origin of the camera in world space. This uncertainty in the camera's absolute position might introduce additional variability into the XY plane measurements, though further investigation may be necessary to fully validate this hypothesis. Given that using GT Z has a distinct impact in the world coordinate system compared to the camera coordinate system, we also compare the velocity error of the methods with their position error in this coordinate system, as shown in Figure 4.5.

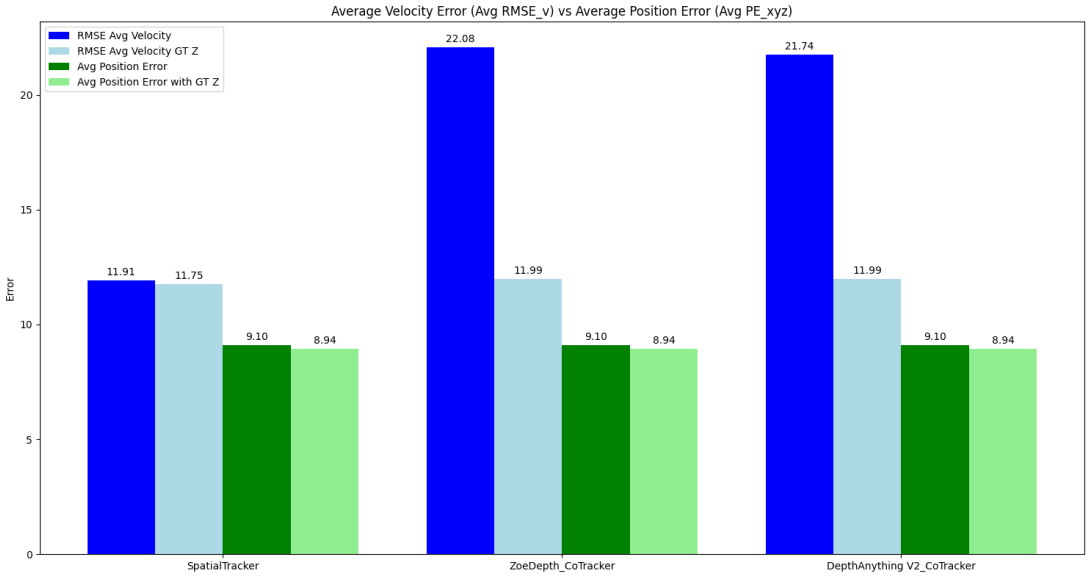
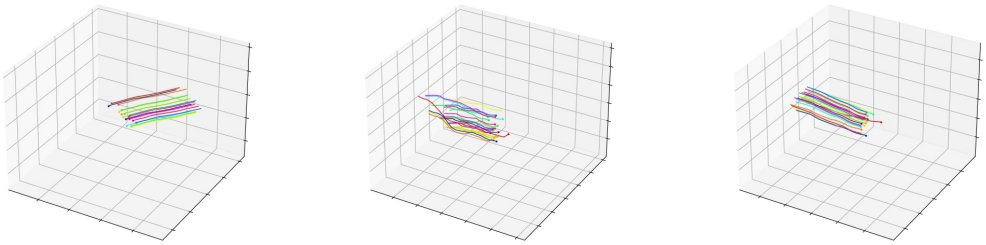


Figure 4.5: Comparison of RMSE velocity and position errors across three tracking methods in world coordinate system: SpatialTracker, ZoeDepth + CoTracker, and DepthAnything V2 + CoTracker. The chart displays both average and ground truth (GT Z) measurements for each method, highlighting differences in performance for velocity estimation and position accuracy. Position error is scaled down by a factor of 1000 to make the plot more visible.

Figure 4.5 presents the relation between position error and velocity error in world coordinate system, as similarly illustrated in Figure 4.3. Comparing the two figures, it is clear that there is a difference in the pattern of the average velocity error and the average position error: using GT Z results in a lower average position error and also a lower average velocity error, among all methods. We can also see this in an example trajectory in Figure 4.6. Additionally, we see that using GT Z results in identical error values among all methods indicating the effect of COLMAP's estimation of camera poses and the trackers. This is because COLMAP uses one scaling factor for the transformation. This scaling factor is based on the measured distance from COLMAP. Then this factor is automatically applied to all coordinates, which ensures all coordinates are scaled to the same world coordinate units. Their independent influence will be analyzed in more detail in Section 4.6. Figure 4.6 shows the difference between the ground truth, estimated, and estimated with ground truth z trajectories in world coordinate system. GT depth improves the tracks by making the tracks less noisy compared to the estimated trajectory.

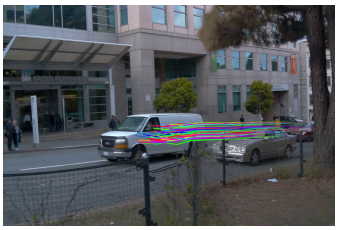


i. GT trajectory ii. Fully estimated trajectory iii. Estimated w GT Z trajectory

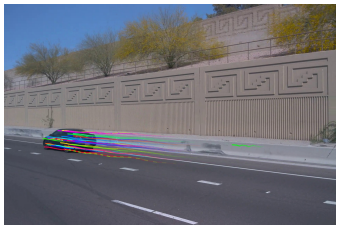
Figure 4.6: This figure displays three versions of the same trajectories of an arbitrary video for each method: ground truth, estimated, and estimated with GT Z in the world coordinate system. The figures reveal that the estimated trajectory exhibits more noise compared to when GT depth is incorporated. The differing directions in the coordinate systems used by COLMAP and Waymo, where the X coordinate points forward in Waymo and the Z coordinate points forward in COLMAP, account for the discrepancy in trajectory orientation. However, this difference does not affect the final velocity calculations. This visualization is adapted from the TAP visualization code. [[12]]

(A) Est: 10.17, GT: 2.83

(B) Est: 5.37, GT: 3.00



(C) Est: 16.59, GT: 0.23

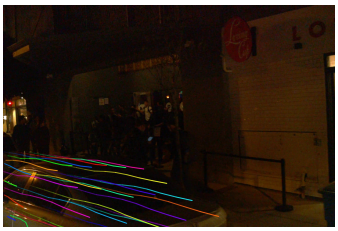


(D) Est: 9.59, GT: 34.26

(E) Est: 13.51, GT: 0.24

(F) Est: 10.09, GT: 19.14

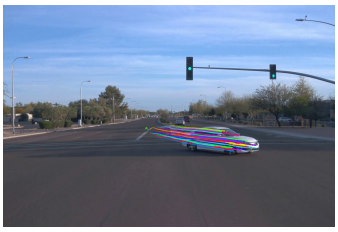
(G) Est: 12.00, GT: 0.41



(H) Est: 5.29, GT: 0.02

(I) Est: 4.79, GT: 0.05

(J) Est: 6.56, GT: 33.51



(K) Est: 9.35, GT: 3.21

(L) Est: 14.62, GT: 31.74

Figure 4.7: This figure presents frames from twelve arbitrary video sequences, each illustrating the estimated real-world velocity (Est) of a car alongside the ground truth (GT) real-world velocity for that same car. All velocities are placeholder values of 0.0. This comparison provides a high-level overview of the performance of our best-performing benchmark based on SpatialTracker. This visualization is adapted from the TAP visualization code. [[12]

4.6 Sensitivity analysis top benchmark

For deeper understanding of the factors influencing average $RMSE_v$, we now present a comprehensive sensitivity analysis. This analysis is conducted for the SpatialTracker-based framework in the world coordinate system, allowing us to also assess the impact of estimating extrinsics. The sensitivity analyses of the other two methods are given in the Appendix B.1. By systematically varying the use of estimated versus ground truth values for extrinsics, depth, and visibility prediction by the tracker methods, we can isolate the contribution of each component to overall tracking performance.

Metric	$RMSE_v$
Est extrinsics + Est depth + Est visibility	11.91
Est extrinsics + GT depth + Est visibility	11.74
GT extrinsics + Est depth + Est visibility	10.18
Est extrinsics + Est depth + GT visibility	12.97
Est extrinsics + GT depth + GT visibility	11.50
GT extrinsics + GT depth + GT visibility	8.62

Table 4.3: Sensitivity analysis results for SpatialTracker showing velocity RMSE under various combinations of estimated and ground truth inputs in world coordinates. Est = estimated, GT = ground truth. The RMSE is calculated on frame level.

Comparing the results in Table 4.3 with the average $RMSE_v$ of COLMAP + SpatialTracker (11.91 m/s) in Table 4.2 reveals an unexpected outcome. Surprisingly, substituting the visibility with its ground truth variant does not lower the average $RMSE_v$ compared to the fully estimated model. Only the use of ground truth extrinsics shows a slight improvement. Notably, incorporating ground truth depth only slightly increases performance while visibility predictions in isolation significantly worsen performance. These findings suggest complex interactions between system components and highlight the need for further investigation into their interdependencies.

4.7 Sensitivity analysis on outlier cases

To answer subquestion 3 for our main research question (Section 3.1), we study the primary causes of $RMSE_v$, based on our pest performing benchmark. We firstly look at the distribution of the 50 videos in mini-eval-dataset and their error scores and challenges in the respective videos.

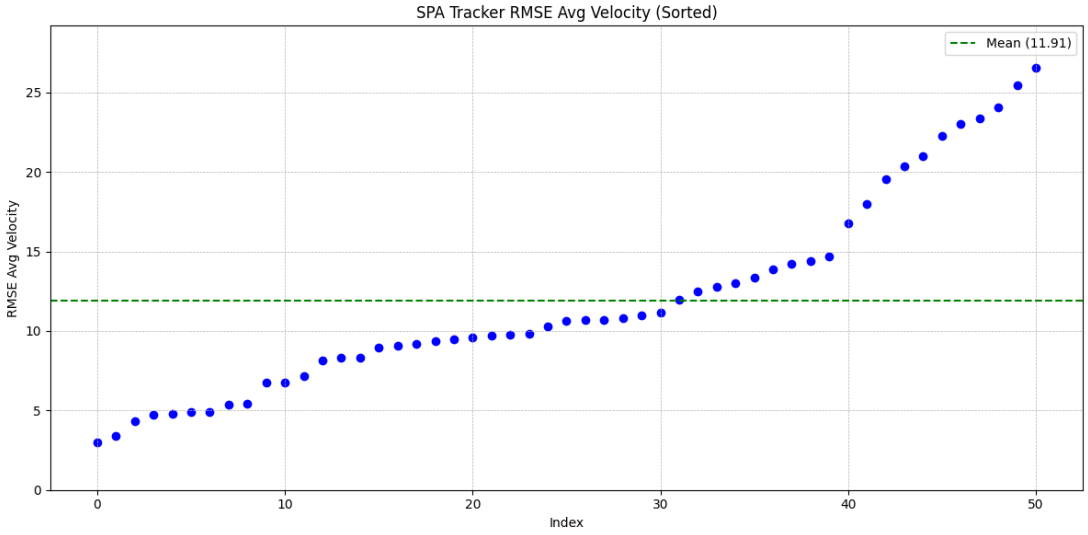
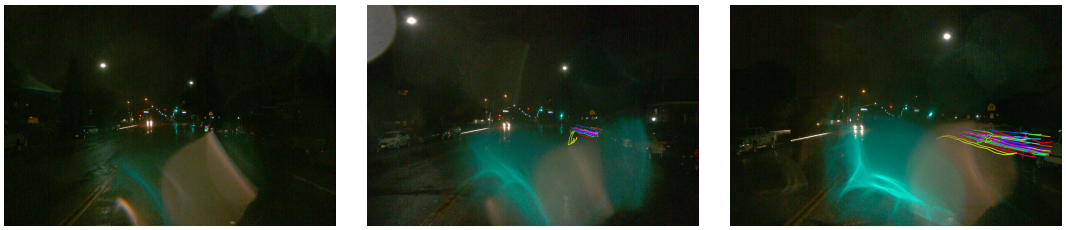


Figure 4.8: Illustration of the RMSE of average velocity measurements using the SpatialTracker-based benchmark. The blue dots represent individual scores per video sorted by increasing RMSE. The red dashed line indicates the mean RMSE (11.91). The x-axis represents the video index, and the y-axis shows the RMSE of average velocity. This graph helps to analyze the distribution of tracking errors and identify outliers in the velocity estimations.

Figure 4.8 shows us that among all videos the mean of the average $RMSE_v$ is 8.43 m/s. In 19 cases, the RMSE exceeds this mean value, indicating instances of lower accuracy. The majority of cases, 31 in total, exhibit RMSE values below the mean.

We now go through sensitivity analysis on the outlier cases to research what common features, environments or issues cause these outliers to find out what the primary sources of error are. This will set the scope of effectiveness of our benchmarks. Figure 4.9 up to to Figure 4.13 show the five videos furthest away from the mean visible in Figure 4.8. All visualizations in these figures are adapted from the TAP visualization code. [[12]]. These figures are sorted in descending error going from worst average $RMSE_v$ to best average $RMSE_v$ within these outliers. The figure captions explain each video and their environment.

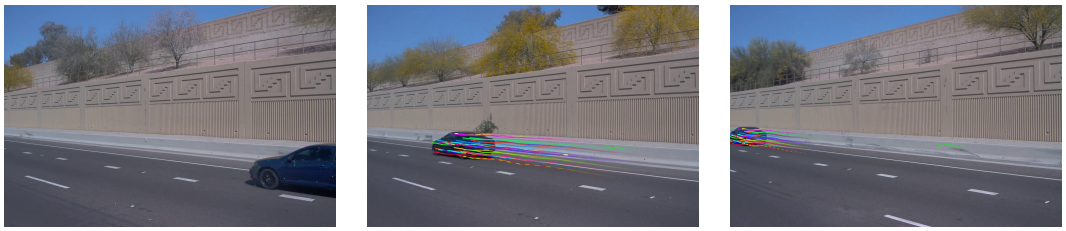
GT Velocity: 0.09 m/s Estimated Velocity: 10.19 m/s



(A) (B) (C)

Figure 4.9: This figure presents three non-consecutive frames (A, B, C) from the worst performing video sequence, demonstrating the movement of tracked objects over time. The colored pixels represent tracking predictions generated by the benchmark system, based on a 30-point grid overlaid on the target object. The target object is a standing car. Therefore, the GT Avg velocity is 0.09 m/s. The estimated Avg velocity is 10.19 m/s. These tracked points correspond to ground truth query pixels provided in the DriveTrack dataset. The progression from left to right illustrates the tracking process, showcasing how the system maintains object tracking through changing scenes and perspectives in low-light conditions.

GT Velocity: 34.26 m/s Estimated Velocity: 9.59 m/s



(A) (B) (C)

Figure 4.10: This figure presents three non-consecutive frames (A, B, C) from the third to worst performing video sequence demonstrating the tracking of a fast-moving vehicle. The colored pixels represent tracking predictions generated by the benchmark system, based on a 30-point grid overlaid on the target object, a car moving rapidly across the frame. The progression from left to right illustrates the system's attempt to maintain tracking on the vehicle as it travels along a highway with a decorative wall in the background. This sequence highlights the challenges faced by the tracking system in accurately following a high-speed object, particularly as it quickly changes position relative to the camera. The GT average velocity is 34.26 m/s against the estimated avg velocity of 9.59 m/s.

GT Velocity: 29.71 m/s Estimated Velocity: 3.49 m/s

(A) (B) (C)

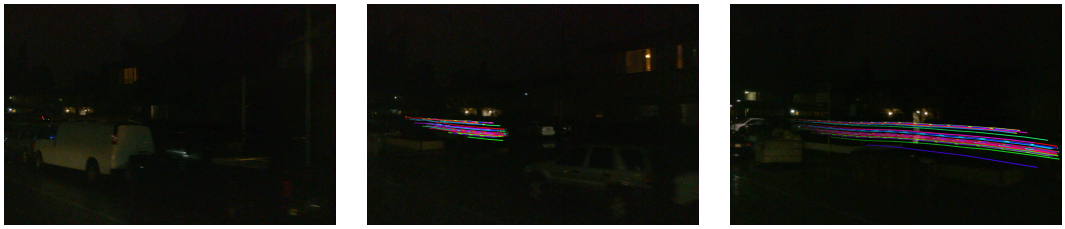
Figure 4.11: This figure presents three non-consecutive frames (A, B, C) from a video sequence demonstrating the tracking of a moderately moving vehicle. The colored pixels represent tracking predictions generated by the benchmark system, based on a 30-point grid overlaid on the target object: a yellow-green car moving across the frame. The progression from left to right illustrates the system's performance in maintaining tracking of the vehicle as it travels along a road with intermittent occlusions because of the poles on the road and other vehicles present.

GT Velocity: 33.51 m/s Estimated Velocity: 6.56 m/s

(A) (B) (C)

Figure 4.12: This figure presents three non-consecutive frames (A, B, C) from a video sequence captured on a highway with multiple vehicles. The estimated average velocity of 6.56 m/s is displayed in the top-left corner of each frame, while the ground truth average velocity is 33.51 m/s, indicating a significant discrepancy between the estimated and actual speeds. The sequence shows a wide view of a multi-lane highway with various vehicles at different distances from the camera. The tracked object is the grey car coming from the left. The progression from left to right demonstrates a high-speed moving car. The large difference between the estimated and ground truth velocities underscores the difficulty in accurately measuring speed in such dynamic environments as there are no other challenging features in this specific video.

GT Velocity: 0.24m/s Estimated Velocity: 15.09 m/s



(A) (B) (C)

Figure 4.13: This figure presents three non-consecutive frames (A, B, C) from the best performing video sequence captured within the outliers in a low-light environment. The colored pixels represent tracking predictions generated by the benchmark system, based on a 30-point grid overlaid on the target object - a nearly stationary vehicle. The estimated average velocity of 15.09 m/s is displayed in the top-left corner of each frame, while the ground truth average velocity is just 0.24 m/s. This significant discrepancy highlights a major challenge in the tracking system's performance under these conditions.

Based on our analysis of the five outliers, as detailed in the figure captions of the corresponding videos, we can identify several challenging circumstances that significantly impact performance. High-speed objects, such as those moving rapidly on a highway, consistently result in underestimated velocity due to issues with COLMAP's failure, specifically with these calibrations mentioned in the methodology, to accurately estimate the ego-motion of the moving camera. In low-light environments with minimal motion, the estimated velocity tends to be overestimated, with sensitivity analysis indicating that the X and Y tracking is problematic, likely due to reduced color contrast in dark conditions. Multiple occlusions also affect velocity estimates, as objects passing through occlusions cause the tracking method to lose the original point and track another pixel, as illustrated in Figure 4.11. Camera lens issues further complicate accurate velocity estimation, as demonstrated by Figure 4.9, which shows that raindrops on the lens and a low-light environment obscure the car, making it extremely difficult to estimate depth and track points correctly. Our sensitivity analysis of these outlier videos (Appendix Table B.1 to B.6) highlights a notable improvement in velocity RMSE when different ground truth factors are applied. These differences depend on the scenarios mentioned above.

Chapter 5

Conclusion

This research aimed to explore the effectiveness of state-of-the-art 3D tracking methods in estimating real-world velocity from videos, with a particular focus on long-range tracking systems. The findings presented offer significant insights into both the performance and limitations of the tested methods.

When comparing different 3D point tracking methods for pixel velocity in image space, SpatialTracker emerged as the most effective, achieving the lowest RMSE for real world velocity estimations. This method consistently outperformed others, including those employing joint approaches, across both camera and world coordinate systems. This success is likely due to SpatialTracker's strong correlation among the coordinates in the X-axis, Y-axis and Z-axis. This helps maintain accurate scaling and proportions between these axes, as validated in Figure 4.3 and Figure 4.5. The authors of SpatialTracker already made it clear that their method works well with edge cases in terms of occlusions and inconsistencies due to their enforced 3D motion priors such as ARAP. The other factorized methods in this study do not have these priors and will need manual adjustments to achieve this. Evidently, this research revealed that regressing depth and tracks together yields better results than treating each one individually. While *DepthAnything V2* did not offer a significant improvement over *ZoeDepth* in velocity estimation, it provided comparable results, indicating that enhancements in depth estimation do not always translate to better velocity tracking under the tested conditions. However, depth estimation remains a valuable complement to tracking, as it improves overall tracking performance.

In terms of integrating these methods into benchmarks for real-world velocity estimation, the combination of *Colmap*, *SpatialTracker*, and a 30-pixel grid size proved effective. This configuration balanced accuracy with computational efficiency, making it a possible method for evaluating velocity estimation in practical scenarios, especially when camera extrinsics are accurately estimated. The results also highlighted a discrepancy: while results in camera coordinate systems were consistent, the magnitude of errors in world coordinates was higher due to the lack of consideration for extrinsics and camera movement.

The study also identified several limitations of state-of-the-art 3D tracking methods, particularly in challenging environments. Issues such as high-speed objects, low-light conditions, intermittent occlusions, and camera lens problems significantly impacted tracking performance. These issues are, however, also encountered with LiDAR and radar systems when estimating velocity, despite these methods gener-

ally being accurate. This indicates that the performance of our best performing benchmark in edge cases holds up well against other widely used systems. Furthermore, the analysis underscored the critical need for accurate extrinsic calibration, as even minor errors in camera pose estimation can lead to substantial inaccuracies in velocity tracking. This asks for careful consideration of camera pose estimation methods when engineering an optimized version of the methods. These limitations define the scope in which our methods work well.

In conclusion, this research validated the hypothesis that within current long-range 3D point tracking methods can provide reasonably accurate real-world velocity estimations, but only within the specific operational limits. *SpatialTracker* stood out as the best-performing method, yet even this approach struggled with high-speed and low-visibility scenarios, emphasizing the need for further refinement. The results suggest that while these methods are effective, their practical application requires careful consideration of environmental factors and precise calibration.

5.1 Limitations and Future Work

In future research, it would be highly beneficial to investigate the performance of our methods across a wider range of environments to enhance their applicability and robustness. The ultimate goal is to create reproducible techniques that can accurately estimate real-world velocity in diverse scenarios, such as athletes on a soccer field, aircraft in flight, or other dynamic settings. This broader evaluation will help us understand how our methods perform under different conditions and ensure their versatility in practical applications. Additionally, we are eager to delve deeper into understanding the relationships between the X, Y, and Z components within a velocity vector. Our current analysis has focused on these dimensions separately, but a more nuanced investigation into how these components interact and influence each other is crucial. By exploring the interdependencies among X, Y, and Z, we can gain insights into how their combined effects impact velocity estimation. Moreover, while this research has not concentrated on engineering and optimizing the methods, a significant next step would be to develop and refine a framework that incorporates these methods, such as SpatialTracker, for real-time velocity estimation. Building a system capable of processing data in real-time would represent a major advancement, making it possible to deploy these methods in dynamic environments and applications. Such progress would contribute to creating more robust and practical solutions for velocity estimation, an area that has often been overlooked despite its importance. By focusing on these aspects, we aim to push the boundaries and address the challenges of real-world velocity estimation more effectively.

Appendix A

Sensitivity analysis other methods

Metric	Velocity RMSE
All Estimated	20.80
Est extrinsics + GT depth + Est visibility	12.42
GT extrinsics + Est depth + Est visibility	20.40
Est extrinsics + Est depth + GT visibility	19.71
Est extrinsics + GT depth + GT visibility	12.35
GT extrinsics + GT depth + GT visibility	10.53

Table A.1: Ablation study results for ZoeDepth + CoTracker showing velocity RMSE under various combinations of estimated and ground truth inputs in world coordinates. Est = estimated, GT = ground truth.

Metric	Velocity RMSE
All Estimated	20.84
Est extrinsics + GT depth + Est visibility	12.40
GT extrinsics + Est depth + Est visibility	20.34
Est extrinsics + Est depth + GT visibility	19.30
Est extrinsics + GT depth + GT visibility	12.35
GT extrinsics + GT depth + GT visibility	10.53

Table A.2: Ablation study results for DepthAnything V2 + CoTracker showing velocity RMSE under various combinations of estimated and ground truth inputs in world coordinates. Est = estimated, GT = ground truth.

Appendix B

Extra results sensitivity analysis of outliers

Metric	ZoeDepthCoTracker	SpaTracker	DepthAnythingCoTracker
RMSE Avg Velocity	37.777577	30.309624	51.381599
RMSE Z	137.414808	138.248079	142.788766
RMSE XY	1720.510766	1720.896780	1716.545294
RMSE Avg Velocity GT Z	51.776855	58.085465	56.576200
GT Mean Velocity	0.087569	0.087569	0.087569
MV: Est extrinsics + Est Z + Est visibility	35.140113	10.194368	54.679847
MV: Est extrinsics + GT Z + Est visibility	20.426282	20.142798	20.426282
MV: GT extrinsics + Est Z + Est visibility	35.778173	14.321271	52.196053
MV: Est extrinsics + Est Z + GT visibility	32.542317	9.130451	53.512816
MV: Est extrinsics + GT Z + GT visibility	22.161477	21.914924	22.161477
MV: GT extrinsics + GT Z + GT visibility	19.683035	18.918253	19.683035

Table B.1: In-depth analysis of the worst performing video.

Metric	ZoeDepthCoTracker	SpaTracker	DepthAnythingCoTracker
RMSE Avg Velocity	24.723341	24.013606	24.641589
RMSE Z	533.861517	536.204491	534.219347
RMSE XY	10343.310614	10347.706888	10344.166045
RMSE Avg Velocity GT Z	23.725936	24.153700	23.778151
GT Mean Velocity	34.264923	34.264923	34.264923
MV: Est extrinsics + Est Z + Est visibility	8.236710	9.591139	8.361907
MV: Est extrinsics + GT Z + Est visibility	9.197919	9.273915	9.197919
MV: GT extrinsics + Est Z + Est visibility	32.355120	33.339827	32.459451
MV: Est extrinsics + Est Z + GT visibility	8.252720	9.700144	8.373514
MV: Est extrinsics + GT Z + GT visibility	9.178127	9.230764	9.178127
MV: GT extrinsics + GT Z + GT visibility	33.190747	33.394745	33.190747

Table B.2: In-depth analysis of the second worst performing video.

Metric	ZoeDepthCoTracker	SpatialTracker	DAV2CoTracker
RMSE Z	345.025243	330.875284	337.378439
RMSE XY	10022.089490	10013.982498	10025.980496
RMSE Avg Velocity Ext	41.572709	26.218688	52.989359
RMSE Avg Velocity Vis	57.865499	44.397334	63.027924
RMSE: Est extrinsics + Est Z + Est visibility	41.767453	26.537331	53.169492
RMSE: Est extrinsics + GT Z + Est visibility	24.427840	23.082112	24.427840
RMSE: GT extrinsics + Est Z + Est visibility	41.572709	26.218688	52.989359
RMSE: Est extrinsics + Est Z + GT visibility	57.865499	44.397334	63.027924
RMSE: Est extrinsics + GT Z + GT visibility	7.026333	6.164294	7.026333
RMSE: GT extrinsics + GT Z + GT visibility	5.847590	4.779707	5.847590

Table B.3: In-depth analysis of the third worst performing video.

Metric	ZoeDepthCoTracker	SpatialTracker	DAV2CoTracker
RMSE Avg Velocity	21.316226	23.153060	21.399371
RMSE Z	92.743842	89.795430	92.114850
RMSE XY	10272.094595	10279.307206	10273.859556
RMSE Avg Velocity GT Z	24.956883	24.087619	25.197942
GT Mean Velocity	29.714111	29.714111	29.714111
MV: Est extrinsics + Est Z + Est visibility	6.301976	3.485636	6.464753
MV: Est extrinsics + GT Z + Est visibility	2.589017	2.605584	2.589017
MV: GT extrinsics + Est Z + Est visibility	28.805413	27.084961	29.243870
MV: Est extrinsics + Est Z + GT visibility	6.444952	3.598760	6.395829
MV: Est extrinsics + GT Z + GT visibility	2.537117	2.553981	2.537117
MV: GT extrinsics + GT Z + GT visibility	27.186012	27.168903	27.186012

Table B.4: In-depth analysis of the fourth worst performing video.

Metric	ZoeDepthCoTracker	SpaTracker	DAV2CoTracker
RMSE Avg Velocity	20.175520	22.984069	19.727494
RMSE Z	164.772463	165.974346	164.789444
RMSE XY	8508.401780	8518.470896	8522.781396
RMSE Avg Velocity GT Z	23.695918	22.119079	23.811291
GT Mean Velocity	33.511016	33.511016	33.511016
MV: Est extrinsics + Est Z + Est visibility	11.869866	6.564014	12.095508
MV: Est extrinsics + GT Z + Est visibility	7.406743	7.384638	7.406743
MV: GT extrinsics + Est Z + Est visibility	30.015300	26.659671	30.328286
MV: Est extrinsics + Est Z + GT visibility	11.571265	6.707621	11.622072
MV: Est extrinsics + GT Z + GT visibility	7.526642	7.504465	7.526642
MV: GT extrinsics + GT Z + GT visibility	26.864417	26.723505	26.864417

Table B.5: In-depth analysis of the fifth worst performing video.

Metric	ZoeDepthCoTracker	SpaTracker	DAV2CoTracker
RMSE Avg Velocity	23.709632	20.752376	23.426042
RMSE Z	135.381514	136.602741	133.345109
RMSE XY	1912.912953	1911.336123	1917.290938
RMSE Avg Velocity GT Z	26.293428	28.231232	26.374848
GT Mean Velocity	0.242933	0.242933	0.242933
MV: Est extrinsics + Est Z + Est visibility	32.128417	15.088629	23.096644
MV: Est extrinsics + GT Z + Est visibility	23.327268	23.310061	23.327268
MV: GT extrinsics + Est Z + Est visibility	27.755133	8.584254	19.584270
MV: Est extrinsics + Est Z + GT visibility	35.618179	16.191214	26.274305
MV: Est extrinsics + GT Z + GT visibility	20.538181	20.789408	20.538181
MV: GT extrinsics + GT Z + GT visibility	11.120069	11.070775	11.120069

Table B.6: In-depth analysis of the sixth worst performing video.

Bibliography

[1] R. Arnheim. *The Journal of Aesthetics and Art Criticism*, 11(2):172–173, 1952.

[2] Y. Bai, L. Wong, and T. Twan. Survey on fundamental deep learning 3d reconstruction techniques, 2024.

[3] A. Balasingam, J. Chandler, C. Li, Z. Zhang, and H. Balakrishnan. Drivetrack: A benchmark for long-range point tracking in real-world videos, 2023.

[4] S. F. Bhat, I. Alhashim, and P. Wonka. Localbins: Improving depth estimation by learning local distributions, 2022.

[5] S. F. Bhat, R. Birkl, D. Wofk, P. Wonka, and M. Müller. Zoedepth: Zero-shot transfer by combining relative and metric depth, 2023.

[6] R. Birkl, D. Wofk, and M. Müller. Midas v3.1 – a model zoo for robust monocular relative depth estimation, 2023.

[7] Y. Cabon, N. Murray, and M. Humenberger. Virtual kitti 2, 2020.

[8] Y. Cabon, N. Murray, and M. Humenberger. Virtual KITTI 2. *CoRR*, abs/2001.10773, 2020.

[9] E. R. Chan, C. Z. Lin, M. A. Chan, K. Nagano, B. Pan, S. D. Mello, O. Gallo, L. Guibas, J. Tremblay, S. Khamis, T. Karras, and G. Wetzstein. Efficient geometry-aware 3d generative adversarial networks, 2022.

[10] W. Changchang. Visualsfm: A visual structure from motion system, 2011.

[11] J. Cho, D. Min, Y. Kim, and K. Sohn. Diml/cvl rgb-d dataset: 2m rgb-d images of natural indoor and outdoor scenes, 2021.

[12] C. Doersch, A. Gupta, L. Markeeva, A. Recasens, L. Smaira, Y. Aytar, J. Carreira, A. Zisserman, and Y. Yang. Tap-vid: A benchmark for tracking any point in a video, 2023.

[13] C. Doersch, Y. Yang, M. Vecerik, D. Gokay, A. Gupta, Y. Aytar, J. Carreira, and A. Zisserman. Tapir: Tracking any point with per-frame initialization and temporal refinement, 2023.

[14] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network, 2014.

[15] J. Engel, T. Schöps, and D. Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *European Conference on Computer Vision (ECCV)*, September 2014.

[16] D. Fernández Llorca, A. Hernández Martínez, and I. Garcia Daza. Vision-based vehicle speed estimation: A survey. *IET Intelligent Transport Systems*, 15(8):987–1005, 2021.

[17] D. Fernández-Llorca, A. Hernandez Martinez, and I. garcia daza. Vision-based vehicle speed estimation: A survey. *IET Intelligent Transport Systems*, 15, 05 2021.

[18] D. Fernández Llorca, A. Hernández Martínez, and I. García Daza. Vision-based vehicle speed estimation: A survey. *IET Intelligent Transport Systems*, 15(8):987–1005, May 2021.

[19] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, jun 1981.

[20] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, 2012.

[21] I. Ghosh, S. Ramasamy Ramamurthy, A. Chakma, and N. Roy. Sports analytics review: Artificial intelligence applications, emerging technologies, and algorithmic perspective. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 13(5):e1496, 2023.

[22] C. Godard, O. M. Aodha, and G. J. Brostow. Unsupervised monocular depth estimation with left-right consistency, 2017.

[23] L. Haas, A. Haider, L. Kastner, T. Zeh, T. Poguntke, M. Kuba, M. Schardt, M. Jakobi, and A. W. Koch. Velocity estimation from lidar sensors motion distortion effect. *Sensors*, 23(23), 2023.

[24] J. Harikrishnan, A. Sudarsan, A. Sadashiv, and R. A. Ajai. Vision-face recognition attendance monitoring system for surveillance using deep learning technology and computer vision. In *2019 international conference on vision towards emerging trends in communication and networking (ViTECoN)*, pages 1–5. IEEE, 2019.

[25] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[26] J. Hayakawa and B. Dariush. Ego-motion and surrounding vehicle state estimation using a monocular camera. In *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, June 2019.

[27] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks, 2016.

[28] D. K. Jain, R. Jain, L. Cai, M. Gupta, and Y. Upadhyay. Relative vehicle velocity estimation using monocular video stream. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2020.

[29] M. Kampelmühler, M. Müller, and C. Feichtenhofer. Camera-based vehicle velocity estimation from monocular video. 02 2018.

[30] N. Karaev, I. Rocco, B. Graham, N. Neverova, A. Vedaldi, and C. Rupprecht. Cotracker: It is better to track together, 2023.

[31] B. Ke, A. Obukhov, S. Huang, N. Metzger, R. C. Daudt, and K. Schindler. Repurposing diffusion-based image generators for monocular depth estimation, 2024.

[32] S. Koppula, I. Rocco, Y. Yang, J. Heyward, J. Carreira, A. Zisserman, G. Brostow, and C. Doersch. Tapvid-3d: A benchmark for tracking any point in 3d, 2024.

[33] Y. Li and J. Ibanez-Guzman. Lidar for autonomous driving: The principles, challenges, and trends for automotive lidar and perception systems. *IEEE Signal Processing Magazine*, 37(4):50–61, July 2020.

[34] Z. Li, S. F. Bhat, and P. Wonka. Patchfusion: An end-to-end tile-based framework for high-resolution monocular metric depth estimation, 2023.

[35] Z. Li and N. Snavely. Megadepth: Learning single-view depth prediction from internet photos. In *Computer Vision and Pattern Recognition (CVPR)*, 2018.

[36] L. Liebe, F. Sauerwald, S. Sawicki, M. Schneider, L. Schuhmann, T. Buz, P. Boes, A. Ahmadov, and G. de Melo. Farsec: A reproducible framework for automatic real-time vehicle speed estimation using traffic cameras, 2023.

[37] Y. Liu, J. Jiang, J. Sun, L. Bai, and Q. Wang. A survey of depth estimation based on computer vision. In *2020 IEEE Fifth International Conference on Data Science in Cyberspace (DSC)*, pages 135–141, 2020.

[38] D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.

[39] V. K. Madasu and M. Hanmandlu. Estimation of vehicle speed by motion tracking on image sequences. In *2010 IEEE Intelligent Vehicles Symposium*, pages 185–190, 2010.

[40] J. McCormac, A. Handa, S. Leutenegger, and A. J. Davison. Scenenet rgb-d: 5m photorealistic images of synthetic indoor trajectories with ground truth, 2017.

[41] R. McCraith, L. Neumann, and A. Vedaldi. Real time monocular vehicle velocity estimation using synthetic data, 2021.

[42] H. H. Meinel and W. Bösch. *Radar Sensors in Cars*, pages 245–261. Springer International Publishing, Cham, 2017.

[43] M. J. M. M. Mur-Artal, Raúl and J. D. Tardós. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[44] P. K. Nathan Silberman, Derek Hoiem and R. Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

[45] P. K. Nathan Silberman, Derek Hoiem and R. Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

[46] M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, M. Assran, N. Ballas, W. Galuba, R. Howes, P.-Y. Huang, S.-W. Li, I. Misra, M. Rabbat, V. Sharma, G. Synnaeve, H. Xu, H. Jegou, J. Mairal, P. Labatut, A. Joulin, and P. Bojanowski. Dinov2: Learning robust visual features without supervision, 2024.

[47] X. Pan, N. Charron, Y. Yang, S. Peters, T. Whelan, C. Kong, O. Parkhi, R. Newcombe, and C. Y. Ren. Aria digital twin: A new benchmark dataset for egocentric 3d machine perception, 2023.

[48] D. Parekh, N. Poddar, A. Rajpurkar, M. Chahal, N. Kumar, G. P. Joshi, and W. Cho. A review on autonomous vehicles: Progress, methods and challenges. *Electronics*, 11(14), 2022.

[49] U. Rajapaksha, F. Sohel, H. LAGA, D. Diepeveen, and M. Bennamoun. Deep learning-based depth estimation methods from monocular image and videos: A comprehensive survey. *ACM Computing Surveys*, July 2024.

[50] R. Ranftl, K. Lasinger, D. Hafner, K. Schindler, and V. Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer, 2020.

[51] T. Ridnik, E. Ben-Baruch, A. Noy, and L. Zelnik-Manor. Imagenet-21k pretraining for the masses, 2021.

[52] M. Roberts, J. Ramapuram, A. Ranjan, A. Kumar, M. A. Bautista, N. Paczan, R. Webb, and J. M. Susskind. Hypersim: A photorealistic synthetic dataset for holistic indoor scene understanding. In *International Conference on Computer Vision (ICCV) 2021*, 2021.

[53] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3234–3243, 2016.

[54] A. M. Roy, J. Bhaduri, T. Kumar, and K. Raj. Wildect-yolo: An efficient and robust computer vision-based accurate object localization model for automated endangered wildlife detection. *Ecological Informatics*, 75:101919, 2023.

[55] G. Salvo, L. Caruso, A. Scordo, G. Guido, and A. Vitale. Comparison between vehicle speed profiles acquired by differential gps and uav. 07 2014.

[56] A. Saxena, M. Sun, and A. Y. Ng. Make3d: Learning 3d scene structure from a single still image. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5):824–840, 2009.

[57] J. L. Schönberger and J.-M. Frahm. Structure-from-Motion Revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[58] G. Seguin, K. Alahari, J. Sivic, and I. Laptev. Pose estimation and segmentation of people in 3d movies. volume 37, pages 1643–1655, 2015.

[59] N. Snavely, S. M. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3d. In *ACM siggraph 2006 papers*, pages 835–846. 2006.

[60] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, V. Vasudevan, W. Han, J. Ngiam, H. Zhao, A. Timofeev, S. Ettinger, M. Krivokon, A. Gao, A. Joshi, S. Zhao, S. Cheng, Y. Zhang, J. Shlens, Z. Chen, and D. Anguelov. Scalability in perception for autonomous driving: Waymo open dataset, 2020.

[61] S. Vedula, S. Baker, P. Rander, R. Collins, and T. Kanade. Three-dimensional scene flow. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 722–729 vol.2, 1999.

[62] P. Wang, X. Huang, X. Cheng, D. Zhou, Q. Geng, and R. Yang. The apolloscape open dataset for autonomous driving and its application. *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[63] Q. Wang, S. Zheng, Q. Yan, F. Deng, K. Zhao, and X. Chu. Irs: A large naturalistic indoor robotics stereo dataset to train deep models for disparity and surface normal estimation, 2021.

[64] W. Wang, D. Zhu, X. Wang, Y. Hu, Y. Qiu, C. Wang, Y. Hu, A. Kapoor, and S. Scherer. Tartanair: A dataset to push the limits of visual slam. 2020.

[65] M. Westoby, J. Brasington, N. Glasser, M. Hambrey, and J. Reynolds. 'structure-from-motion' photogrammetry: A low-cost, effective tool for geoscience applications. *Geomorphology*, 179:300–314, 2012.

[66] T. Weyand, A. Araujo, B. Cao, and J. Sim. Google Landmarks Dataset v2 - A Large-Scale Benchmark for Instance-Level Recognition and Retrieval. In *Proc. CVPR*, 2020.

[67] K. Xian, C. Shen, Z. Cao, H. Lu, Y. Xiao, R. Li, and Z. Luo. Monocular relative depth perception with web stereo data supervision. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 311–320, 2018.

[68] K. Xian, J. Zhang, O. Wang, L. Mai, Z. Lin, and Z. Cao. Structure-guided ranking loss for single image depth prediction. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[69] Y. Xiao, Q. Wang, S. Zhang, N. Xue, S. Peng, Y. Shen, and X. Zhou. Spatialtracker: Tracking any 2d pixels in 3d space, 2024.

[70] C. Xu, J. Bao, C. Wang, and P. Liu. Association rule analysis of factors contributing to extraordinarily severe traffic crashes in china. *Journal of Safety Research*, 67:65–75, 2018.

[71] M. Xu, Y. Wang, B. Xu, J. Zhang, J. Ren, S. Poslad, and P. Xu. A critical analysis of image-based camera pose estimation techniques, 2022.

[72] Y. Yan, S. Yancong, and M. Zengqiang. Research on vehicle speed measurement by video image based on tsai's two stage method. In *2010 5th International Conference on Computer Science Education*, pages 502–506, 2010.

[73] L. Yang, B. Kang, Z. Huang, X. Xu, J. Feng, and H. Zhao. Depth anything: Unleashing the power of large-scale unlabeled data. In *CVPR*, 2024.

[74] L. Yang, B. Kang, Z. Huang, Z. Zhao, X. Xu, J. Feng, and H. Zhao. Depth anything v2. *arXiv:2406.09414*, 2024.

[75] Y. Yao, Z. Luo, S. Li, J. Zhang, Y. Ren, L. Zhou, T. Fang, and L. Quan. Blendedmvs: A large-scale dataset for generalized multi-view stereo networks. *Computer Vision and Pattern Recognition (CVPR)*, 2020.

[76] F. Yu, H. Chen, X. Wang, W. Xian, Y. Chen, F. Liu, V. Madhavan, and T. Darrell. Bdd100k: A diverse driving dataset for heterogeneous multitask learning, 2020.

[77] F. Yu, Y. Zhang, S. Song, A. Seff, and J. Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.

[78] C. Zheng, W. Wu, C. Chen, T. Yang, S. Zhu, J. Shen, N. Kehtarnavaz, and M. Shah. Deep learning-based human pose estimation: A survey, 2023.

[79] Y. Zheng, A. W. Harley, B. Shen, G. Wetzstein, and L. J. Guibas. Pointodyssey: A large-scale synthetic dataset for long-term point tracking, 2023.