

Thesis

BikeScenes-planning: Towards end-to-end bicycle path planning

Aleksander Seremak



Thesis

BikeScenes-planning: Towards end-to-end bicycle path planning

by

Aleksander Seremak

to obtain the degree of Master of Science in Robotics
at the Delft University of Technology,
to be defended publicly on Monday, June 22, 2026.

Thesis committee:	Dr. Holger Caesar	TU Delft, Supervisor
	Dr. Micah Prendergast	TU Delft
	Dr. Raj Thilak Rajan	TU Delft

BikeScenes-planning: Towards end-to-end bicycle path planning

Aleksander Seremak Holger Caesar

Abstract—While automated driving for cars has seen significant advancements, the automation of two-wheeled vehicles remains largely underexplored. This gap is primarily due to a lack of appropriate multi-modal datasets and methods that incorporate bicycle roll, yaw, and forward velocity. Consequently, existing end-to-end (E2E) path planners designed for automobiles cannot be directly applied to the bicycle domain. In this paper we address these limitations by introducing the first comprehensive E2E path planning framework tailored specifically for bicycles. First, we present *BikeScenes-planning*, a novel multimodal dataset captured using the SenseBike platform, comprising 360° RGB images, precise global 2D position, and explicit roll measurements. Second, we demonstrate the necessity of departing from the Average Displacement Error (ADE) metric for bicycle path planner evaluation, and propose a novel, decoupled metric that prioritizes minimizing lateral offset to account for the narrow drivable area in bicycle lanes and the predominant requirement to follow the expert trajectory while allowing for speed deviation. Finally, we establish a baseline for E2E planning that explicitly incorporates roll dynamics into its predictions while using the memory-preserving architectures. Through this work, we pave the way for E2E bicycle path planners, which have the potential to enable fully autonomous two-wheeled vehicles and indirectly improve autonomous driving safety assessments by modeling bicycles as dynamic, reactive agents in simulation evaluations.

I. INTRODUCTION

Over the past decades, automated driving has evolved from a conceptual, futuristic vision into a transformative technology, with cars being the most researched domain of vehicles. Early successful attempts to introduce driving automation date back to the 1980s [1]. Automobiles, however, are not the sole means of urban transportation – especially in Europe, where two-wheeled vehicles such as bicycles are highly efficient for short-distance travel and navigating dense city centers [2]. Despite their widespread use, there is a significant research gap regarding the automation of two-wheeled vehicles. This deficit is largely due to their complex balancing dynamics and the operational environment with highly stochastic moving agents.

Recently, a new approach to path planning emerged in the autonomous driving (AD) industry - end-to-end path planning (E2E). This paradigm diverges from the conventional modular approach, which treats each module as a standalone component, producing an independent output. A typical modular pipeline consists of perception, prediction, and planning [3]. The modular design offers interpretability, simplifies debugging, and enables parallel development across teams, thus improving project management. However, because each module is optimized for different objectives – for instance, a detection module tends to maximize mean average precision (mAP)

while planning prioritizes safety – the entire system may not align with the final goal. Moreover, sequential execution of these modules, each producing an isolated result, can lead to information loss and error accumulation.

Conversely, the E2E paradigm addresses these limitations by employing a single, fully differentiable pipeline. While E2E models offer a data-driven solution mapping raw sensor inputs to trajectories or control signals, they do not strictly require a unified “black-box” architecture (e.g., PilotNet [4]). In fact, modern E2E models such as UniAD [5] employ a modular structure by utilizing interconnected sub-networks that can generate intermediate representations. Therefore, the defining requirement for the E2E paradigm is not the absence of modules, but rather the necessity for backpropagation through the entire network. Because information flows between these modules at the feature level, the entire pipeline can be jointly optimized, effectively mitigating the information loss and error accumulation in contrast to the classical approach.

To successfully train these E2E planners, a vast amount of high-quality, multi-modal data is required. Current automotive state-of-the-art models rely heavily on 360° camera images, often supplemented by LiDAR data [6, 7]. While extensive datasets and sophisticated E2E networks exist for cars, the bicycle domain completely lacks the data necessary to implement this fully differentiable planning paradigm. The development of the E2E bicycle planners can be crucial for three key reasons. Such planners can be used to develop fully autonomous two-wheeled vehicles. Additionally, E2E planners have the potential to accelerate the development of advanced balancing safety systems for bicycles in which neural networks implicitly model complex dynamics. Moreover, E2E bicycle planners may enable highly realistic, reactive bicycle agents for closed-loop autonomous driving simulations, thereby enhancing overall AD safety.

Therefore, we contribute the following to the research community:

- We utilize the SenseBike bicycle equipped with a comprehensive sensor suite to collect the first-ever multi-modal 360° coverage dataset - *BikeScenes-planning* - at a scale comparable to automotive datasets, which we release upon publication. The *BikeScenes-planning* enables research on multiple tasks, such as behavior modeling, object detection, and tracking, across various environmental conditions in a previously underexplored bicycle domain.
- We introduce a new evaluation metric for bicycle end-to-end path planners, tailored to the characteristics of bicycle velocity profiles and dynamics.

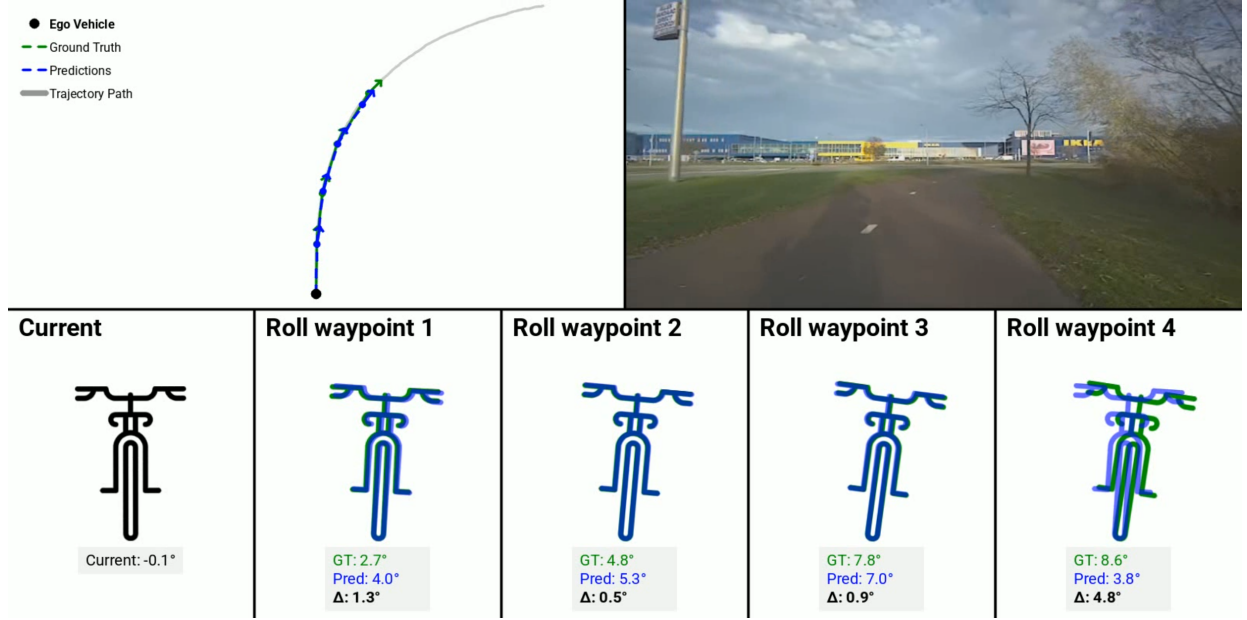


Fig. 1: Visualization of the E2E planner correctly learning the entanglement of roll while turning in bicycle dynamics. The predictions are marked in **blue**, whereas the ground truth data is in **green**. Each waypoint was predicted at 1-second intervals. Top-left: predicted vs ground truth waypoints in ego-centric frame with heading predictions (arrows); top-right: front camera image (left lens), bottom: predicted vs ground truth roll.

- We establish a baseline for end-to-end bicycle path planners by proposing a novel approach that leverages RGB / RGB-D images as semantic information and explicitly incorporates roll dynamics into the planning process.

II. RELATED WORK

The two-wheeled vehicle domain remains vastly under-explored, unlike extensively researched autonomous driving applications for cars. One possible reason is the lack of research data. Currently, the number of multimodal datasets collected using bicycles is very limited, with most existing datasets collected for a specific task. Therefore, there is little research on machine learning for tasks in the bicycle domain, such as E2E path planning.

A. Datasets

1) *Automotive datasets*: Multi-modal datasets have directly accelerated autonomous driving research, evolving from simple monocular datasets to extensive sensor rigs incorporating 360° coverage camera images, LiDAR, GNSS, and IMU sensors. The pioneering KITTI dataset [8] was the first widely available dataset providing LiDAR point cloud in addition to front-facing stereo camera images and GNSS/IMU data. It was released in 2012 and contains 22 scenes totaling 1.5 hours.

nuScenes [9] introduced a new industry standard by releasing a multi-modal dataset comprising 6 cameras covering the full surrounding of the ego vehicle, LiDAR, 5 radars, and GNSS/IMU. The dataset is built from 1000 scenes, each 20 seconds long, which corresponds to approximately 5.5 hours of data. With the release of the nuScenes dataset [9], the scientific community was able to dive into autonomous

driving research and explore machine learning-based tasks, including end-to-end path planning. Datasets such as Waymo Open Dataset [10] (5.5 hours), and Lyft L5 [11] (2.5 hours) further standardized the inclusion of RGB images, LiDAR point clouds, and GNSS/IMU data. The first planning-oriented dataset, nuPlan [12], comprises 1500 hours of driving and its closed-loop capabilities, combined with this significant amount of data, enable a more robust evaluation of planners.

The automotive datasets laid the groundwork for further developments in the autonomous driving domain. Although they supply the research community with comprehensive data, the lack of roll and a significant domain gap make them unsuitable for bicycle-specific setting.

2) *Bicycle datasets*: Only a handful of datasets collected using bicycles exist. Unlike the automotive datasets, which can serve multiple purposes, most of the bicycle datasets have been collected for a very specific task. The Salzburg Bicycle 3d (SaBi3d) dataset [13] was captured solely to analyze the car-to-bicycle overtaking maneuvers. This dataset contains only LiDAR point clouds from around 30 minutes of recordings and lacks camera data. The BikeActions dataset [14] was created to assess the Vulnerable Road Users (VRU) behavior modeling, i.e., pedestrians and cyclists. The LiDAR sensors mounted at the front and back, along the front-facing camera and GNSS-RTK receiver, were used for data collection. This sensor configuration is not suitable for accurate 360° data capture. Moreover, the BikeActions dataset is built from 12 sequences, resulting in 1.2 hours of recorded data. The BikeScenes-lidarseg dataset [15] was collected purely for the segmentation task and contains the RGB image from the front-facing camera and a 360° coverage LiDAR point cloud. However, the BikeScenes-lidarseg dataset contains data from

a 1.3-kilometer loop, which results in a dataset less than 6 minutes long.

Consequently, the scientific community currently lacks a 360° coverage, multi-modal dataset of sufficient scale that incorporates roll dynamics to train E2E planners.

B. End-to-end path planning

End-to-end planner architectures have evolved significantly in recent years. PilotNet [4] proved that an Imitation Learning paradigm, especially Behavioral Cloning, can directly drive a car using a 9-layer CNN with just 72 hours of training data. It directly maps the front image to kinematic driving commands. Subsequent methods abstracted the driving task, predicting a valid set of waypoints or discrete trajectories rather than raw control signals as described in Table II.

To learn correct regression of waypoints/trajectories, the model should be penalized with the appropriate loss function. Most behavioral cloning (BC) based models rely on direct spatial regression to predict future waypoints/trajectories, split primarily between two mathematical approaches. Methods such as PDM-Open [16], VAD [17], VAD-v2 [18], InterFuser [19], or ReasonNet [20] use Mean Absolute Error (MAE), which scales the penalty linearly with the error margin. In contrast, models like PilotNet [4], Hydra-MDP [6], and UniAD [5] employ Mean Squared Error (MSE) that penalizes the error quadratically. Iterative models such as iPad [23] use a Minimum over N (MoN) loss, whereas flow-based models like GuideFlow [22] utilize a Rectified Flow Matching (RFM) objective and an Energy-Based Model (EBM) refinement loss.

The spatial input plays an important role in E2E path planning as it provides the necessary data to understand the ego vehicle's surroundings. The vast majority of models require RGB multi-view images, which are generated by concatenating the RGB images from cameras mounted on the vehicle, mostly covering a 360° horizontal FOV. Some approaches, like Hydra-MDP [6], InterFuser [19], ReasonNet [20], or TransFuser [7], utilize the LiDAR point cloud in addition to camera data. To process the spatial input, most approaches incorporate backbone architectures, among which ResNet [24] variations are predominantly used, as shown in Table I. Additionally, to incorporate the ego dynamics in the predictions, models also employ the status of the ego-vehicle, comprising of i.e velocity, acceleration, and turning rate. To perform prediction, the models require a goal which is supplied mostly as a high-level navigation command [5, 7, 17, 21] or a navigation goal in the form of a 2D coordinate vector. Some models can accept both modalities depending on the evaluation benchmark [6, 18, 19, 20, 23]. Finally, most of the models predict trajectories [6, 17, 18, 19, 21] or a set of 2D waypoints [5, 7, 16, 20].

The lack of an appropriate dataset made exploring the E2E planning paradigm in the bicycle domain rather difficult. Therefore, our contributions are the first of their kind in the bicycle domain and tackle dataset curation and further benchmarking the E2E planner for bicycles.

III. BIKESCENES-PLANNING DATASET

To tackle the identified research gaps in Section II we present a *BikeScenes-planning* dataset. The dataset has been collected within the vicinity of Delft in the Netherlands on November 12th and 13th, 2025. We captured the data from bicycle paths and in mixed traffic, where bicycles share a road with cars. The detailed routes are shown in Figure 2. To create *BikeScenes-planning*, a multi-modal data acquisition platform was used – SenseBike, shown in Figure 3. The collected data originates from three LiDAR sensors, four stereo cameras (incorporating IMU measurements), and a Global Navigation Satellite System (GNSS) utilizing Real-Time Kinematics (RTK) corrections. The comprehensive data capture procedure is detailed in Appendix C. We then follow a nuScenes [9] convention and curate a dataset of 20-second-long scenes. In total, the finalized *BikeScenes-planning* dataset aggregates the two capture days into 889 scenes (Table X), culminating in approximately 5 hours of multimodal operational data.

A. SenseBike

SenseBike is a data acquisition platform of the TU Delft Intelligent Vehicle group (Figure 3). It is equipped with multi-modal sensors that are governed by a Single Board Computer (SBC) – NVIDIA Jetson Nano NX 16 GB. The LiDAR sensors' configuration provides a comprehensive 360° coverage. The front-facing solid-state LiDAR (RoboSense M1P) captures the dense point cloud at a far distance, whereas two RoboSense Helios 32 LiDARs ensure a proper data capture from both sides due to a 70° horizontal field-of-view. The synchronized LiDAR data is collected at 10 Hz.

To capture the image data, SenseBike is equipped with a comprehensive stereo camera setup that provides RGB images along with a point cloud and depth images. Four ZED X cameras were used to ensure a 360° spatial coverage and collect data at 15 Hz. As shown in Figure 10, three ZED X cameras mounted at the front rack and one at the rear LiDAR support beam provide a synchronized multi-view stereo vision feed. The camera placement has been carefully selected to prioritize field-of-view overlap at the front of the bicycle (Figure 11) by orienting the side cameras at a +/- 92° offset relative to the longitudinal axis of motion. Frame synchronization is guaranteed through the use of a Capture Card – a PCB designed to trigger and hardware synchronize the camera feed. We use a camera setup to estimate the SenseBike's pose using Visual-Inertial Odometry (VIO), which combines IMU readings from the cameras and stereo vision. A detailed description of the camera setup is provided in Appendix B.

Accurate global position estimation is achieved by utilizing a GNSS receiver capable of processing Real-Time Kinematic (RTK) corrections. The GNSS data is processed at 10 Hz with a 1 Hz RTK corrections frequency, which we receive through NTRIP Caster. The base station DLF100NLD, located near the TU Delft campus, is selected as the reference point for Real-Time Kinematics corrections.

Method	Sensor Modalities	Backbone [†]	Learning Paradigm	Auxiliary tasks	Vocab	Benchmark
Pilot Net [4]	Camera	CNN	Behavioral Cloning	-	-	Own
Hydra-MDP [6]	Lidar & Camera	ResNet34 (L) & ResNet34 (C)	Multi-target Knowledge Distillation	✓	✓	NAVSIM
PDM-Open [16]	-	-	Behavioral Cloning	-	-	nuPlan
UniAD [5]	Camera	ResNet50 / ResNet101 / V2-99	Behavioral Cloning	✓	-	nuScenes
TransFuser [7]	Lidar & Camera	ResNet34 (L) & ResNet34 (C)	Behavioral Cloning	-	-	CARLA Leaderboard
VAD [17]	Camera	ResNet50	Behavioral Cloning	✓	-	nuScenes & Town05
VAD-v2 [18]	Camera	ResNet50	Probabilistic Behavioral Cloning	✓	✓	Town05
InterFuser [19]	Lidar & Camera	ResNet34 (L) & ResNet34 (C)	Behavioral Cloning	✓	-	Town05
ReasonNet [20]	Lidar & Camera	PointPillars (L) & ResNet50 (C)	Behavioral Cloning	✓	-	Town05 & DOS
SparseDrive [21]	Camera	ResNet50 / ResNet101	Behavioral Cloning	✓	-	nuScenes
GuideFlow [22]	Camera	ResNet34 / ResNet50 / V2-99	Generative (EBM and Flow matching)	✓	✓	NAVSIM & Bench2Drive
iPad [23]	Camera	ResNet34	Generative proposal-based	✓	-	NAVSIM & Bench2Drive

TABLE I: Sensor modalities, perception backbones, learning paradigm, and benchmark comparison for selected end-to-end path planning methods. [†] C denotes a camera, whereas L denotes a LiDAR sensor, and Vocab denotes the subset of trajectories (Vocabulary of trajectories).

Method	Input [†]	Ego-status	Goal Modality [†]	Planner Output	Planner head	WP/Traj. loss	Architecture
Pilot Net [4]	YUV Image	-	N/A	Path curvature (1/r)	Linear Layer	MSE	CNN
Hydra-MDP [6]	RGB MV & PC	✓	Cmd / nav goal	Trajectory	Multi-head MLP	MSE	Transformer
PDM-Open [16]	Centerline	✓	Array of 2D goals	Waypoints	MLP	MAE	MLP + GRU
UniAD [5]	RGB MV	✓	Cmd	Waypoints	MLP & BEV interaction	MSE	Transformer
TransFuser [7]	RGB MV & BEV image	✓	Cmd	Waypoints	MLP + GRU	MAE	Transformer
VAD [17]	RGB MV	✓	Cmd	Trajectory	MLP	MAE	Transformer
VAD-v2 [18]	RGB MV	✓	Cmd / nav goal	Trajectory	Prob. Traj. Decoder	MAE	Transformer
InterFuser [19]	RGB MV & PC	✓	Cmd / nav goal	Trajectory	GRU	MAE	Transformer
ReasonNet [20]	RGB MV & PC	✓	Cmd / nav goal	Waypoints	MLP + GRU	MAE	Transformer
SparseDrive [21]	RGB MV	✓	Cmd	Trajectory	Decoder	MAE + Focal	Transformer
GuideFlow [22]	RGB MV	✓	Planning anchors	Multi-modal trajectories	MLP	LRF & LRFE	Flow matching
iPad [23]	RGB MV	✓	Cmd / nav goal	Trajectory	MLP	MoN	Transformer

TABLE II: Planner input, goal modality, output, planning head, waypoint/trajectory loss, and architecture comparison for selected end-to-end path planning methods. [†] RGB MV denotes RGB multi-view, whereas PC denotes Point cloud. [‡] Cmd denotes a high-level command (e.g., turn left, drive straight), whereas nav goal denotes the navigation goal supplied as a set of coordinates.

B. Dataset curation

The initial data collection yielded over 20 hours of raw recordings, subsequently curated into approximately 5 hours of 20-second scenes. During this curation process, hardware limitations, such as sensor malfunctions that caused dropped frames, were addressed, and the dataset was further filtered to safeguard data integrity by discarding flawed sequences.

Using the entire SenseBike sensor suite during capture nearly exhausts the Single Board Computer (SBC) ’s computational capabilities, occasionally resulting in dropped camera frames. To mitigate this, we develop and implement a ”relaxed fill” algorithm that identifies dropped frames across all camera streams and replaces them with the closest valid frame, provided it falls within a 200-ms tolerance window. The relaxed filled frames are explicitly flagged in the final dataset, allowing for their exclusion during the training process. A detailed discussion is provided in Appendix D.

Accurate global pose estimation is critical for a planning-oriented dataset. Therefore, we impose a strict GNSS uncertainty threshold of 22 cm. Samples exceeding this threshold are discarded to prevent injecting the noisy samples into the subsequent fusion stage, where Visual-Inertial Odometry (VIO) is fused with the GNSS data through Factor Graph Optimization (FGO). Further discussion on GNSS pre-processing, VIO, and FGO is available in Appendix D-I.

In addition to individual frame drops, the recordings are analyzed for critical system failures, such as simultaneous recording failures across all cameras. We treat these multi-camera dropouts as strict sequence discontinuities. The result-

ing gaps in recordings act as hard boundaries during the scene-splitting workflow (described in Appendix D-G) to ensure data integrity within each extracted scene.

Finally, roll measurements are highly susceptible to bias caused by improper camera initialization or severe road vibrations. We address this issue by implementing a post-processing step to correct for the roll bias. To prevent vehicle turning dynamics from improperly influencing the bias estimation, we isolate raw roll samples recorded while driving straight, effectively excluding stationary and turning samples. Subsequently, linear interpolation is performed across the turns and stationary periods to construct a continuous bias. We further mitigate road vibrations by applying a 90-second rolling window mean to the continuous bias values. Subtracting this mean from the raw roll samples yields the corrected roll, which is subsequently used in the global pose and scene fusion.

C. Statistics

The dataset spans 71 kilometers of bicycle infrastructure throughout Delft, with an average forward velocity of 3.91 m/s (14 km/h) across all scenes. To evaluate route complexity, we performed binary turn detection for each 20-second segment. Within the dataset, scenes containing exclusively left turns account for 9.79%, exclusively right turns for 8.66%, and complex scenes featuring both left and right maneuvers represent 7.76%. The remaining majority of the data (73.79%) does not contain any turns. This structural imbalance is expected, given that cycling infrastructure is predominantly built along straight roads. To mitigate this issue, riders have been instructed

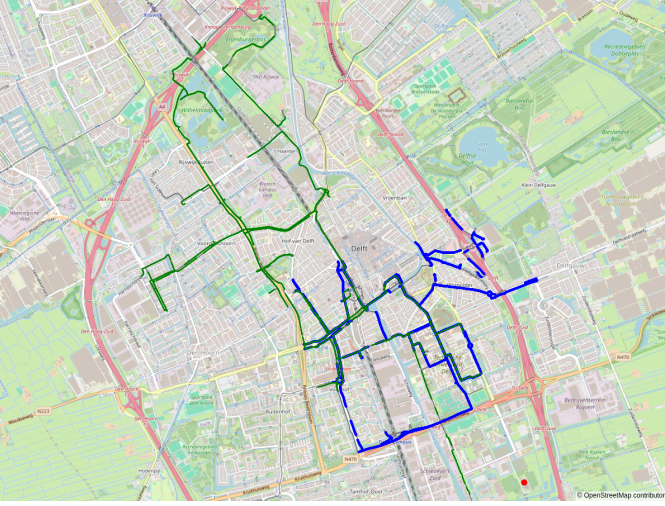


Fig. 2: Scenes visualization on the map. Scenes from November 12th are marked in **blue**, whereas scenes from November 13th are marked in **green**. The discarded pieces of the recordings are highlighted in **gray**. The GNSS RTK base station is shown in **red**.



Fig. 3: SenseBike - a data acquisition platform of TU Delft.

to maximize turning maneuvers during data collection. A detailed, per-day comparison is shown in Table XI.

D. Validation set

To match the split proportions established by nuScenes [9], we prepared a validation set consisting of 128 scenes. To strictly prevent data leakage, this split was constructed on a per-recording basis rather than a per-scene basis, effectively isolating entire 5-to-6-minute-long sequences from the training data. We ensure a similar distribution of turning events between the validation and training sets by using linear programming and subsequent manual inspection of scenes. Additionally, the linear programming was constrained to select an equal number of scenes from each recording day to prevent bias towards single weather conditions in the evaluation.

The output from this optimization step was treated as a baseline candidate and further manually inspected. Any recording flagged for insufficient route complexity or excessive geographic repetitiveness (e.g., identical loops through the TU Delft campus) was excluded from the optimization pool. The assignment algorithm was then iteratively rerun until all selected sequences met our quality standards. A detailed discussion of this methodology is available in Appendix F.

Furthermore, we established a dedicated evaluation subset – a hard validation set – comprising the most challenging scenarios manually extracted from the broader validation set. We particularly target trajectories with sharp turns or intersections, as well as routes through the TU Delft Campus during rush hours. A comprehensive breakdown of the hard validation subset is provided in Appendix F-C.

IV. TOWARDS E2E PATH PLANNERS FOR BICYCLES

E2E path planners designed for cars typically disregard roll predictions, since it is negligible in standard automotive dynamics. Conversely, bicycles exhibit coupled turning behavior (cornering), relying on both steering angle, inward lean, and forward velocity. Consequently, automotive planners cannot be directly transferred to two-wheeled vehicles, which require explicit accounting for roll in the predictions.

To address this limitation, we propose a novel E2E architecture tailored for bicycle path planning that directly integrates roll in its predictions. To benchmark our approach, we utilize the novel BikeScenes-planning dataset, which provides synchronized RGB and RGB-D images, global poses, ego-state vectors, and explicit roll measurements.

Following the Imitation Learning (IL) paradigm, specifically Behavioral Cloning (BC), we train the network using expert demonstrations as ground truth. We aim to learn a policy π that mimics the human rider behavior π^* . Our policy maps the inputs to the 7-dimensional future waypoints.

A. Planning problem

Bicycle dynamics differ fundamentally from those of automobiles. While cars rely predominantly on steering angles for lateral control, bicycles navigate by coupling steering, roll and forward velocity, actively leaning into turns. Therefore, applying standard automotive E2E models to bicycle path planning may yield inaccurate and potentially unsafe predictions.

To address this, we formulate planning as a spatio-temporal problem. Diverging from standard E2E autonomous driving planners, our models predict a sequence of 7-dimensional waypoint vectors. Each vector encodes the 2D position and yaw in the ego-centric coordinate frame, alongside roll and forward velocity. As yaw and roll are represented in angles, they introduce a numerical discontinuity at the periodic boundary (at 0° or 360°). Therefore, values that are physically close together (e.g., 1° and 359°) may yield an artificially large difference (358° instead of just 2°), possibly destabilizing the learning process through massive gradient spikes. Therefore, to circumvent the inherent issues with the periodic characteristics of angles, we encode roll and yaw using continuous sin and cos embeddings. The network predicts t_{horizon}/dt future waypoints uniformly spaced by time interval dt .



Fig. 4: RGB images taken at the same time instance (a,b,c,d) and their corresponding depth images (e,f,g,h).

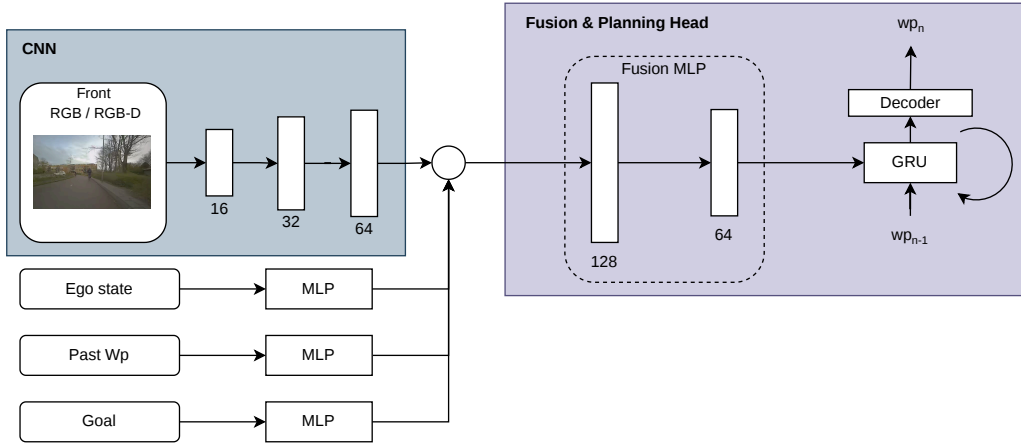


Fig. 5: The GRU-based E2E bicycle path planner with the RGB or RGB-D image input with the GRU planning head.

B. Architecture

Our proposed E2E planner comprises two primary components: a Convolutional Neural Network (CNN) for spatial feature extraction and a multimodal Fusion & Planning Head for waypoints prediction (Figure 5). Because bicycles are compact vehicles with limited payload capacity in terms of size, weight, and power consumption, we prioritize lightweight architectures that can run on low-power onboard hardware.

To process high-dimensional image data, we utilize a CNN following the PilotNet [4] concept. Convolutional layers are well-known for extracting prominent features from 2D spatial input, making them the optimal tool for this task. To maintain the required computational efficiency, we deliberately restrict the vision backbone to a compact 3-layer configuration.

The Fusion & Planning Head accepts the aggregated visual and non-visual features as the input, which is then fused by Fusion MLP and compressed to match the subsequent architecture. Because planning requires predicting a sequence of codependent spatial points, we use a Gated Recurrent Unit (GRU), following the approach used in Transfuser [7]. The

physical position and kinematics of a waypoint at step t directly influence the feasible prediction at step $t + 1$. Therefore, we use an autoregressive GRU that models these sequential dependencies by directly feeding previous predictions back into the network to generate the future waypoints.

C. Input Processing and Feature Extraction

Our E2E planner processes both spatial data and the ego-vehicle's kinematic state. The spatial input consists of RGB or RGB-D images from the front-facing camera. To construct the RGB-D images, 3-channel RGB data is concatenated with 16-bit depth images, resulting in a 4-channel RGB-D input. To encode this data, we utilize a lightweight 3-layer CNN as shown in Figure 5. The first layer uses a kernel size of 5 to capture a global receptive field, while subsequent layers use a kernel size of 3. Lastly, we apply 4x4 average adaptive pooling and subsequently flatten the output.

Alongside the imagery input, the network processes the kinematic ego state, historical trajectory waypoints, and a navigation goal. The current ego-state is explicitly provided by

the forward velocity, roll rate, and yaw rate along optionally encoded roll (via sin and cos embeddings) in the form of a 3- or 5-dimensional ego-state vector. Because turning at higher speeds necessitates greater roll angles, we explicitly use forward velocity in the ego-state. The roll and yaw rates directly encompass the current turning behavior.

To capture the temporal dynamics of the ego-vehicle, we provide a history of past 7-dimensional waypoints spanning t_{history} seconds, sampled at intervals of dt . This historical context incorporates position, roll, yaw, and forward velocity to allow for proper dynamics reasoning.

Finally, the network is conditioned on a navigational goal, represented as a 7-dimensional waypoint at a fixed future time horizon t_{goal} . A goal vector of such form encodes not only the target coordinates in the ego-centric frame but also the desired heading and roll. Detailed discussions regarding the dataset and architecture are provided in Appendix E and H.

To extract dense feature vectors from these non-visual inputs, the ego state, past waypoints, and goal are independently processed through dedicated Multi-Layer Perceptrons (MLPs) before reaching the fusion stage.

D. Multimodal Fusion and Recurrent Planning Head

To generate the final predictions, the extracted features are fused and sequentially decoded to form a feasible trajectory. As shown in Figure 5, the flattened spatial features are concatenated with the MLP-encoded ego-state, past waypoints, and goal to form a unified multimodal feature vector. This concatenated feature vector is fed through a 2-Layer Fusion MLP, which sequentially compresses the feature representation to 128 and finally to 64 dimensions. Following an approach similar to Transfuser [7], we further use this compressed 64-dimensional feature vector to initialize a hidden state of an autoregressive Gated Recurrent Unit (GRU) decoder. As further discussed in the ablation study, we evaluate two distinct GRU input dimensions. The 2-dimensional variant is initialized with a zero vector and autoregressively decodes future waypoints by feeding the isolated 2D position of the previous prediction wp_{n-1} back into the GRU. In contrast, the 7-dimensional variant is initialized with a zero vector containing only the ego-state’s current forward velocity, and feeds the entire 7-dimensional waypoint prediction wp_{n-1} in subsequent steps. This decoding procedure is repeated t_{horizon}/dt times, outputting a dynamically feasible future trajectory that inherently couples forward velocity, roll, yaw, and waypoint coordinates at every time step.

E. Loss function

Because bicycles operate at significantly lower speeds than cars, longitudinal errors are less critical as long as they remain within a reasonable, physically feasible range. In contrast, minimizing the lateral error should be prioritized, as the bike paths are significantly narrower than the car lanes and cyclists tend to ride closer to each other than cars. Therefore, we emphasize that the primary objective when training the bicycle E2E planners should be to minimize the lateral offset. To achieve this, we depart from the standard Average

Displacement Error (ADE) autonomous-driving metric, which combines lateral and longitudinal deviations in a single loss function. We emphasize that the longitudinal and lateral deviations should be evaluated separately for bicycles and provide quantitative evidence to support this. Therefore, we propose a decoupled evaluation metric and a composite loss function that explicitly prioritizes lateral offset penalization alongside dynamic constraints such as roll and velocity.

The composite loss function independently penalizes all components of the predicted waypoints W in the horizon:

$$\mathcal{L} = \frac{1}{W} \sum_{w=1}^W \left(\lambda_{\text{lat}} \mathcal{L}_{\text{lat}}^w + \lambda_{\text{lon}} \mathcal{L}_{\text{lon}}^w + \lambda_{\text{roll}} \mathcal{L}_{\text{roll}}^w + \lambda_{\text{yaw}} \mathcal{L}_{\text{yaw}}^w + \lambda_{\text{vel}} \mathcal{L}_{\text{vel}}^w \right) \quad (1)$$

where the weighting coefficients are set to $\lambda_{\text{lat}} = 0.43$, $\lambda_{\text{lon}} = 0.25$, $\lambda_{\text{roll}} = 0.125$, $\lambda_{\text{yaw}} = 0.125$, and $\lambda_{\text{vel}} = 0.07$.

Lateral loss: To calculate the lateral offset loss term $\mathcal{L}_{\text{lat}}^w$, the discrete expert trajectory is converted to a sequence of lines connecting consecutive 2D coordinates. The lateral error is then defined as the minimum distance between the predicted waypoint and the closest ground-truth line segment. To explicitly penalize corner-cutting behavior, the predicted point is projected onto the trajectory segments and clamped to the segment boundaries. A computationally efficient vectorized derivation is provided in Appendix G.

Longitudinal path length loss: While decoupling lateral and longitudinal losses, the longitudinal loss needs to be carefully crafted to account for road curvature. Therefore, we propose and implement a path length longitudinal loss ($\mathcal{L}_{\text{lon}}^w$). Building upon the vectorized segment approach utilized in the lateral loss derivation, the longitudinal path length is calculated as the total path length along the discrete trajectory up to the lateral projection. This formulation structurally eliminates the coordinate system vulnerability in which the network minimizes the loss by artificially compressing waypoints and stopping short during sharp maneuvers. A Huber Loss is further applied between the predicted and ground-truth longitudinal path lengths. A detailed derivation building upon the vectorized lateral loss can be found in Appendix G.

Roll and yaw loss: To penalize for roll and yaw angular deviations ($\mathcal{L}_{\text{roll}}^w$ and $\mathcal{L}_{\text{yaw}}^w$), the predicted sin and cos embeddings are normalized to the unit circle yielding a valid angle representation. The sum of squared errors between the normalized predictions and the ground-truth embeddings is then used to calculate the loss for a waypoint.

Velocity loss: Although velocities could be deduced solely from the 2D coordinates of consecutive waypoints, relying purely on implicit spatial predictions may yield noisy or invalid velocities. Therefore, a dedicated Huber Loss ($\mathcal{L}_{\text{vel}}^w$) is applied to explicitly supervise forward velocity predictions, enforcing kinematic consistency and generating a more stable reference signal suitable for downstream low-level controllers.

V. EXPERIMENTS

In this section, we describe our experimental setup for the E2E planner for bicycle. Additionally, we present an ablation

Model	Past	Planning head	GRU input dim	Roll	Lat. err	Lon. err	Vel. err	Roll err	Yaw err
2nd-degree polynomial	-	-	-	-	31.90	60.06	-	-	-
RGB Input Models									
Goal	-	MLP	-	-	19.73	10.69	<u>0.42</u>	1.32	<u>2.51</u>
Past	✓	MLP	-	-	19.29	9.47	0.45	1.39	2.59
Gru0	✓	GRU	2	-	18.48	9.14	<u>0.42</u>	1.32	2.48
Gru0 7wp	✓	GRU	7	-	18.02	9.00	0.41	1.44	2.57
Gru0 roll	✓	GRU	2	✓	18.65	7.26	0.43	<u>1.35</u>	2.54
Gru0 7wp roll	✓	GRU	7	✓	<u>18.47</u>	<u>8.07</u>	<u>0.42</u>	1.51	2.63
RGB-D Input Models									
Goal	-	MLP	-	-	28.96	42.34	<u>0.47</u>	<u>1.53</u>	3.67
Past	✓	MLP	-	-	29.85	14.37	0.44	1.68	4.53
Gru0	✓	GRU	2	-	22.80	21.03	0.50	1.67	2.90
Gru0 7wp	✓	GRU	7	-	25.79	<u>11.96</u>	0.44	1.58	3.48
Gru0 roll	✓	GRU	2	✓	38.17	16.01	0.53	1.90	4.15
Gru0 7wp roll	✓	GRU	7	✓	<u>24.12</u>	9.62	0.44	1.51	2.90

(a) Evaluation of errors averaged on the **full validation set**.

Model	Past	Planning head	GRU input dim	Roll	Lat. err	Lon. err	Vel. err	Roll err	Yaw err
2nd-degree polynomial	-	-	-	-	59.75	128.07	-	-	-
RGB Input Models									
Goal	-	MLP	-	-	32.44	17.67	<u>0.42</u>	1.95	<u>4.52</u>
Past	✓	MLP	-	-	32.96	16.10	0.44	2.04	4.65
Gru0	✓	GRU	2	-	28.54	16.91	<u>0.42</u>	1.96	4.50
Gru0 7wp	✓	GRU	7	-	28.16	17.02	0.41	2.24	4.56
Gru0 roll	✓	GRU	2	✓	29.13	13.97	<u>0.42</u>	2.07	4.55
Gru0 7wp roll	✓	GRU	7	✓	29.55	<u>15.44</u>	0.41	2.24	4.76
RGB-D Input Models									
Goal	-	MLP	-	-	45.63	53.89	0.48	2.24	6.64
Past	✓	MLP	-	-	43.45	26.53	<u>0.46</u>	2.35	6.39
Gru0	✓	GRU	2	-	35.60	32.28	0.50	2.43	5.04
Gru0 7wp	✓	GRU	7	-	47.60	<u>23.48</u>	0.44	2.48	6.61
Gru0 roll	✓	GRU	2	✓	61.48	27.68	0.53	2.42	6.75
Gru0 7wp roll	✓	GRU	7	✓	<u>40.00</u>	17.77	0.44	<u>2.33</u>	<u>5.43</u>

(b) Evaluation of errors averaged on the **hard subset** of a validation set.

TABLE III: The tables compare both RGB and RGB-D input modalities. Past denotes the use of historic waypoints; Planning head – the architecture used to predict waypoints; GRU input dim – input dimension to the GRU autoregressive layer (wp_{n-1}); Roll – inclusion of embedded roll in ego-state; Lat. err. – lateral offset error from the expert’s trajectory in centimeters; Lon. err. – longitudinal error between predicted waypoint and ground-truth in centimeters; Vel. err - velocity error in m/s; Roll err – roll deviation between ground-truth and prediction in degrees; Yaw err – yaw deviation between ground-truth and prediction in degrees.

study to highlight the importance of each architectural component and to ground the need to depart from the ADE metric for bicycles. Furthermore, we provide results for RGB and RGB-D model variations and quantify the models’ estimation of coupled cornering characteristics. All methods are evaluated on both the validation set and its more challenging subset (hard validation set). A non-learnable baseline is established as a reference in the ablation study.

A. Training setup

We utilize our *BikeScenes-planning* dataset for both training and validation. After excluding the validation set, the remaining data was split into training and early stopping subsets (10%), with the latter used to monitor generalization and trigger early stopping. All architectures were trained using a batch size of 128 on an NVIDIA RTX 5000 Ada GPU. The input to RGB models is an RGB image from the left

lens of the stereo camera. Consequently, the RGB-D models take as input RGB images concatenated with 16-bit depth images, yielding RGB-D representations with a resolution of 960×600 pixels. For the temporal parameters, we define the history horizon t_{history} as 2 seconds, the future prediction horizon t_{horizon} as 4 seconds, and the temporal resolution dt as 1 second. Consequently, our networks predict waypoints for a 4-second future horizon at 1-second intervals. We optimize the identical loss function defined in Equation 1 across all models.

As previously described in Section IV-E, the primary objective of our E2E planning approach is to minimize the lateral offset from the expert’s trajectory. Consequently, to mitigate overfitting, an early stopping mechanism with a patience of 4 epochs was implemented, tied to the lateral loss on the early stopping subset. Subsequently, each planner is evaluated at discrete intervals within a 4-second horizon, and the average error across the entire timeframe is reported.

Gru0							
RGB	Depth (EF)	Depth (LF)	Lat. err	Lon. err	Vel. err	Roll err	Yaw err
✓	-	✓	45.03	125.22	0.49	1.94	7.26
-	✓	-	36.07	24.10	<u>0.44</u>	2.35	5.73
✓	✓	-	<u>22.80</u>	<u>21.03</u>	0.50	<u>1.67</u>	<u>2.90</u>
✓	-	-	18.48	9.14	0.42	1.32	2.48

TABLE IV: Fusion stage comparison and only-depth variation of the best RGB-D model evaluated on a validation set. EF - early fusion (RGB-D image); LF - late fusion (RGB and Depth image branches). Lat. err. – lateral offset error from the expert’s trajectory in centimeters; Lon. err. – longitudinal error between predicted waypoint and ground-truth in centimeters; Vel. err - velocity error in m/s; Roll err – roll deviation between ground-truth and prediction in degrees; Yaw err – yaw deviation between ground-truth and prediction in degrees.

Metric	Cartesian Loss	Path Length Loss
Progress Ratio $\uparrow$	0.920	0.985
Path Length Error (s_mae) $\downarrow$	0.789	0.534
Jerk $\downarrow$	0.377	0.336
Vel. Error Correlation (r) $\downarrow$	0.855	0.360
Pred. Explicit Speed Corr. (r)	-0.338	-0.311
Pred. Spatial Speed Corr. (r)	-0.617	-0.356

TABLE V: Comparison of longitudinal loss functions between two *Gru0* *7wp* models optimized for cartesian and path length longitudinal loss evaluated on the subset of the validation set containing only sharp turns. The **expert’s Spatial Speed Corr. (r)** is $r_{\text{expert}} = -0.276$ and calculated from the expert demonstrations. The model trained with our longitudinal path length loss successfully aligns both spatial and explicit velocity predictions with an expert baseline. Negative values in Speed Corr. indicate a slowdown behavior while cornering, whereas the positive Vel. Error Correlation shows the increased absolute error in velocity predictions (derived from spatial data) while turning.

B. Cartesian vs path length longitudinal loss

The results presented in the Table V demonstrate the necessity of adopting the path length longitudinal loss over trivial Cartesian loss in the ego-centric frame. We evaluated our best performing model on the lateral offset error metric, *Gru0*, optimized for either of the losses on the subset of scenes containing only sharp turns. The scenes containing at least one $\sim 90^\circ$ turn were manually selected from the validation set because the Cartesian and longitudinal path length losses differ marginally on straight road segments but exhibit major differences during cornering.

The model optimized for our path length longitudinal loss exhibited a significantly lower path-length error by 32.3% and a drastically lower velocity error correlation (Vel. Error Correlation $r = 0.360$, -57.9% reduction) between the spatial velocity errors and cornering. Additionally, its velocity calculated from spatial predictions showed a nearly expert-like (r_{expert}) correlation (Pred. Spatial Speed Corr. $r = -0.356$) between speed and cornering maneuver.

This evaluation proves that our proposed metric eliminates the vulnerability of Cartesian loss that causes severe and unrealistic braking while cornering. It directly prevents the network from artificially minimizing error by congesting infeasible waypoints at the turn’s apex, explicitly enforcing the

longitudinal progress along the expert trajectory. Although in autonomous driving the goal of E2E planners is mostly to maximize progress, our method is optimized to achieve the same progress along the path as the expert (a *Progress Ratio* of 1). Such a target enforces the appropriate spatial spacing between the waypoints and, therefore, a feasible and safe path while cornering. A qualitative comparison of the two losses in a sharp turn scenario is further provided in Appendix I-B.

C. Non-learnable baseline

As there are no E2E path planners for bicycles, we establish a non-learning reference baseline in the form of 2nd-degree polynomial and perform a curve fitting between the current ego-pose and the goal. To ensure a fair comparison, we isolate the samples at the 1, 2, 3, and 4 second timesteps for evaluation. Even though the polynomial method performed better than the worst model (Table III), the qualitative analysis in Appendix I-A shows that such a polynomial-based path-planning method, which does not utilize imagery input, fails catastrophically during cornering since it cannot build an appropriate understanding of the scene.

D. Ablation study

We investigate the effects of past information, planning head, GRU input dimension, and roll incorporation in the state vector and present the average metrics for lateral offset error, longitudinal velocity, and roll and yaw errors. The evaluation results are shown in the Table III.

Past information: The incorporation of the past samples marginally improved or worsened the performance on the lateral metric. The cause of these marginal improvements may be due to the already information-rich ego-state input. However, the past information improves the prediction on the secondary metric - longitudinal path length. Most notably, it decreased the longitudinal error for the RGB-D model by nearly 3 times (from 42.34 cm to 14.37 cm) and more than 2 times (from 53.89 cm to 26.53 cm) on the validation set and the more challenging hard subset, respectively. By using past samples, the model can infer the speed and interpolate it to accurately predict the spatial spacing between the future waypoints, thereby reducing the longitudinal path length error. The improvements are noticeable across all validation sets and model variations. As this metric is the second most important metric, and the influence of the past samples on lateral offset

is significant, we use this past waypoint information in further evaluation. Therefore, these past data samples are used in all evaluated models containing the GRU planning head.

Planning head: We investigate the influence of the simple MLP planning head and memory-preserving GRU. Overall, the added complexity introduced by the GRU improved the models’ performance. Most notably, the performance increase in RGB-D variations evaluated on the validation set yielded a significant decrease in lateral error by 21.27% (6.16 cm) between the best MLP (*Goal*) and GRU (*Gru0*) model. Overall, incorporating the GRU planning head yielded lower lateral error across all RGB models. We argue that the GRU, due to its memory-preserving architecture, can learn the underlying dynamics of bicycle motion and therefore outperforms single-shot models with an MLP planning head. A detailed discussion is provided in Appendix I-D and I-E.

GRU input dimension: For the RGB models, utilizing a 7-dimensional predicted waypoint vector as input to the GRU layer improved the performance on the lateral error metric. Although the improvements are not major, the consistency across models shows that the embedded roll, heading, and forward velocity inputs to the GRU layer improve spatial predictions. We hypothesize that this is strictly bound to the underlying dynamics of bicycle motion, in which all waypoint components are codependent and must be coupled to yield a valid path. Surprisingly, increasing the GRU input waypoint dimensionality marginally increased the roll prediction error. It may be due to noise introduction as the network learns the association between roll and cornering from pure spatial coordinates. Moreover, the RGB-D models utilizing 7-dimensional GRU input exhibit a significant improvement in the longitudinal error metric compared to their 2D input equivalents. We argue that the GRU layers leverage the additional information embedded in the 7D waypoint to build an understanding of proper progression along the path.

Roll in state vector: The explicit incorporation of the roll in the state vector (5-dimensional ego-state) yields mixed results on the lateral error metric. The reduction in lateral errors is most prominent in RGB-D models with 7-dimensional GRU inputs evaluated on the hard subset. The roll in ego-state vector reduced the error for *Gru0 7wp* model by 7.6 cm (a 15.97% reduction). However, it has a significant positive impact on the longitudinal error reduction across both input modalities. The best RGB-D model on longitudinal error metric evaluated on the hard validation set, *Gru0 7wp roll*, noted a significant error reduction of 5.71 cm (a 24.3% reduction) in comparison to its equivalent using 3-dimensional ego-state input, *Gru0 7wp*. Moreover, the best-performing models on the longitudinal path length metric always incorporate an embedded roll in the ego-state vector. We argue that roll may act as a stabilizing component for the longitudinal path length metric. Because of the bicycle’s coupled dynamics, the roll addition yields a complete current dynamic state, which is essential for accurately inferring speed constraints (especially while cornering), and results in appropriately spaced waypoints along the path. Even though the longitudinal path length metric should not be the predominant one in E2E, incorporating roll into the state vector and minimizing this metric should be further explored.

Model	Lat. err. ↓	Lon. err. ↓	ADE ↓
Goal	45.63	53.89	111.33
Past	43.45	26.53	<u>77.74</u>
Gru0	35.60	32.28	83.85
Gru0 7wp	47.60	<u>23.48</u>	78.19
Gru0 roll	61.48	27.68	96.09
Gru0 7wp roll	<u>40.00</u>	17.77	66.69

TABLE VI: ADE vs Lat/Lon comparison of RGB-D models evaluated on the hard validation set. The results highlight the necessity of the decoupled metric as the best model on predominant metric scores 3rd worst on ADE.

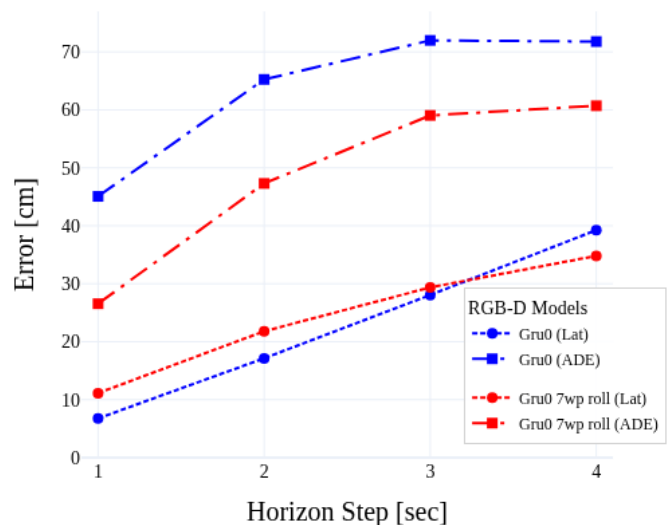


Fig. 6: Average ADE vs Lateral Error over the 4-second prediction horizon with a timestep of 1 second for the best RGB-D model on lateral error (*Gru0*) and ADE (*Gru0 7wp roll*) metric evaluated on the entire validation set. The ■ denotes the scores obtained on the ADE metric, whereas ● denotes scores on the lateral error metric.

E. ADE vs decoupled Lat/Lon loss

The evaluation of RGB-D models on the challenging hard subset, detailed in Table VI, highlights the fundamental discrepancy between ADE and the decoupled lateral and longitudinal metrics. While the *Gru0* achieved the lowest lateral offset (35.60 cm), it performed poorly under the ADE metric, recording the third-highest error (83.85 cm). Conversely, the model with the lowest ADE score (*Gru0 7wp roll*) exhibited a 12.4% (4.40 cm) increase in lateral offset error compared to the top-performing lateral model. This performance decrease is even more pronounced in the second-best ADE model, which uses past waypoint information and an MLP planning head (*Past*), exhibiting a 22.1% (7.85cm) increase in lateral error. This inverse correlation highlights the inherent flaw in the spatially coupled nature of the ADE.

Excluding the simplest planner (*Goal*), the longitudinal deviations across all evaluated models fall within a small, less than 15-centimeter range. With the average recorded velocity of less than 4 m/s (14.4 km/h), this error corresponds to SenseBike arriving at the longitudinal waypoint split of a second earlier or later. Because the ADE utilizes both

spatial dimensions equally, a model can achieve a superior ADE score by minimizing the along-track progress error while simultaneously exhibiting a significant lateral deviation. This phenomenon can lead to incorrect conclusions about the model’s performance when the model exhibits superior ADE performance but actually performs worse on the predominant lateral error metric, as shown in Figure 6. As a result, the E2E planners for bicycles should neither be optimized nor evaluated on ADE, as it can actively mask the critical lateral offset error that governs the safe bicycle path, especially in narrow bicycle lanes. A detailed discussion and analysis of the prediction horizon are provided in Appendix I-C.

F. RGB vs RGB-D

Including a 16-bit depth image as a 4th channel degraded performance across all models on most metrics. As shown in Table III, the best RGB-D model, *Gru0*, exhibits a 4.78 cm larger average lateral error than the best performing RGB model, *Gru0 7wp*, on the entire validation set (a 26.5% increase). Moreover, the performance comparison between depth fusion approaches is presented in the Table IV. The late fusion variation of the best RGB-D model, having separate branches for RGB and depth image, exhibits even worse performance on lateral error (45.03 cm) and marks a catastrophic longitudinal deviation from the expert trajectory of 125.22 cm. In our comparison we do not pretrain the CNN depth-vision branch (and depth branch for late fusion model). The pretraining of the depth-vision branch on an auxiliary task (e.g., drivable area prediction) or copying the corresponding CNN weights from the RGB models could potentially improve the performance and should be further explored. Additionally, this performance degradation may be caused by inaccurate depth images and, therefore, noise injection in our model. Due to hardware limitations discussed in Appendix C, the cameras were recording at the lowest resolution, effectively preventing the use of the best-performing depth estimation pipeline (further discussed in Appendix I-F).

G. Roll-turn correlation

To investigate models’ ability to correctly couple the lean-in behavior with turning, a detailed correlation study is performed, and the results are presented in Table XVI. All models were evaluated on a carefully selected validation subset, containing only sharp turns. Both RGB and RGB-D models learned the importance of correctly coupling roll with the change in heading. The best models, *Gru0* (RGB) and *Gru0 7wp* (RGB-D), learned a near human-like correlation (-81.45), scoring -82.70 and -82.88, respectively. These results show that the Behavioral Cloning paradigm can successfully be implemented to learn the coupled bicycle dynamics.

H. Qualitative results

The qualitative results further validate the ability of the best RGB E2E bicycle planner, *Gru0 7wp*, to successfully plan the path for a bike. As shown in Figure 1, the planner not only predicts future waypoints accurately but also associates

Model	RGB corr	RGB-D corr
GT	-81.45	
Goal	-86.19	-89.05
Past	-84.04	-90.50
Gru0	-82.70	-87.47
Gru0 7wp	-86.90	-82.88
Gru0 roll	-87.56	-77.00
Gru0 7wp roll	-76.98	-84.20

TABLE VII: Correlation (corr) between roll and turning for all RGB and RGB-D models variations evaluated on the validation set subset containing only sharp turns. GT - denotes the expert’s correlation.

its spatial prediction with roll. For that particular scenario, the deviations in predicted and ground-truth roll in a 3-second horizon are rather minimal, averaging to an absolute difference of $\Delta = 0.9^\circ$. Moreover, the predicted waypoints closely match the experts’ trajectory, with nearly perfect heading. The minimal longitudinal difference is expected as the correlation of the velocity calculated from spatial predictions with the turning is smaller ($r = -0.356$) than the expert’s correlation ($r = -0.276$), resulting in lower speeds while cornering, thus the predictions are located closer to the ego-vehicle. Additional visualizations are provided in Appendix I-H.

VI. DISCUSSION & LIMITATIONS

We successfully applied the Behavioral Cloning paradigm to create a first-ever end-to-end bicycle path planner. To train our planner, we developed the data acquisition platform (SenseBike) to collect raw data and subsequently curated a pioneering multi-modal *BikeScenes-planning* dataset.

Even though models using single-modality RGB inputs outperform their RGB-D equivalents in our study, we encourage the research community to further investigate the incorporation of depth information into E2E bicycle path planners. The second iteration of the *BikeScenes-planning*, which will be available to the public, will be captured at a higher resolution and will incorporate higher-fidelity depth images. Additionally, a dedicated test set will be curated to further assess the models’ generalization abilities.

Lastly, the *BikeScenes-planning* dataset should not be limited to the E2E bicycle path planning task. On the contrary, we invite researchers to use this dataset for other means, including object detection, motion forecasting, or behavioral modeling.

REFERENCES

- [1] W. Gillan, “Prometheus and drive: their implications for traffic managers,” in *Conference Record of papers presented at the First Vehicle Navigation and Information Systems Conference (VNIS ’89)*, 1989, pp. 237–243.
- [2] R. Woods and J. Masthoff, “A comparison of car driving, public transport and cycling experiences in three european cities,” *Transportation Research Part A: Policy and Practice*, vol. 103, pp. 211–222, 2017. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0965856415300318>
- [3] L. Chen, P. Wu, K. Chitta, B. Jaeger, A. Geiger, and H. Li, “End-to-end autonomous driving: Challenges and frontiers,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 12, pp. 10 164–10 183, 2024.

- [4] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," 2016. [Online]. Available: <https://arxiv.org/abs/1604.07316>
- [5] Y. Hu, J. Yang, L. Chen, K. Li, C. Sima, X. Zhu, S. Chai, S. Du, T. Lin, W. Wang, L. Lu, X. Jia, Q. Liu, J. Dai, Y. Qiao, and H. Li, "Planning-oriented autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2023, pp. 17 853–17 862.
- [6] Z. Li, K. Li, S. Wang, S. Lan, Z. Yu, Y. Ji, Z. Li, Z. Zhu, J. Kautz, Z. Wu, Y.-G. Jiang, and J. M. Alvarez, "Hydra-mdp: End-to-end multimodal planning with multi-target hydra-distillation," 2024. [Online]. Available: <https://arxiv.org/abs/2406.06978>
- [7] A. Prakash, K. Chitta, and A. Geiger, "Multi-modal fusion transformer for end-to-end autonomous driving," *CoRR*, vol. abs/2104.09224, 2021. [Online]. Available: <https://arxiv.org/abs/2104.09224>
- [8] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.
- [9] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nusscenes: A multimodal dataset for autonomous driving," in *CVPR*, 2020.
- [10] 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, Y. Zhang, J. Shlens, Z. Chen, and D. Anguelov, "Scalability in perception for autonomous driving: Waymo open dataset," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.
- [11] J. Houston, G. Zuidhof, L. Bergamini, Y. Ye, L. Chen, A. Jain, S. Omari, V. Iglovikov, and P. Ondruska, "One thousand and one hours: Self-driving motion prediction dataset," in *Conference on Robot Learning*. PMLR, 2021, pp. 409–418.
- [12] K. T. e. a. H. Caesar, J. Kabzan, "Nuplan: A closed-loop ml-based planning benchmark for autonomous vehicles," in *CVPR ADP3 workshop*, 2021.
- [13] C. Odenwald and M. Beeking, "Sabi3d—a lidar point cloud data set of car-to-bicycle overtaking maneuvers," *Data*, vol. 9, no. 8, p. 90, 2024.
- [14] M. A. Buettner, K. Mazumder, L. Koecher, M. Finkbeiner, S. Niebler, and F. B. Flohr, "Bikeactions: An open platform and benchmark for cyclist-centric vru action recognition," *arXiv preprint arXiv:2601.10521*, 2026.
- [15] D. Goren and H. Caesar, "Bikescenes: Online lidar semantic segmentation for bicycles," *arXiv preprint arXiv:2510.25901*, 2025.
- [16] D. Dauner, M. Hallgarten, A. Geiger, and K. Chitta, "Parting with misconceptions about learning-based vehicle motion planning," 2023. [Online]. Available: <https://arxiv.org/abs/2306.07962>
- [17] B. Jiang, S. Chen, Q. Xu, B. Liao, J. Chen, H. Zhou, Q. Zhang, W. Liu, C. Huang, and X. Wang, "Vad: Vectorized scene representation for efficient autonomous driving," 2023. [Online]. Available: <https://arxiv.org/abs/2303.12077>
- [18] S. Chen, B. Jiang, H. Gao, B. Liao, Q. Xu, Q. Zhang, C. Huang, W. Liu, and X. Wang, "Vadv2: End-to-end vectorized autonomous driving via probabilistic planning," 2024. [Online]. Available: <https://arxiv.org/abs/2402.13243>
- [19] H. Shao, L. Wang, R. Chen, H. Li, and Y. Liu, "Safety-enhanced autonomous driving using interpretable sensor fusion transformer," 2022. [Online]. Available: <https://arxiv.org/abs/2207.14024>
- [20] H. Shao, L. Wang, R. Chen, S. L. Waslander, H. Li, and Y. Liu, "Reasonnet: End-to-end driving with temporal and global reasoning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 13 723–13 733.
- [21] W. Sun, X. Lin, Y. Shi, C. Zhang, H. Wu, and S. Zheng, "Sparsedrive: End-to-end autonomous driving via sparse scene representation," in *2025 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2025, pp. 8795–8801.
- [22] L. Liu, C. Jia, G. Yu, Z. Song, J. Li, F. Jia, P. Wu, X. Hao, and Y. Luo, "Guideflow: Constraint-guided flow matching for planning in end-to-end autonomous driving," 2026. [Online]. Available: <https://arxiv.org/abs/2511.18729>
- [23] K. Guo, H. Liu, X. Wu, J. Pan, and C. Lv, "ipad: Iterative proposal-centric end-to-end autonomous driving," 2025. [Online]. Available: <https://arxiv.org/abs/2505.15111>
- [24] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015. [Online]. Available: <https://arxiv.org/abs/1512.03385>
- [25] S. Umeyama, "Least-squares estimation of transformation parameters between two point patterns," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 13, no. 4, pp. 376–380, 1991.
- [26] W. Kabsch, "A solution for the best rotation to relate two sets of vectors," *Foundations of Crystallography*, vol. 32, no. 5, pp. 922–923, 1976.
- [27] F. Dellaert and G. Contributors, "borglab/gtsam," May 2022. [Online]. Available: <https://github.com/borglab/gtsam>

APPENDIX TABLE OF CONTENTS

Appendix A: SenseBike	14
A-A Single Board Computer	14
A-B Perception sensors	14
A-C Global Positioning	14
A-D User Control Interface	14
Appendix B: 360-degree Camera setup	14
B-A Prerequisites	14
B-B Camera sensors	14
B-C Software Development Kit	15
B-D Camera sensors mounting points	15
Appendix C: Data capture	15
C-A Capture procedure and setup	16
C-B Captured data formats	18
C-C Capture	18
Appendix D: Data curation	20
D-A Camera files preprocessing	20
D-B Frame information extraction	20
D-C Camera sequence synchronization	20
D-D Frame drop analysis	20
D-E Relax fill for dropped frames	21
D-F Stationary periods detection	23
D-G Scenes generation and turn detection	23
D-H Local pose estimation	24
D-I Global pose estimation	24
D-J Frame extraction	27
D-K Visual inspection	27
D-L Dataset structure and information	27
Appendix E: BikeScenes - planning dataset	28
E-A Training data temporal filtering	28
E-B Waypoints	28
E-C Trajectory	28
E-D Goal selection	28
E-E State vector	29
E-F Image fetching	29
E-G Dataset precomputation	29
Appendix F: Validation set	29
F-A Validation recording proposals	30
F-B Visual inspection	30
F-C Hard validation set	30
Appendix G: Loss function and evaluation metric	30
G-A Composite loss function	30
G-B Lateral loss	31
G-C Longitudinal path length loss	31
G-D Roll and yaw loss	32
G-E Velocity loss	32
Appendix H: E2E planners	32
H-A Architectures	32

Appendix I: Experiments	32
I-A Non-Learnable Geometric Baselines	32
I-B Path length loss vs Cartesian Loss	33
I-C ADE vs lat/lon decoupled loss	35
I-D Lateral error in prediction horizon	35
I-E Longitudinal path length error in prediction horizon	35
I-F RGB vs RGB-D	36
I-G Speed/Roll - turn correlation	36
I-H Qualitative Examples	37

APPENDIX A

SENSEBIKE

SenseBike is a bicycle data acquisition platform equipped with a multi-modal sensor suite. This hardware configuration is designed to capture high-fidelity spatial and semantic information regarding the ego-vehicle’s surroundings while simultaneously providing precise global positioning and vehicle pose estimation.

A. Single Board Computer

The software on the SenseBike is governed by a Single Board Computer – NVIDIA Jetson Orin NX 16 GB. The device operates on an Ubuntu 20.04-based Linux for Tegra (L4T) 35.4.1 operating system, provided by NVIDIA JetPack 5.1.2. To manage conflicting software dependencies – specifically the simultaneous operations of ROS 2 Galactic, ROS 2 Humble, and the Stereolabs ZED SDK (versions 5.1 and 5.2) – the entire software stack is strictly containerized using Docker.

To ensure sustained computational performance and prevent thermal throttling under heavy sensor processing loads, the SBC and the Stereolabs Capture Card are housed in a custom-manufactured enclosure equipped with an active push-pull thermal management system. As illustrated in Figure 7, this system utilizes dual fans mounted diagonally to enforce proper cross-ventilation. A side-mounted intake fan forces ambient air into the enclosure, while a rear-mounted exhaust fan actively extracts heated air. To mitigate moisture and particulate ingress, mesh filters were installed at the intake and outlet. This optimized thermal design maintains safe operating temperatures for the core processing units during extended data capture sessions across moderate weather conditions.

B. Perception sensors

SenseBike is equipped with a multi-modal sensor suite designed to capture full 360° spatial coverage around the ego-vehicle. The vision system, detailed in Appendix B, collects data alongside a three-sensor LiDAR array. This LiDAR configuration comprises one solid-state Robosense M1P sensor positioned at the front – directly below the forward-facing camera – and two Robosense Helios 32-channel sensors mounted at the rear (Figure 10). The data from both sensor modalities is logged on the SBC.

C. Global Positioning

To obtain precise global positioning for the SenseBike, the system utilizes a Global Navigation Satellite System (GNSS) setup enhanced with Real-Time Kinematic (RTK) positioning. This is achieved using an ArduSimple simpleRTK2B GNSS/RTK board, which connects to the SBC via a USB hub. The board integrates a u-blox ZED-F9P multi-band GNSS module paired with an external GNSS antenna. RTK corrections are gathered from the NTRIP caster operated by the Dutch Geodetic Infrastructure Cooperation (NSGI). The necessary network connection to the caster is established through a cellular network using a Teltonika router, resulting in correction streams at a frequency of 1 Hz. For data acquisition

within the city of Delft, the system was configured to use the DLF100NLD base station located on the TU Delft campus. To ensure sufficient temporal resolution for vehicle dynamics, the u-blox module is configured to output the final GNSS data at a frequency of 10 Hz.

D. User Control Interface

During the primary data collection phase for the BikeScenes-planning dataset, sensor operation was managed through the web graphical user interface (GUI). The operator connected to the SenseBike’s local WiFi access point and interacted with a Flask-based web application via a smartphone.

To ensure a higher reliability and bypass the limitations of the Flask framework, the platform was later upgraded with a dedicated hardware interface. A Waveshare 7HP-CAPQLED touch screen was mounted directly onto the handlebars (Figure 8), allowing the operator to manage data acquisition without relying on local network stability. A custom enclosure was designed in SolidWorks and manufactured using Fused Deposition Modeling (FDM) 3D printing manufacturing technology from Polyethylene Terephthalate Glycol (PETG). At the software level, the SBC’s display manager was configured and calibrated for portrait orientation to optimize the operator’s interface layout.

APPENDIX B

360-DEGREE CAMERA SETUP

To align with the sensor configurations of prominent autonomous driving datasets, such as nuScenes [9], nuPlan [12], or Waymo Open Dataset [10], the SenseBike was equipped with a comprehensive camera suite. Because this setup was integrated into SenseBike’s existing sensor stack, adherence to strict hardware and system prerequisites while selecting the camera sensors was required.

A. Prerequisites

SenseBike’s operation is governed by a Single Board Computer (SBC) – an NVIDIA Jetson Orin NX 16GB – which manages all software processes. Because the existing sensor suite was already integrated with this SBC, the new camera system had to be fully compatible with the NVIDIA Jetson ecosystem. Furthermore, with most of the SBC’s USB-A ports already utilized, the cameras required an alternative data interface. Relying on USB connections was also undesirable from a technical standpoint, as precise hardware synchronization across multiple USB cameras is notoriously difficult to achieve. Supplying power to cameras presented another major constraint: to prevent computational throttling, the cameras could not draw power directly from the SBC. Moreover, as SenseBike relies on a single battery to power all onboard electronics, the camera setup needed to be highly power-efficient, drawing less than 10 watts in total.

B. Camera sensors

Based on the prerequisites outlined in Appendix B-A, we selected the four-camera array utilizing Stereolabs ZED X

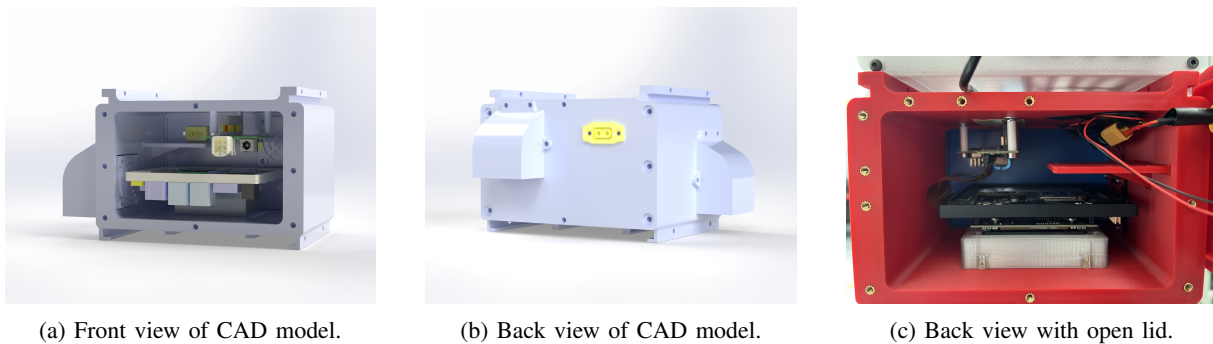


Fig. 7: NVIDIA Jetson and Capture Card enclosure mounted on SenseBike.



Fig. 8: Control display mounted on a handlebar.

stereo cameras. These sensors feature a global shutter and a built-in polarizer to mitigate motion blur and glare during outdoor urban data collection. Additionally, the integrated high-frequency Inertial Measurement Unit (IMU) within each camera provides crucial data for precise pose estimation.

To achieve full 360° spatial coverage while minimizing blind spots, we prioritized the horizontal field of view (FOV). Therefore, we opted for a camera variant with a 2.2 mm focal length, which yields a 120° horizontal FOV and a maximum depth range of 20 meters. As illustrated in Figure 11, this configuration guarantees sufficient overlap between adjacent camera frames, eliminating the blind spots that would occur with the alternative 4 mm (80° FOV) variant.

Data transfer and power delivery are handled via the GMSL2 protocol. Because the NVIDIA Jetson carrier board lacks native GMSL2 support, we integrated a Stereolabs Capture Card (CC) DUO. The CC connects directly to the SBC's carrier board via two 22-to-15-pin CSI cables, utilizing both available CSI ports to handle the bandwidth of four high-resolution stereo streams (Figure 9). The CC directly powers the cameras and manages hardware-level synchronization by triggering all sensors simultaneously, ensuring the physical image exposures occur at the exact same instant. Even though the GMSL2 protocol is serialized and the captured images become available in the SBC's memory asynchronously, all frames captured within a single grab period (the interval

between two consecutive hardware triggers) are temporally aligned. Ultimately, this architecture reliably streams four hardware-synchronized HD1200 stereo video feeds at 15 Hz. Further synchronization steps are detailed in Appendix D.

C. Software Development Kit

Camera sensor control and multi-stream data flow are managed by the official ZED Software Development Kit (SDK). Specifically, we utilize the Positional Tracking module from SDK to accurately estimate the camera's pose as detailed in Appendix D. The SDK uses the SBC's system clock for data processing, ensuring temporal consistency of the multi-modal sensor suite of SenseBike.

D. Camera sensors mounting points

The physical placement of the camera array was selected to maximize overlapping coverage at the front of the bicycle, as forward-facing redundancy is critical for effective path planning. Consequently, three cameras were mounted on the bicycle's front rack (Figure 11b). To minimize lateral blind spots while ensuring the rider's body would not occlude the field of view, the left and right front cameras were angled outward at a 92° offset relative to the vehicle's longitudinal axis of motion.

The custom camera mounts were designed in SolidWorks and manufactured via FDM using PETG as the primary material, following rapid prototyping principles. The front cameras are secured to a dedicated mounting plate on the front rack, while the rear camera is directly attached to the LiDAR support beam. The mounting point of the rear camera has been carefully chosen to prevent any occlusion caused by the supporting beam. The mounting points are shown in Figure 10.

APPENDIX C DATA CAPTURE

Data acquisition was conducted in the vicinity of Delft, Netherlands, utilizing the SenseBike's complete multi-modal sensor suite detailed in Table VIII. To ensure proper capture, the operator needs to follow a strict startup and recording procedure. The specific workflow depends on the recording approach, as described in Sections C-A1 and C-A2. The data is stored in two separate formats to ensure modularity and minimize computing overhead.

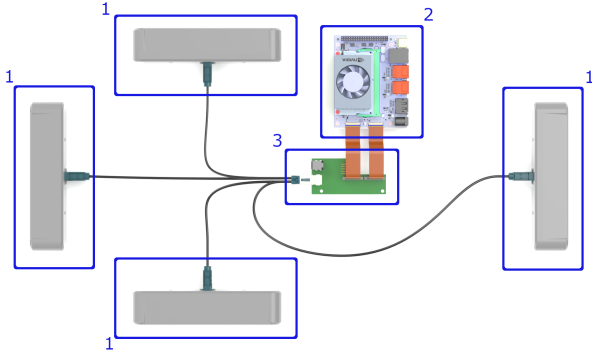


Fig. 9: Components of camera setup. 1 – ZED X camera; 2 – NVIDIA Jetson Orin NX 16 GB; 3 – Capture Card Duo.

Sensor	Qty	Pub. Freq. (Hz)
ZED X camera	4	15
Robosense M1P LiDAR	1	10
Robosense Helios 32 LiDAR	2	10
u-blox ZED-F9P (GNSS)	1	10
ArduSimple simpleRTK2B (RTK)	1	1

TABLE VIII: Sensors used in data capture and their publish frequencies.

A. Capture procedure and setup

SenseBike’s data acquisition pipeline was modified to integrate the 360° ZED camera array described in Appendix B. To maintain system stability, we adhered to a modular design philosophy, implementing a dedicated Docker container specifically for the cameras (hereafter referred to as the ZED container). This approach facilitated rapid integration and aligned seamlessly with the architecture of the existing sensor suite.

Despite utilizing a ROS 2 component container for the ZED nodes to minimize CPU overhead, the system experienced resource saturation during capture, resulting in dropped frames. These missing frames are mitigated during post-processing, as detailed in Appendix D-E.

However, a subsequent upgrade to the ZED camera drivers and SDK (version 5.2.0) introduced robust error and timeout handling. These software improvements allowed us to safely bypass the ROS 2 environment entirely for orchestrating the camera capture. Consequently, we developed a highly efficient, automated periodic recording paradigm. Both the initial ROS 2-based procedure (utilized for the dataset presented in this thesis) and the advanced periodic recording method (developed for future operations) are described in detail below.

1) *Capture using ROS 2 and GUI*: SenseBike utilizes a Flask-based web application as a GUI. Aligning with this deployed software architecture, we developed a separate control module for the ZED container. This addition allows the operator to start, stop, and explicitly terminate all camera-related processes. The ZED container is based on the ROS 2 Humble image for L4T 35.4.1, which is fully compatible with our NVIDIA Jetson SBC. Because the official Stereolabs Docker image did not fully support our specific L4T version,

we customized the environment by manually compiling and installing the ZED SDK 5.0.5 alongside the ZED ROS 2 Wrapper.

The system startup sequence for ROS 2-based recording is illustrated in Figure 12. Upon powering up the SenseBike, the GNSS container initializes automatically; this exception is necessary because establishing a stable connection to the NTRIP caster requires significant lead time. Once all other peripheral electronic components (such as the cellular router and the GNSS+RTK board) are fully operational, the operator manually launches the Docker containers that start respective ROS nodes for each sensor set via the GUI. Because the ZED cameras necessitate a more complex initialization sequence, we developed a separate GUI section dedicated to camera controls, featuring an embedded terminal to monitor system logs in real time.

As detailed in Figure 13, launching the ZED container triggers a ROS 2 component container that manages the nodes for all four cameras. This architecture was implemented to strictly limit the communication overhead between the ZED nodes. To stabilize the startup procedure and prevent CPU spiking, the camera nodes are launched with a time offset, allowing computing resources to stabilize. The primary initialized camera is configured to publish the ROS 2 odometry and map transformations, utilizing the predefined extrinsic transformations between the sensors. Subsequently, an XACRO command generates the URDF file of the combined camera suite, enabling the robot state publisher to broadcast the complete transformation (tf) tree. Although this configuration successfully facilitated data capture at an SVGA resolution (1920x600 pixels, side-by-side stereo), the system still experienced frequent frame drops due to computational bottlenecks. Furthermore, to avoid system-level fatal errors inherent to the underlying SenseBike software stack, riders were strictly instructed to limit continuous recording sessions to 5–7 minutes.

While capturing the data using the ROS 2 approach, the rider is required to manually start and stop recordings through a service call to all ZED camera nodes as described in Figure 14. This manual operation inherently introduces stationary data artifacts at the beginning and end of each sequence, as the rider must be fully stopped to safely interact with the GUI. Because the service calls are utilized to start and end recordings, the cameras suffered from temporal sequence desynchronization of up to 4 seconds. To address this latency, the recorded video feeds are trimmed during post-processing to enforce a uniform temporal window across all camera streams. The detailed discussion is provided in Appendix D-A.

2) *Periodic recordings*: Leveraging the release of newer camera drivers and an upgraded ZED SDK, we developed an optimized methodology that resolves the primary limitations of the ROS 2 approach outlined in Appendix C-A1. By completely bypassing the GUI and ROS 2 environment for the vision system, this new architecture enables continuous data acquisition at an HD1200 resolution (3840x1200) without any systematic system failures. Furthermore, the operator is no longer required to manually start and stop recordings, as the data streams are automatically segmented every 5 minutes.

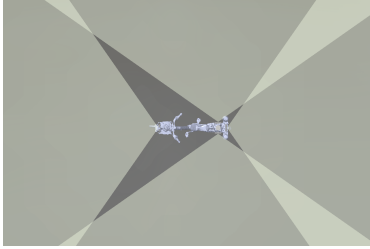


(a) Front cameras mounted on front rack.

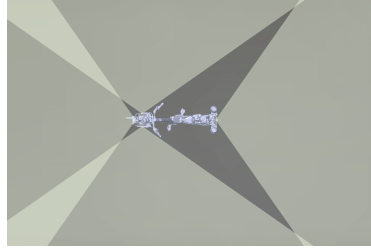


(b) Back camera mount on supporting beam.

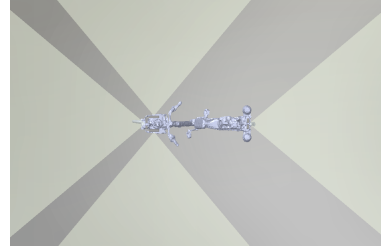
Fig. 10: Mounting points of ZED X cameras.



(a) Back mounted cameras with 110° FOV



(b) Front mounted cameras with 110° FOV



(c) Front mounted cameras with 80° FOV

Fig. 11: Proposed configurations of camera setup with visualized FOV in yellow.

The updated workflow requires the operator to initially launch the LIDAR sensors via the GUI. The recording procedure is then initiated by selecting a desktop icon (Figure 8), which executes a comprehensive bash script. This script configures the necessary environmental variables and provisions X server access for proper ZED container initialization. To ensure proper data capture, the script first verifies the availability of the LiDAR sensors. If the LiDAR containers are unresponsive, a Zenity-based GUI prompt warns the rider, offering the option to proceed and start recording without LiDAR sensors or to abort. Network connectivity is similarly verified to ensure the GNSS container can reliably reach the NTRIP Caster. Finally, a bash cleanup trap function is initialized to ensure safe shutdown handling, and a persistent Zenity "Stop" button is rendered on the screen to allow the operator to safely terminate logging at any time. In parallel, a subprocess is created to initialize the ZED container, while the main process governs other sensor recordings. Both of the processes can be interrupted by the Zenity "Stop" button.

As illustrated in Figure 15, the non-visual sensor data recording is triggered by the system's wall clock. Consequently, recordings are automatically segmented at 5-minute wall-clock intervals (e.g., 14:00, 14:05, 14:10), regardless of when the recording initially started. To ensure proper periodic operation, a non-visual sensor recording container (ROS 2 recording container) is stopped and restarted at every iteration. If two consecutive recording sessions are initiated within the

same 5-minute chunk, the script dynamically resolves any file naming conflicts. Due to this periodic sequence, the ROS bag logging typically begins 1–2 seconds after the exact wall-clock boundary.

Simultaneously, the ZED recording container initializes a standalone C++ recording executable (Figure 16). This script mirrors the 5-minute chunking logic and naming conflict resolution implemented in the ROS 2 recording container pipeline. Because the dataset strictly requires synchronized 360° coverage, the software performs a hardware check and immediately exits if any of the four cameras are unresponsive. Once verified, all cameras are configured to collect data with a resolution of HD1200 at 15 frames per second (fps). To preserve the SBC computing resources solely for data capture, the additional camera functionalities are disabled and raw SVO2 files are collected directly.

Each camera recording is managed within its own isolated thread. The executable fetches the current wall-clock chunk start time, creates a directory for the camera files, and assigns the SVO2 filenames. With no errors or user interruptions occurring, the C++ ZED recording script continuously performs frame grabs while monitoring for hardware faults. At every frame iteration, the system evaluates whether a 5-minute wall-clock boundary has been crossed. If a boundary is detected, the active data streams are seamlessly redirected to new files in the subsequent time directory. The redirection is executed almost instantaneously to prevent frame loss. This multi-threaded

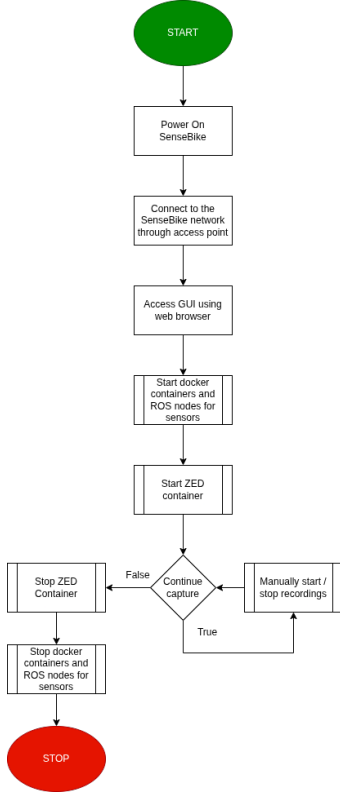


Fig. 12: Startup procedure using Flask App for manual start and stop recordings.

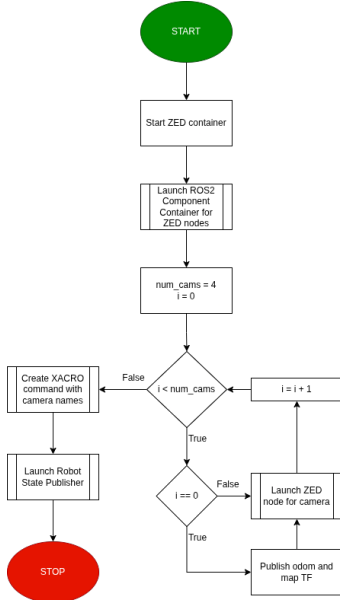


Fig. 13: ZED container startup in ROS 2 as in Figure 12.

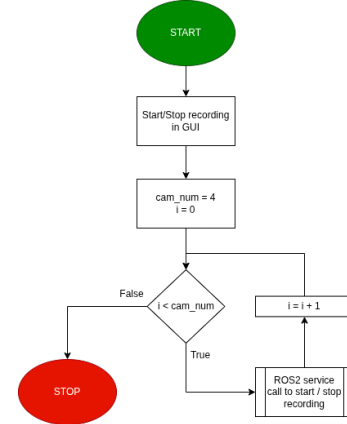


Fig. 14: Manual start/stop of the camera recordings in ROS 2 as in Figure 12.

procedure runs continuously until the operator triggers the Zenity "Stop" button or a fatal error occurs, which effectively stops all camera threads and closes the connections.

B. Captured data formats

Regardless of the recording procedure, the data is stored in exactly the same formats. All non-vision sensor streams – specifically the LiDAR point clouds and GNSS readings – are logged via the ROS 2 recording container. This data is serialized into the MCAP format for ROS 2 bags, utilizing the baseline data logging pipeline provided by the SenseBike manufacturer.

The camera data is saved in SVO2 files, a proprietary format developed by Stereolabs. An SVO2 file contains a raw video stream alongside embedded sensor data. While the images are captured at a frequency of 15 Hz, SVO2 enables the storage of raw IMU data at its native hardware frequency of 400 Hz. These files also embed all essential camera metadata, including intrinsic calibration parameters, hardware model, and resolution settings. Furthermore, the ZED SDK can read an SVO2 file exactly as if a live camera was physically connected, which proves useful in further post-processing tasks.

The visual resolution depends on the specific capture pipeline. Data collected using the initial ROS 2 approach (Appendix C-A1) is limited to an SVGA resolution (1920x600 pixels) at 15 frames per second (fps). In contrast, the optimized periodic recording framework allows for data capture at a superior HD1200 resolution (3840x1200 pixels) while maintaining the same 15 fps rate. To efficiently store this high-bandwidth data, all raw imagery are compressed using H.265 (HEVC).

C. Capture

Because the bicycle infrastructure is well developed in the city of Delft, we aimed to collect data mainly from dedicated bicycle paths, as well as paths in mixed traffic where bicycles share the road with cars. To ensure the capture of high-quality,

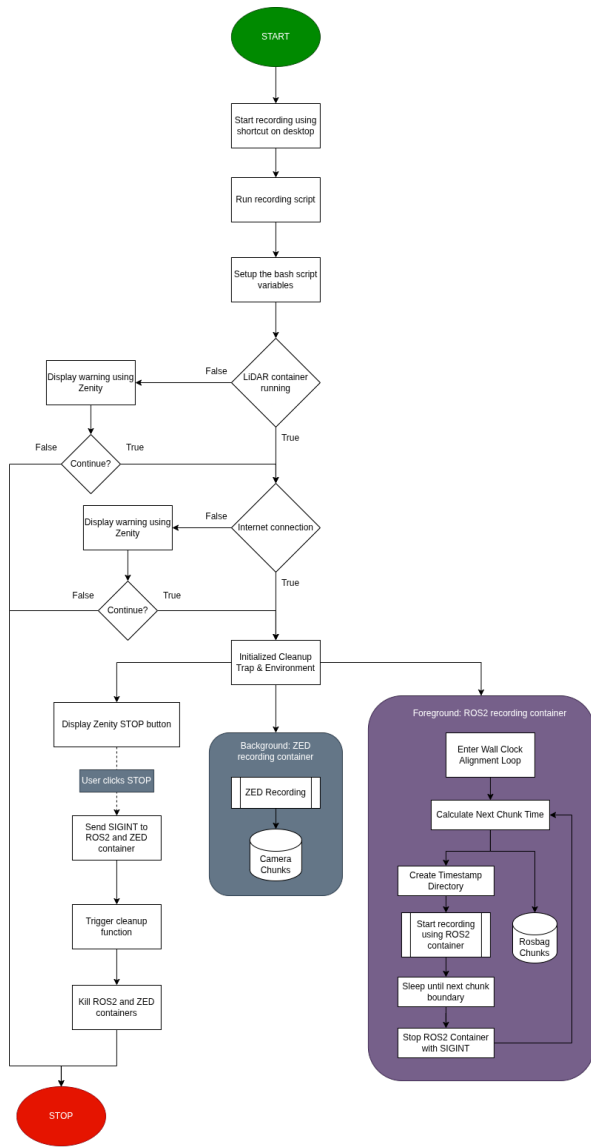


Fig. 15: Periodic recording workflow.

safe ground-truth trajectories, the designated riders were instructed to strictly adhere to all traffic regulations and exhibit highly cautious riding behavior, actively avoiding any reckless or aggressive maneuvers. This capture protocol was enforced to capture the safest driving behavior possible. For example, when approaching an intersection during light transition, riders were mandated to stop safely rather than attempting to cross on a yellow signal. Additionally, we encouraged operators to take as many turns as possible to ensure that the capture data contains turning events.

The raw sensor data utilized in this thesis was recorded over two consecutive days: November 12 and 13, 2025. The taken routes are shown in Figure 2. To capture varying levels of congestion, certain high-traffic trajectories, such as the bicycle paths routing through the TU Delft campus, have been taken multiple times to log different congestion scenarios during peak and off-peak hours. Furthermore, we have deliberately taken routes both around the city center of Delft, in the

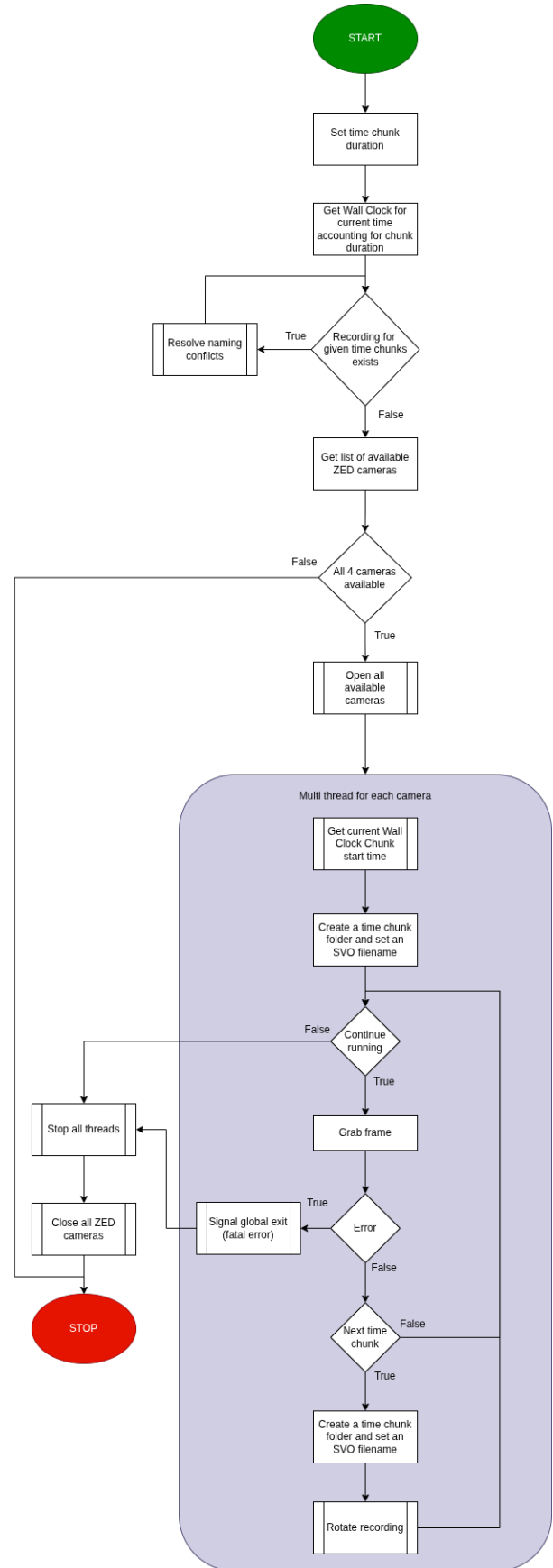


Fig. 16: ZED recording workflow from Figure 15.

suburbs, and around the TU Delft Campus to ensure a greater diversity of captured environments. These two days of recording yielded approximately 20 hours of raw data. We collected point clouds from all LiDAR sensors, GNSS+RTK data, and camera raw feed, including IMU readings from each camera. Because the SenseBike is not fully waterproof, all data capture was strictly performed in dry, rainless conditions.

APPENDIX D DATA CURATION

Although all sensors are time synchronized, the raw captured data requires extensive post-processing to yield a valid dataset for E2E path planning. The inherent limitations of the manual ROS 2 capture approach (discussed in Appendix C-A1) necessitated rigorous post-processing of the camera files.

The data curation pipeline involves aligning camera sequences to common time bounds across all camera files from the same recording, synchronizing camera sequences, mitigating dropped frames caused by computational bottlenecks, and trimming stationary periods from the recordings. Subsequently, turn events are identified using IMU data, the global pose of the SenseBike is estimated, and the continuous recordings are partitioned into 20-second scenes, adopting the methodology of nuScenes [9]. Figure 18 illustrates the complete dataset curation workflow.

A. Camera files preprocessing

Because the manual ROS 2 capture procedure initializes and terminates camera nodes sequentially (Appendix C-A1), the resulting SVO2 files for a single recording exhibit varying temporal lengths. To establish a uniform and overlapping temporal window across all cameras, we developed a parallel preprocessing workflow (Figure 19) that extracts the sequences' metadata and further cuts the SVO2 files accordingly.

This temporal alignment requires identifying a common start and end timestamp across all SVO2 files for a given recording (detailed in Figure 21). Because the GMSL2 protocol used to transmit camera data is serialized, timestamps for the corresponding synchronized frames are not identical across all devices. Consequently, the common start timestamp is defined as the maximum of the first frame timestamps across all cameras, while the common end timestamp is defined as the minimum of the last frame timestamps. To ensure stability, a 1-second offset is subtracted from the end timestamp as some of the SVO2 files exhibit corrupted frames at the very end.

Following timestamp identification, the specific frame indices corresponding to these temporal boundaries are extracted for each camera. The temporally aligned start frame index is designated as the first valid frame index for which the timestamp is greater than the common timestamp. Similarly, the temporally aligned end frame index is selected. Utilizing these indices, the SVO2 files are accurately trimmed via Stereolabs' proprietary software, guaranteeing synchronized multi-camera data lengths for downstream processing.

B. Frame information extraction

Following the temporal alignment of the multi-camera SVO2 files, frame-level metadata is extracted to facilitate subsequent frame drop diagnostics. The extraction pipeline isolates three key parameters from each camera feed: the absolute timestamp, the captured frame index, and the expected frame index.

The captured frame index serves as a strictly incremental counter of the frames successfully recorded and available within the file. Conversely, the expected frame index represents the theoretical frame count based on the hardware grab period, thus explicitly accounting for dropped frames and allowing for frame drops identification. Because extracting this metadata requires the ZED SDK to open each SVO2 file, the process is parallelized in 8 subprocesses (the maximal number ensuring stable ZED SDK operation). The data was then serialized into CSV file. The architectural workflow of this process is depicted in Figure 22.

C. Camera sequence synchronization

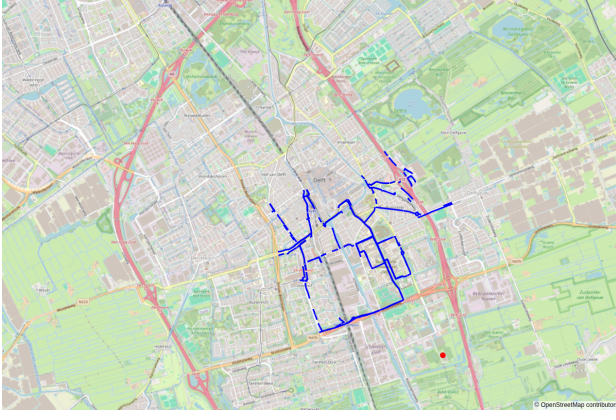
The temporal trimming of SVO2 files inherently introduces minor frame-level misalignments across the multi-camera streams because that workflow is sequence blind. Therefore, a secondary re-synchronization algorithm was developed and is implemented to align the metadata sequences. The algorithm is based on the manufacturer's specifications that frames across all cameras are considered temporally synchronized if their capture timestamps fall within a single hardware grab period, which corresponds to 0.066 seconds at a 15 Hz capture rate.

The algorithms require a reference sequence – front-facing camera by default – to which the remaining three cameras (target sequences) are aligned. As the initial trimming procedure (Appendix D-A) guarantees close temporal proximity, the alignment search space is constrained to a discrete shift range of $s \in [-5, 5]$ frames. For each candidate shift s , the target sequence is offset, and the element-wise timestamp differences between the target and reference frames are calculated. To prevent sporadic timing anomalies caused by the malfunction of SBC or ZED SDK, these differences are filtered using the Interquartile Range (IQR) method to remove outliers before calculating the mean temporal difference.

The shift s that yields the minimum mean difference is selected as the optimal alignment, provided this mean value is strictly below the 0.066-second grab period threshold. This optimal shift is then stored and applied directly to the target sequence to temporally align it with the reference. If no shift within the $[-5, 5]$ range satisfies the grab period constraint for a given sequence, the reference sequence is advanced by one frame, and the search iteration repeats for all target sequences.

D. Frame drop analysis

During data acquisition, the almost full utilization of the SBC computational resources occasionally led to system unresponsiveness, resulting in dropped camera frames. To investigate and quantify this data loss, a diagnostic procedure was implemented to detect missing frames and estimate the frequency of simultaneous drop events across all cameras. The



(a) Recording date: 12.11.2025



(b) Recording date: 13.11.2025

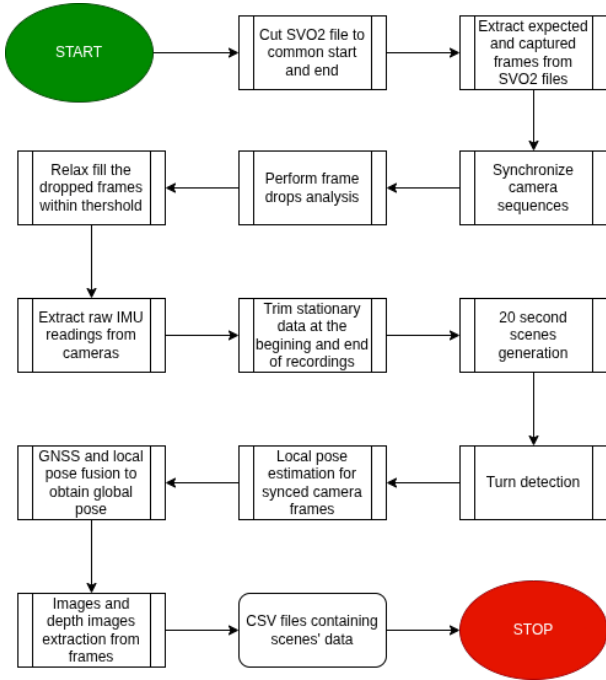
Fig. 17: Routes taken per recording date, marked in **blue**, with the GNSS RTK base station marked in **red**.

Fig. 18: Workflow for camera dataset curation.

percentage of the corrupted synced instances per recording day is provided in Table IX. The results did not exhibit any traces of single systematic sensor failures – all cameras dropped frames randomly. An SBC process optimization, specifically disabling the active localization module, led to a significant reduction in frame drop rates during the second recording session (13.11.2025).

The result of this diagnostic phase is a frame-synchronized dataframe containing the frame indices and capture timestamps for all cameras, with dropped frames explicitly represented as null entries. The computational bottlenecks responsible for these anomalies were fully mitigated in the subsequent deployment of the periodic recording paradigm (Appendix C-A2) alongside updates to the ZED SDK and underlying camera drivers.

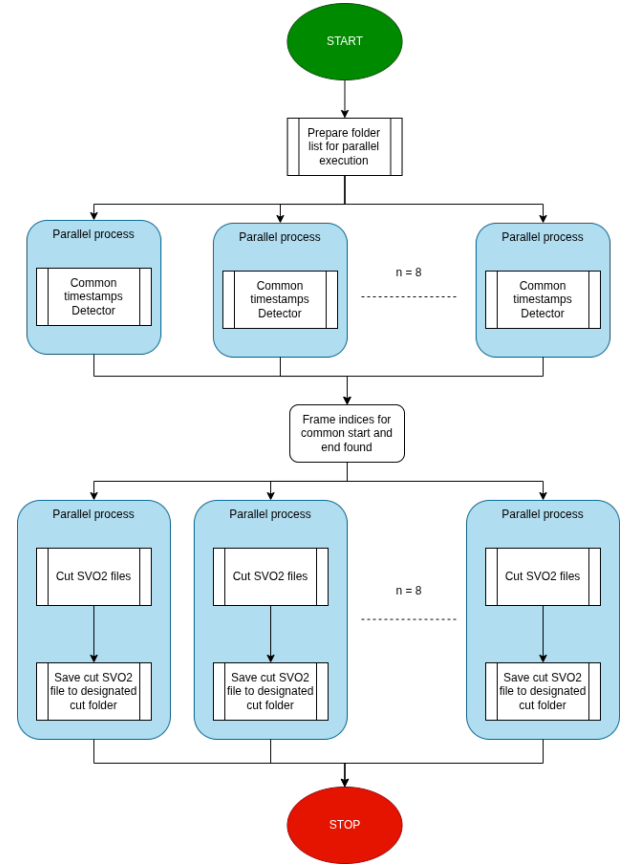


Fig. 19: Process of raw camera files temporal alignment.

E. Relax fill for dropped frames

To mitigate the impact of dropped frames and maintain temporal continuity across the multi-camera sequences, a relaxed fill algorithm for dropped frames is developed and introduced in the data curation pipeline. As the camera data is captured at 15 Hz, missing frames are replaced with a copy of the closest frame, typically 66.67 ms apart, provided that the temporal gap does not exceed a 200-ms threshold and at least one camera successfully captured a valid frame at that

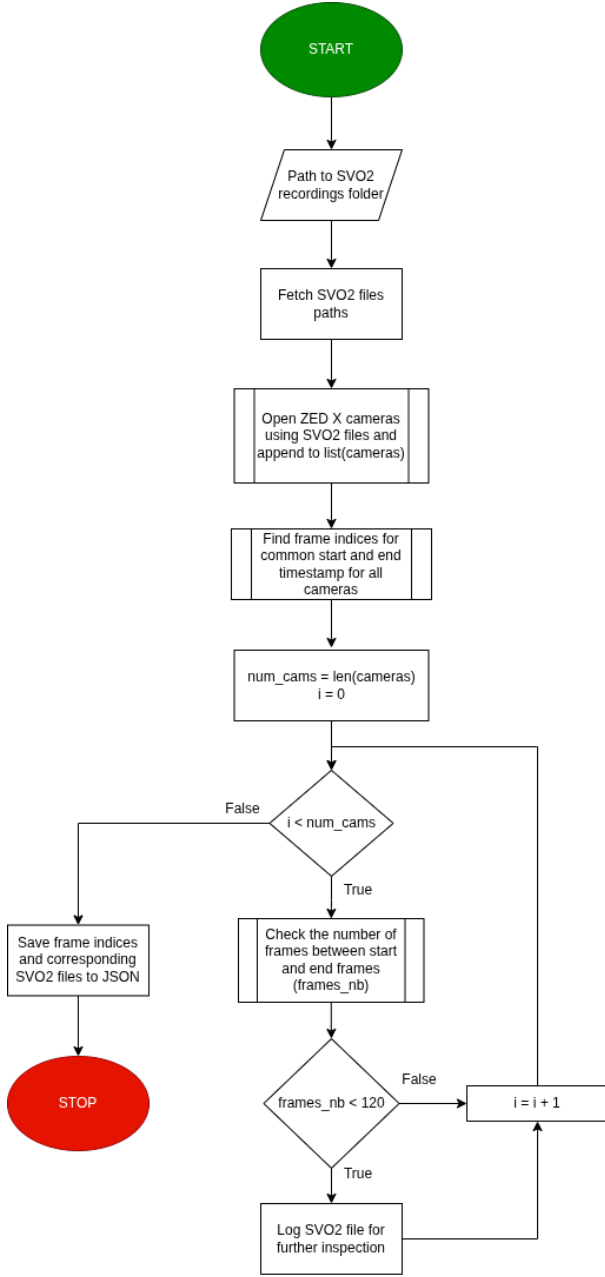


Fig. 20: Common timestamp Detector workflow form Figure 19.

Date	0 cams	1 cam	2 cams	3 cams	4 cams
12.11.2025	77.20%	8.25%	6.71%	6.29%	1.47%
13.11.2025	94.02%	2.4%	1.65%	1.74%	0.19%

TABLE IX: Percentage of simultaneous frame drops per number of devices.

specific time instant.

To prevent compounding errors, a strict constraint is enforced to only allow originally valid frames to be relaxed filled (previously relaxed filled frames cannot be used). Furthermore, to prevent data leakage and ensure data integrity, the relaxed filled frames are flagged and further excluded from the training splits.

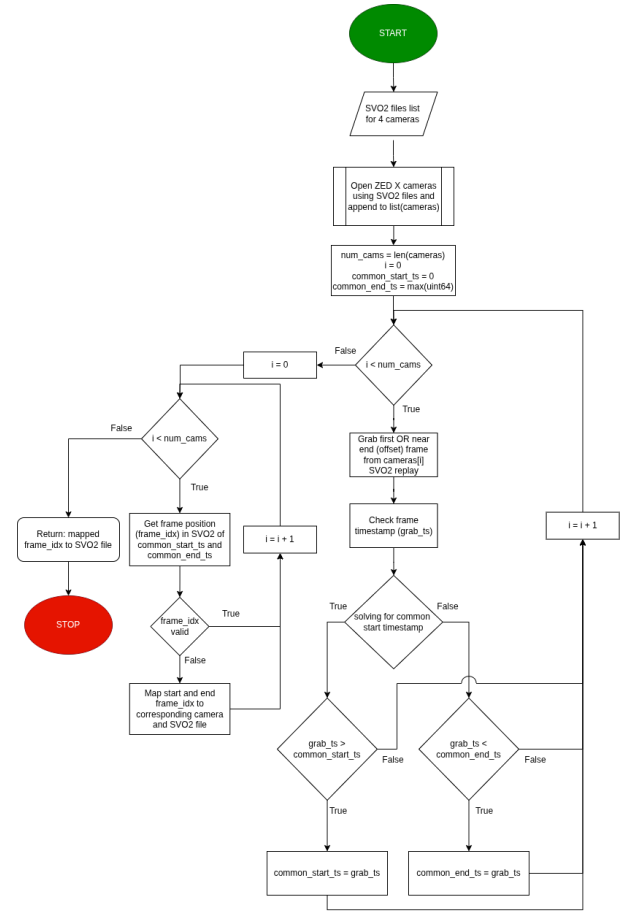


Fig. 21: Detailed workflow of finding start and end frame indices described in Figure 20.

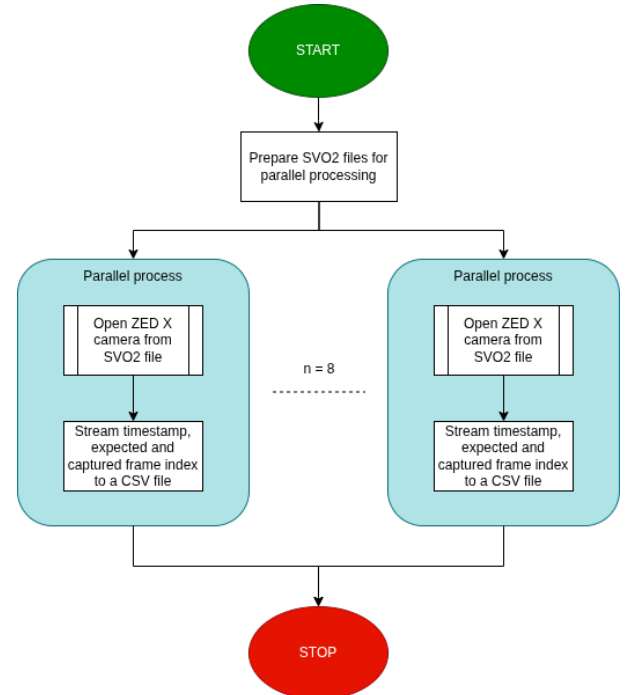


Fig. 22: Frame information extraction from SVO2 files.

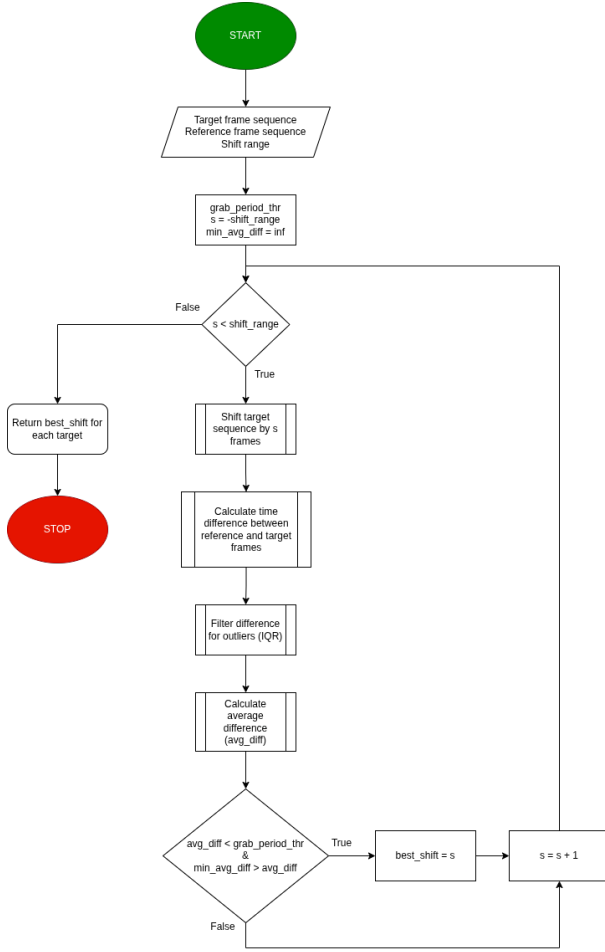


Fig. 23: Frame shift algorithm to perform sequence synchronization.

F. Stationary periods detection

The recording procedure using SenseBikes’s GUI and ROS 2, outlined in Appendix C-A1, requires starting and stopping the capture while stationary. This inherently introduces stationary data artifacts at the beginning and end of each recording.

To automatically detect and trim these periods, a kinematic filtering algorithm is applied utilizing the raw IMU data from the front-facing camera, as shown in Figure 24. IMU measurements are extracted from the SVO2 files, and the Z-axis acceleration is compensated for gravity. The high-frequency noise is filtered using a 10 Hz low-pass filter. Following this, the angular velocities in the Y and Z axes, along with the gravity-compensated Z-axis acceleration, are extracted.

To quantify movement intensity, the Root Mean Square (RMS) is calculated for each of these three signals over a rolling window of 0.8 seconds. Rather than enforcing a rigid kinematic threshold, a dynamic baseline is established by multiplying the sequence’s minimum RMS Z-axis acceleration by a factor of 1.5. A given timestamp is initially assigned a *raw* riding status if its acceleration exceeds this dynamic threshold and its angular velocity (in either the Y or Z direction) exceeds a predefined threshold of 1 deg/s.

To confirm sustained movement, a rolling sum is applied. A

sample is officially classified as *riding* only if the density of raw riding samples meets the minimum required riding window of 5 seconds. Finally, boundary timestamps are identified for the first and last continuous riding segments, and the frame-synched dataframe is subsequently trimmed to these boundaries, successfully discarding the initial and final stationary periods.

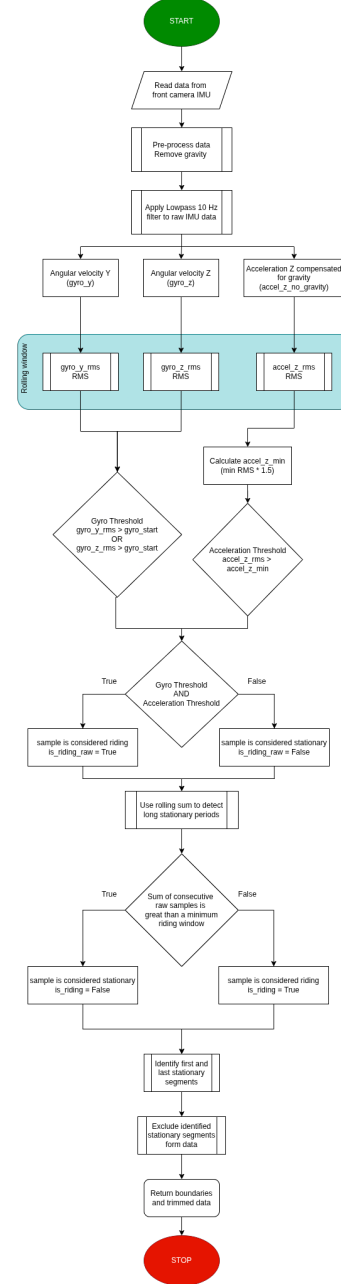


Fig. 24: Riding segments detection using raw IMU data.

G. Scenes generation and turn detection

Following the methodology of nuScenes [9], the data is organized into 20-second scenes. The scene generation process is based on the frame-synchronized dataframe, which has been pre-filtered to exclude stationary boundary periods. To ensure a consistent spatial reference, the dataframe is first strictly

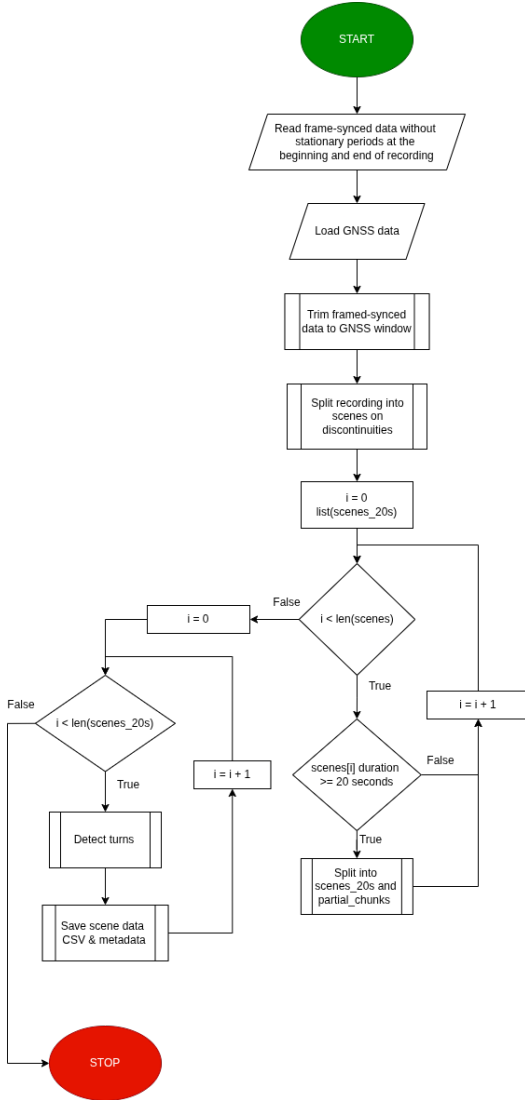


Fig. 25: Block diagram of generating 20-second scenes.

trimmed to the temporal window where both continuous GNSS data and the synchronized frames are available. The process is shown in Figure 25.

To split the recordings into scenes, a sequential method is utilized. Initially, temporal discontinuities within the reference camera’s timestamps are identified. These discontinuities are caused by the simultaneous malfunction of all cameras, resulting in timestamp gaps between consecutive synced frame instances larger than the standard grab period. The recording is initially divided at these boundaries into variable-length contiguous segments. The 20-second scenes are subsequently extracted from these variable-length sequences utilizing a greedy window approach. The residual final chunk of each segment that is shorter than 20 seconds is discarded to enforce uniform scene durations.

To perform a scene-level turn detection, the Z-axis angular velocity (gyro z) is utilized and pre-processed with a 10 Hz low-pass filter. We empirically set the turning threshold thr to 25 deg/s after manually reviewing the dataset. A scene is classified as containing a right turn if any angular velocity

sample exceeds this positive threshold. Conversely, if the sample falls below the negative threshold ($-thr$), the scene is classified as containing the left turn. Based on this approach, a scene can contain left, right, and both turns at the same time. Finally, the straight path segments are identified by validating that at least one sample falls within the range $[-thr, thr]$.

H. Local pose estimation

Local pose estimation for the SenseBike is performed using the Positional Tracking module of the ZED SDK during post-processing. This module provides a six-degree-of-freedom pose, comprising spatial translation (X, Y, Z) and rotation (roll, pitch, yaw, and quaternions). It uses Visual-Inertial Odometry (VIO) to integrate Stereo Visual Odometry (VO) and IMU data. The Stereo Visual Odometry estimates the camera’s motion between consecutive frames by tracking visual features, while the IMU provides acceleration and angular velocity data at a frequency of 400 Hz. By utilizing VIO, the significant drift inherent to the camera IMU hardware is corrected through VO yielding a stable and accurate local trajectory.

VIO pose is extracted from the rear-facing camera. This sensor is physically mounted closer to both the GNSS antenna and the bicycle’s true center of rotation during cornering maneuvers. Additionally, the trajectory analysis indicated that the front-facing camera produced an unstable path estimation. The absence of SenseBike’s active pedaling assist required compensatory balancing movements from the rider, which introduced an oscillatory steering pattern and artificially degraded the trajectory’s overall smoothness.

Consequently, the post-processing pipeline extracts the full pose estimation solely from the rear camera. The recorded SVO2 files are processed by custom-developed parallel scripts to efficiently extract the pose information into CSV format. This extracted data serves as the foundation for the subsequent global pose estimation.

I. Global pose estimation

The Sensebike’s global pose estimation consists of five main stages. First, the GNSS data and the local VIO poses are temporally synchronized in the pre-processing stage. Next, an initial transformation to align the VIO local coordinates (X and Y translations) with the global GNSS data is calculated and applied. Subsequently, the VIO data is validated in terms of global heading and corrected for corrupted samples. Next, Factor Graph Optimization (FGO) is used to fuse the VIO trajectory with the GNSS anchor points. Finally, the data is postprocessed to yield a globally aligned 2D pose. This procedure is illustrated in Figure 26 and detailed in the following subsections.

1) *GNSS and VIO fusion*: The GNSS data, which includes Real-Time Kinematic (RTK) corrections, is extracted from a ROS bag containing the raw sensor readings and pre-processed alongside the VIO data from the rear-facing camera. To align reference frames, the rear-facing camera coordinate system is transformed to a forward-facing orientation by π rotation about the Z axis. To ensure the optimization graph is constrained by high-accuracy GNSS data, we apply a strict position-variance

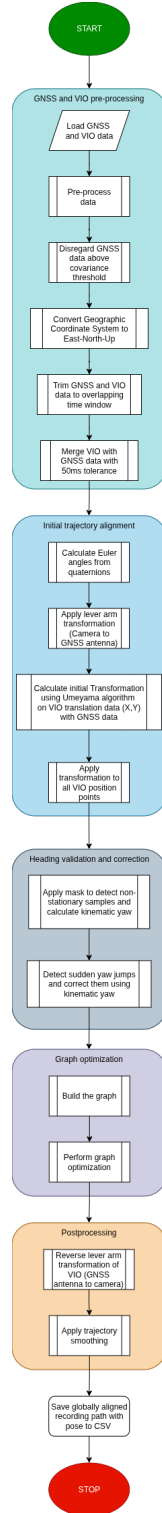


Fig. 26: Global pose estimation workflow.

threshold of $0.05m^2$ in the X and Y directions. Any GNSS sample exceeding this uncertainty is rejected.

Because the raw GNSS readings are stored in a Geographic Coordinate System (GCS) using angular units, they are not directly suitable for metric path planning or alignment. Consequently, the data is projected into a Cartesian East-North-Up (ENU) coordinate frame, measured in meters. The

DLF100NLD GNSS reference station in Delft serves as the local origin for this ENU projection (Figure 2).

Strict temporal overlap between the GNSS and VIO datasets is enforced by cropping both streams to their shared time boundaries. The GNSS records are subsequently merged into the VIO data using a maximum temporal tolerance of 50 ms. In this process, the VIO data acts as the primary time series, while the GNSS data serves as supplementary observations. Consequently, GNSS readings without a corresponding VIO frame within the tolerance window are not matched to any VIO sample. This logic is particularly crucial because the two data streams are sampled at different frequencies: VIO at 15 Hz and GNSS at 10 Hz. The fused dataset is subsequently utilized in the optimization stages.

2) *Initial path alignment*: To guarantee the mathematical stability and convergence of the FGO stage, an accurate initial alignment between two VIO and GNSS trajectories is required. Because the VIO trajectory natively originates at an arbitrary local origin (0,0), it must be geometrically aligned with the global ENU coordinate frame before optimization.

First, the VIO trajectory points are shifted to GNSS antenna coordinates using predefined extrinsic transformations between the sensors. This step is critical to ensure that both the VIO and GNSS data streams describe the spatial trajectory of the identical physical point on the ego-vehicle. Subsequently, with this shared reference, the Umeyama algorithm [25] is applied to the subset of fused GNSS and VIO samples that contain GNSS samples. This yields an optimal rotation, translation, and scale matrix between VIO local waypoints and GNSS global data by minimizing the Root Mean Square Deviation between these two point sets. The transformation is then applied to the entire VIO trajectory to achieve an initial globally aligned two points sets. The Umeyama approach extends the foundational Kabsch algorithm [26] by not only solving for rotation and translation, but also calculating an optimal uniform scale factor and mathematically preventing invalid reflection matrices, making it highly robust for this trajectory matching problem.

3) *Heading validation and correction*: Factor Graph Optimization relies on continuous and physically valid yaw constraints. Since the SenseBike lacks rear-axle suspension, the camera-mounted IMU experiences high vibrations, occasionally causing the VIO to output unrealistic yaw values for individual frames. To address this, a kinematic heading correction algorithm is implemented within the pipeline. Because kinematic yaw derivation requires non-stationary samples, a moving mask is applied to isolate non-stationary samples, enabling the mathematical calculation of a continuous kinematic yaw. The kinematic yaw correction algorithm is based on a strict physical constraint that the instantaneous heading variations exceeding 40° between consecutive 15 Hz frames (equivalent to 600 deg/s) are dynamically impossible. The iterative loop scans the data for invalid yaw samples and corrects them with the kinematically estimated yaw. This iterative architecture is specifically required to ensure the detection and correction of consecutive invalid samples. The number of correction iterations is capped at 100 under the assumption that sustained data corruption exceeding this threshold indicates

a sensor failure rather than corruption, and such scenes are subsequently discarded during the visual inspection phase.

4) *Graph optimization*: Factor Graph Optimization (FGO), implemented via the GTSAM library [27], is used to fuse the Visual-Inertial Odometry (VIO) and GNSS data. Unlike traditional filtering methods (such as the Extended Kalman Filter), FGO functions as a full-trajectory smoother by retaining the complete history of states. This allows all past states to be continuously re-evaluated and corrected as new absolute GNSS measurements are recorded, effectively mitigating accumulated non-linear VIO drift. The implemented graph formulation contains nodes represented as a 2D state of the SenseBike at each timestamp t , comprising X and Y coordinates in ENU coordinate frame along the globally aligned heading. These nodes are constrained by two primary factors: odometry factors and GNSS prior factors.

Odometry factors: The relative spatial transformations derived from the VIO trajectory define the odometry constraints connecting consecutive nodes between time $t - 1$ and t . The odometry noise (standard deviation) is explicitly parametrized to 0.15 meters for spatial translation and 0.05 radians for rotation. This configuration permits the optimizer to tolerate significant drift between frames, which addresses the standard VIO drift while heavily relying on the VIO heading.

Prior factors: The RTK-corrected GNSS coordinates serve as absolute global anchors (prior factors). Because GNSS readings provide highly accurate spatial positioning but lack heading information, a highly asymmetric noise model is applied. The X and Y sensor noise is dynamically initialized using the logged GNSS sensor uncertainty. A strict minimum standard deviation of 0.05 meters is enforced to prevent the graph from over-constraining to potential anomalies in GNSS readings. Since the global positioning data lacks yaw information, it is initialized with the unprocessed VIO yaw and assigned the noise standard deviation of 100 radians. This mathematically decouples the heading from GNSS measurements, instructing GTSAM to rely entirely on the VIO yaw readings while firmly pulling the spatial trajectory toward the global GNSS coordinates.

Non-linear optimizers, such as FGO, require an accurate initial guess to prevent the system from diverging or becoming stuck in local minima. To mitigate this, the VIO local X and Y positions, along with the yaw readings, are previously transformed using the Umeyama algorithm. This preliminary geometric alignment provides a robust initial state estimate, ensuring the final optimization is both stable and computationally efficient.

Finally, the Levenberg-Marquardt optimizer is utilized to solve the constructed graph. The underlying mathematics minimizes a non-linear least-squares cost function, iteratively adjusting the entire trajectory to find the optimal path that best satisfies all the defined odometry constraints and GNSS anchors. As a result, a globally optimized 2D pose trajectory is extracted, yielding precise ENU coordinates and a corrected global heading for the SenseBike.

5) *Post processing*: Following the FGO, the optimized trajectory of the GNSS antenna is extracted. A reverse lever-arm transformation is then applied to mathematically shift the

spatial reference point from the antenna back to the optical center of the rear-facing camera. Finally, to mitigate seldom-present trajectory jitter, a Savitzky-Golay filter is applied to the X and Y coordinates alongside the unwrapped yaw array. To prevent oversmoothing and cutting corners, a strictly small window size of 6 frames (0.4 seconds) is used. The fused and smoothed global pose trajectories are subsequently saved in CSV format.

6) *Graph optimization vs other fusion methods*: Mostly, sensor fusion for localization relies on recursive filtering techniques, such as the Kalman Filter and all its derivatives. Utilizing, for instance, the Extended Kalman Filter (EKF) is computationally lightweight as it operates under the Markov assumption that it only estimates the current state based on the immediately preceding state and the current measurement. Once the state is estimated, it is permanently fixed and therefore cannot be optimized in the future. This may cause problems as our VIO system is prone to accumulating drift over time. If SenseBike enters an area of low GNSS coverage, the VIO drift may cause the fused trajectory to deviate from the actual path. Once the GNSS RTK corrections are restored and the accuracy increases, the path may snap back to that new valid GNSS sample. Conversely, the Factor Graph Optimization (FGO) framework maintains a complete history of past and current states within the graph. When a high-confidence GNSS measurement arrives after a period of signal loss, GTSAM does not merely correct the current position. Instead, it propagates the correction backward through the graph. By treating the VIO measurements as relative constraints (springs) between states, the FGO mathematically smooths out the accumulated VIO drift over the entire trajectory connecting the GNSS anchor points, resulting in a continuous, physically realistic path.

The other limitation of filtering methods, such as EKF, is their handling of non-linear equations of motion. To accommodate non-linearities, the EKF calculates a linear approximation exactly once around the current state estimate. If this estimate is inaccurate due to odometry drift or sudden dynamic movements of the bicycle, the linearization is fundamentally flawed, and the resulting error becomes permanently embedded in the system. In the BikeScenes-planning dataset, riding the SenseBike through regions of high GNSS uncertainty using an EKF could yield highly degraded trajectories due to this compounding linearization error. Conversely, FGO treats the problem as a non-linear least-squares optimization. Solvers such as the Levenberg-Marquardt algorithm, which we utilized in this pipeline, iteratively update the state and perform re-linearization around the new, improved estimate at every step. This iterative refinement makes the FGO significantly more robust to large initial errors and the highly dynamic, non-linear rotational movements characteristic of a bicycle.

Furthermore, the sensor fusion pipeline for the BikeScenes-planning is performed entirely offline, as SenseBike does not have sufficient computational resources to employ real-time fusion during the capture. As offline processing gives access to complete trajectories (both historical and future samples), Factor Graph Optimization was selected as the best candidate for the global pose estimation.

7) *Roll bias correction:* Before merging the global pose with scene data, roll bias is mitigated, which is typically caused by minor global coordinate frame misalignment during camera initialization or sudden high-magnitude vibrations (e.g., pavement impacts). To prevent vehicle turning dynamics from improperly influencing the bias estimation, the raw roll samples recorded while moving in a straight line are isolated, effectively excluding stationary and turning samples from the bias calculations. These isolated samples are further interpolated through the turning events to construct a continuous bias. To filter out the transient kinematic noise from the road surface, a 90-second rolling window mean is applied to the continuous bias values. Finally, this smoothed rolling mean is subtracted from the raw roll array, yielding the corrected ego-vehicle roll, which is subsequently utilized in the final global pose and scene fusion pipeline.

8) *Global pose fusion into 20-second scenes*: The globally aligned 2D pose from FGO for the continuous recording is further merged with 20-second scenes discussed in Appendix D-G. As 20-second scenes are represented in a dataframe format, each temporal row is precisely synchronized with the corresponding global pose sample utilizing the hardware timestamp. Because both data sources' timestamps originate from the same device, the perfect temporal alignment is achieved by timestamp matching. To safeguard data integrity, a final post-processing validation step is implemented, which checks if the captured frame indices between the matched scene and pose samples are strictly identical.

J. Frame extraction

The captured frame index is initially extracted during the camera file diagnostics, as mentioned in Appendix D-B. This index is then preserved throughout the dataset curation process and, therefore, present in the fully fused 20-second scenes. These frame indices are subsequently used to extract the PNG images from the left lens of the ZED X stereo camera using the raw SVO2 files. In addition to the standard RGB PNG images, 16-bit depth maps are simultaneously extracted and saved in PNG format.

First, all target captured frame indices are extracted from 20-second-long scenes for a given recording across all available cameras and saved in CSV files. These files are used in a custom-developed parallel frame extraction script shown in Figure 27. The script loads the target CSV files alongside the corresponding raw SVO2 files and then iterates through the SVO2 video stream and conditionally extracts only the frames matching the target indices. By extracting the frames as the very last step of the data curation process, the computational and storage resources are preserved and utilized for only the required frame PNG files.

K. Visual inspection

All 20-second scenes underwent visual inspection. This evaluation specifically targeted three main factors: insufficient GNSS coverage, non-kinematic spatial jumps within the trajectory, and failures in FGO optimization in terms of incorrect path or heading. Any scene exhibiting at least one

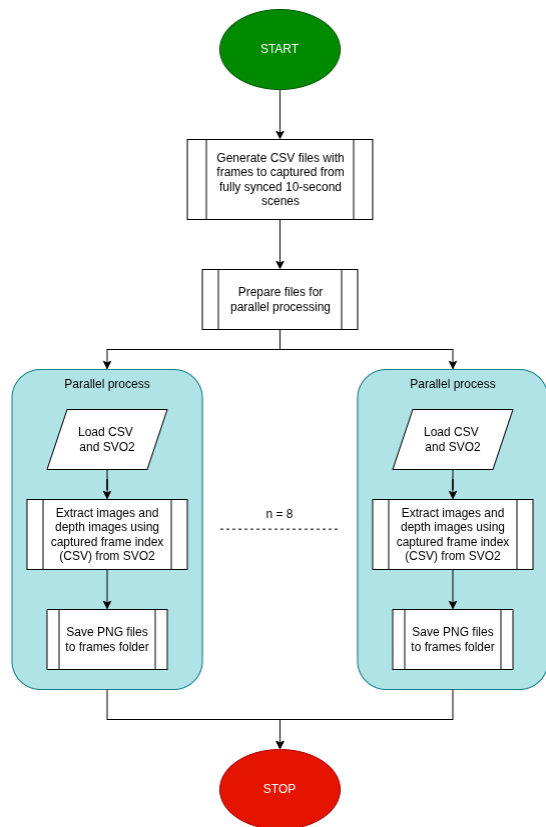


Fig. 27: Frames and depth images extraction from SVO2 files.

Date	Total scenes	Excluded scenes	Valid scenes
12.11.2025	438	47	391
13.11.2025	568	70	498
Both	1006	117	889

TABLE X: Scenes statistics per recording date.

of these anomalies was manually excluded from the dataset. The statistics for both recording dates are shown in Table X.

L. Dataset structure and information

The curated dataset is organized in 20-second scenes exported to CSV format and indexed by recording session. The dataset comprises various scenarios captured around the city of Delft as described in Appendix C-C. The scenes were classified based on cornering presence as described in Appendix D-G.

Every frame in a scene contains fused information about the synchronized camera frames, Visual-inertial Odometry, and globally estimated pose in the ENU coordinate system. Therefore, we store a set of the following information for every camera:

- timestamps
- relative file paths to camera images
- frame index captured
- frame index expected
- information about a relaxed filled frame
- reference camera information

Date	Only Left	Only Right	Left & Right	No turns
12.11.25	48 (12.28%)	30 (7.67%)	26 (6.65%)	287 (73.4%)
13.11.25	39 (7.83%)	47 (9.44%)	43 (8.63%)	369 (74.10%)
Both	87 (9.79%)	77 (8.66%)	69 (7.76%)	656 (73.79%)

TABLE XI: Number of scenes per turning event.

Additionally to the camera-specific information, we store VIO data and global pose in the following form:

- VIO linear velocities
- VIO angular velocities
- global 2D pose in ENU coordinate system (X, Y, and heading)

APPENDIX E

BIKEscenes - PLANNING DATASET

The synchronized 20-second scenes containing data, detailed in Appendix D-L, serve as the foundation for the *BikeScenes-planning* dataset. A custom PyTorch *BikeScenes* class was developed to fetch and parse the data from the scenes' CSV files. The dataset is time-based, meaning that parameters governing sequence lengths are defined in continuous time (seconds) rather than discrete sample counts.

A. Training data temporal filtering

The *BikeScenes* class operates on a temporal basis in seconds. It governs the amount of data available for training through the combination of temporal parameters. Using the provided dt (time step), $t_{horizon}$ (seconds in the future), $t_{history}$ (seconds in the past), $goal_seconds$ (goal offset from last valid sample), and fixed camera frequencies (15 Hz), the pipeline constructs relative time offsets for both historical states and future predictive horizon. This means that the available data will be trimmed to fulfill the past sample requirements by $t_{history}$ ($t_{history}$). Respectively, the end of the scene will be cut by $t_{horizon} + goal_seconds$ ($t_{horizon} + t_{goal}$). Fixing the offset to the goal is critical to prevent the E2E path planners from performing better while closer to the final goal waypoint. Additionally, the *BikeScenes* skips the relaxed filled frames ($\mathcal{R}$) captured using the front camera as described in appendix D-E. Therefore, the set of all valid sampled timestamps within a scene, X_{valid} , is formally defined as:

$$X_{valid} = \{x \mid x \in [t_{history}, t_{scene\ end} - (t_{horizon} + t_{goal})] \wedge x \notin \mathcal{R}\} \quad (2)$$

B. Waypoints

A critical component of the *BikeScenes-planning* dataset is the extraction and formulation of waypoints. Because the underlying raw data is globally aligned, a geometric transformation is applied to project the data from the global coordinate frame into ego-centric waypoints. The function takes a current ego position as a reference frame and transforms all past and future waypoints, sampled at an offset of dt , into this ego-centric coordinate frame. This step is strictly necessary, as the

E2E path planners require a relative, ego-centric coordinate frame.

A 7-dimensional waypoint vector is derived from the future poses in ego-centric frame which are constructed using the geometric transformation shown in Equation 3, where $T_{current}$ is the current transformation of the ego vehicle with respect to the world, T_{target} is the future ego-pose to be transformed in the world coordinate frame, and $T_{ego-future}$ is the resulting ego-pose at the target timestamp in the ego-centric frame.

$$T_{ego-future} = T_{current}^{-1} T_{target} \quad (3)$$

From the derived $T_{ego-future}$, the X and Y spatial coordinates, local yaw, and forward linear velocity are extracted. Furthermore, the pre-processed roll is directly extracted from the target row in the dataframe. Due to numerical discontinuity at the boundaries of a full circle, roll and yaw angles are embedded into $\sin$ and $\cos$ continuous representations to allow for proper ML model learning.

Given the above transformations and embeddings, the waypoints are represented as 7-dimensional vectors shown in Equation 4.

$$wp = \begin{bmatrix} x_{ego} \\ y_{ego} \\ \sin(roll) \\ \cos(roll) \\ \sin(yaw) \\ \cos(yaw) \\ \text{Forward linear velocity} \end{bmatrix} \quad (4)$$

These ego-centric waypoints serve as the primary context vectors during model training. The exact quantity of historical and future waypoints depends on the time horizon dataset parameters, $t_{history}$ and $t_{horizon}$, as well as the timestep dt . Therefore, the total waypoint count is calculated as follows:

$$\begin{aligned} count_{wp\ past} &= \frac{t_{history}}{dt} \\ count_{wp\ future} &= \frac{t_{horizon}}{dt} \end{aligned} \quad (5)$$

C. Trajectory

In addition to waypoints, the global discrete trajectory is also projected into the ego frame utilizing the geometric transformation defined in Equation 3. Only the spatial coordinates are extracted. Consequently, a single trajectory point is represented as a 2-dimensional vector containing the X and Y coordinates in the ego-centric frame:

$$traj = \begin{bmatrix} x_{ego} \\ y_{ego} \end{bmatrix} \quad (6)$$

These trajectories are subsequently used for the lateral and longitudinal path length losses calculation, as discussed in Appendix G.

D. Goal selection

For each valid ego pose, a goal vector is computed in the form of a waypoint vector from Equation 4. As the *BikeScenes-planning* dataset operates in temporal space, the

Parameter	Type	Explanation
data_root	str	Absolute path to the top data folder for date recordings.
date_time_dirs	list	List of specific date and time directory paths to process.
t_history	float	Time horizon for historical state data extraction (in seconds).
t_horizon	float	Time horizon for future trajectory prediction (in seconds).
dt	float	Time step interval between extracted frames (in seconds).
goal_seconds	float	Time offset in the future to extract the ultimate goal waypoint (in seconds).
include_depth	bool	Flag to load depth maps and concatenate them with RGB image tensors.
excluded_csv_path	str	Path to a CSV file specifying individual scenes to skip.
only_include_scenes_csv_path	str	Path to a CSV file specifying an exclusive subset of scenes to load.
transform	callable	PyTorch transformations (e.g., <code>torchvision.transforms</code>) applied to images.
roll_in_state_vector	bool	If True, encodes and includes the vehicle's roll in the state vector.
exclude_date_time_dirs	list	Specific date and time directories to explicitly skip during parsing.
front_cam	bool	Flag to activate and load the front camera feed.
left_cam	bool	Flag to activate and load the left camera feed.
right_cam	bool	Flag to activate and load the right camera feed.
back_cam	bool	Flag to activate and load the back camera feed.
cache_dir	str	Directory path where the precomputed serialized dataset files are stored.
recompute_cache	bool	If True, forces the script to rebuild the dataset cache, overriding existing files.
precomputed_data_path	str	Direct path to manually load a specific, previously compiled dataset pickle file.

TABLE XII: Configuration parameters of a *BikeScenes* class initialization.

temporal distance of the goal vector from the last computed waypoint is governed by the `goal_seconds` parameter. This parameter also influences the total number of valid samples for training, as the sample time difference between the end of the scene and the last valid waypoints cannot be smaller than `goal_seconds` as shown in Equation 2.

E. State vector

The dataset also stores the state of the SenseBike at each frame. The state can be either a 3 or 5-dimensional vector, and its size can be directly controlled by the `roll_in_state_vector` parameter. A 3-dimensional state vector provides: forward linear velocity, roll rate, and yaw rate as shown in Equation 7. The 5-dimensional state vector encodes current roll in *sin* and *cos* embeddings as in the Equation 8.

$$state_{3D} = \begin{bmatrix} \text{Forward linear velocity} \\ \text{Roll rate} \\ \text{Yaw rate} \end{bmatrix} \quad (7)$$

$$state_{5D} = \begin{bmatrix} \text{Forward linear velocity} \\ \text{Roll rate} \\ \text{Yaw rate} \\ \sin(\text{roll}) \\ \cos(\text{roll}) \end{bmatrix} \quad (8)$$

F. Image fetching

The *BikeScenes-planning* dataset contains synchronized 360° camera feed, as discussed in Appendix B and shown in Figure 4. The *BikeScenes* class provides the front image by default; however, the left, right, and rear image retrieval can be controlled by setting the boolean values of `left_cam`, `front_cam`, `right_cam`, and `back_cam` parameters. Furthermore, if spatial depth is required (`include_depth`), the synchronized depth map is concatenated along the channel dimension of the standard 3-channel RGB image, yielding a comprehensive 4-channel RGB-D tensor.

G. Dataset precomputation

To minimize the CPU bottleneck and eliminate redundant operations during model training, we utilize a dynamic dataset caching mechanism. Rather than executing the complex coordinate transformations, quaternion conversions, and temporal alignments on the fly, the *BikeScenes* class processes the raw spatial and temporal data exactly once for any given configuration.

Upon initialization, the caching pipeline creates a serialized string by aggregating all structural parameters detailed in Table XII. This string is subsequently hashed utilizing the MD5 algorithm to generate a unique hash signature. If the hash signature already exists on the disk and the corresponding pickle file can be accessed, the pipeline immediately loads the precomputed data directly into RAM. Conversely, if no match is found, the script rebuilds the dataset, computes all necessary transformations, and saves the data to a cache file.

The resulting metadata, containing image paths, ego-centric waypoints, trajectories, and state vectors, is precomputed and instantly available to the dataloader. During the training loop, the CPU is strictly used to fetch the target image tensors from the file path, optionally perform data augmentation, and concatenate the RGB-D image. This precomputing stage ensures the dataloader saturates the GPU without CPU throttling, enabling efficient end-to-end model training.

APPENDIX F VALIDATION SET

A subset of scenes from the *BikeScenes-planning* dataset is explicitly isolated to form the validation set. Following the split ratio established by the nuScenes dataset [9], around 15% of the total dataset is allocated to validation, resulting in 128 scenes. The count of scenes is shown in Table XIII. To ensure the validation set is representative, the same distribution of turning events for the validation and training set is enforced and provided in Table XI.

Date	Validation set	Training set	Total
12.11.2025	55	336	391
13.11.2025	73	425	498
Both	128	761	889

TABLE XIII: Scene number per recording date.

Date	Left only	Right only	Straight only	Left & right
12.11.2025	6	4	39	6
13.11.2025	5	8	56	4
Both	11	12	95	10

TABLE XIV: Scene number per event in validation set for a recording date.

The initial proposals for validation set generation are formulated as a linear programming optimization problem. Subsequently, the proposed scenes undergo visual assessment, where environments and taken routes are manually evaluated for complexity. If a proposed recording is rejected during this manual review, the optimization algorithm is repeated with the rejected recording excluded from the available pool.

A. Validation recording proposals

Because the 20-second scenes are extracted from continuous recordings, placing a scene in the validation set requires excluding its entire parent recording from the training set to prevent data leakage. Therefore, the optimization problem is formulated to meet the turning event distribution targets while minimizing the total number of parent recordings used in the validation set, thereby preserving the maximum amount of data for training. Additionally, to prevent a single weather condition from dominating the validation set, a constraint is enforced to include an equal number of recordings from both days. Moreover, to prevent the selection of consecutive recordings from similar parts of Delft, the temporal gap between accepted validation scenes is constrained to a minimum of 30 minutes.

$$\text{Minimize: } \sum_{d \in D} X_d$$

$$\text{Subject to: } \sum_{d \in D} C_{m,d} \cdot X_d \geq T_m \quad \forall m \in M \quad (9)$$

$$\sum_{d \in D_1} X_d = \sum_{d \in D_2} X_d \quad (10)$$

$$X_i + X_j \leq 1 \quad \forall i, j \in D \text{ where } ts_j - ts_i \in [0, t_{\text{gap}}] \quad (11)$$

$$X_d \in \{0, 1\} \quad \forall d \in D \quad (12)$$

Where:

- D is the candidate parent recording set
- M is the set of target maneuvers {left_only, right_only, straight_only, left_and_right}
- D_1 is the subset for 12.11.2025
- D_2 is the subset for 13.11.2025
- $C_{m,d}$ is the number of times maneuver m occurred in recording d

- T_m is the target minimum number of scenes with maneuver m
- t_{gap} is the minimum 30-minute offset between recordings
- $X_d \in \{0, 1\}$ is the decision variable for every recording
- ts_d is the start time of a recording d

B. Visual inspection

All parent recordings proposed by linear programming were manually inspected. This qualitative evaluation prevents the aggregation of recordings from visually similar environments (e.g., multiple recordings from TU Delft Campus), regardless of the imposed temporal separation. The recording is flagged once rejected, and the optimization step is repeated without the rejected recording. The process iterates until all proposed scenes are accepted. The final validation set scene count and distribution of turning events are shown in Tables XIII and XIV.

C. Hard validation set

Upon manual visual inspection of the validation set, the most complex scenes are selected to form a more challenging validation set variation – hard validation set. This subset contains mostly turning scenes and passes through the most congested area around the city of Delft. Additionally, the challenging scenarios and atypical road curvature are intentionally embedded within this subset to evaluate planner robustness.

APPENDIX G

LOSS FUNCTION AND EVALUATION METRIC

Riding a bicycle differs significantly from driving a car. One of the main differences is the operational velocity range. While collecting the data, we registered an average velocity of 3.91 m/s. Consequently, longitudinal deviations between predictions and ground truth are not as significant as for automobiles. However, bicycle lanes are narrower, and cyclists tend to ride closer to each other than cars do, meaning even a slight lateral deviation can have catastrophic results. To address this, we propose a novel approach to evaluate bicycle E2E path planners by decoupling longitudinal and lateral errors into separate metrics. This departs from the standard autonomous driving paradigm that utilizes Average Displacement Error (ADE) and Final Displacement Error (FDE). Both ADE and FDE are based on the Euclidean distance between the prediction and ground truth, effectively combining the lateral and longitudinal deviations into a single value. Consequently, we propose a decoupled evaluation metric to effectively evaluate E2E path planners for bicycles.

A. Composite loss function

The main objective of our E2E planners is to predict future 7-dimensional waypoints discussed in Appendix E-B. Therefore, we introduce a composite loss function that penalizes each component separately. The total loss is calculated for every given predicted waypoint w and then averaged over the entire prediction horizon W .

$$\mathcal{L} = \frac{1}{W} \sum_{w=1}^W \left(\lambda_{\text{lat}} \mathcal{L}_{\text{lat}}^w + \lambda_{\text{lon}} \mathcal{L}_{\text{lon}}^w + \lambda_{\text{roll}} \mathcal{L}_{\text{roll}}^w + \lambda_{\text{yaw}} \mathcal{L}_{\text{yaw}}^w + \lambda_{\text{vel}} \mathcal{L}_{\text{vel}}^w \right) \quad (13)$$

Where discount factors are $\lambda_{\text{lat}} = 0.43$, $\lambda_{\text{lon}} = 0.25$, $\lambda_{\text{roll}} = 0.125$, $\lambda_{\text{yaw}} = 0.125$, and $\lambda_{\text{vel}} = 0.07$.

We prioritize minimizing the lateral offset by assigning it the largest weight in our objective function. We equally penalize the roll and yaw predictions. By providing an explicit loss for velocity, we improve kinematic consistency and generate a more stable reference signal suitable for downstream low-level controllers. Additionally, we implement this metric in vector form to enable an efficient parallel computation on a GPU.

B. Lateral loss

The lateral loss is calculated using the vectorized point-to-trajectory distance. The minimal distance across all segments is extracted simultaneously for every predicted waypoint, which is subsequently scaled by λ_{lat} .

Let $\hat{\mathcal{P}} = \{\hat{P}_1, \hat{P}_2, \dots, \hat{P}_W\}$ be the set of predicted 2D $[\hat{x}, \hat{y}]$ positions extracted from the waypoints, and $\mathcal{S}$ defines the line segments between the consecutive trajectory points where each segment s is bounded by starting point A and end point B .

Then, for a given predicted point $\hat{P} \in \mathbb{R}_{\text{ego}}^2$ for waypoint w ($\hat{P}_w$) and a specific segment s , the continuous projection ratio $t_{w,s}$ is:

$$t_{w,s} = \frac{(\hat{P}_w - A_s) \cdot (B_s - A_s)}{\|B_s - A_s\| + \epsilon} \quad (14)$$

where $\epsilon = 10^{-8}$ is a small constant to prevent division by zero.

To ensure the projected points $\hat{P}'$ lie on the segment s , we clamp the ratio to the range $[0,1]$:

$$\hat{t}_{w,s} = \max(0, \min(1, t_{w,s})) \quad (15)$$

The exact Cartesian coordinate of the closest projection $\hat{P}'_{w,s}$ on the segment is then solved via:

$$\hat{P}'_{w,s} = A_s + \hat{t}_{w,s}(B_s - A_s) \quad (16)$$

Finally, the lateral offset for the predicted waypoint $\hat{P}_w$ is defined as the minimum Euclidean distance to the closest valid projected point across all available path segments $\mathcal{S}$:

$$\mathcal{L}_{\text{lat}}^w = \min \|\hat{P}_w - \hat{P}'_w\| \quad (17)$$

The clamping operation shown in Equation 15 is crucial for preserving the spatial boundaries of the physically feasible path, ensuring the lateral distance metric evaluates the true distance to the trajectory. Because point-to-path loss evaluates spatial accuracy (lateral offset error) independent of temporal

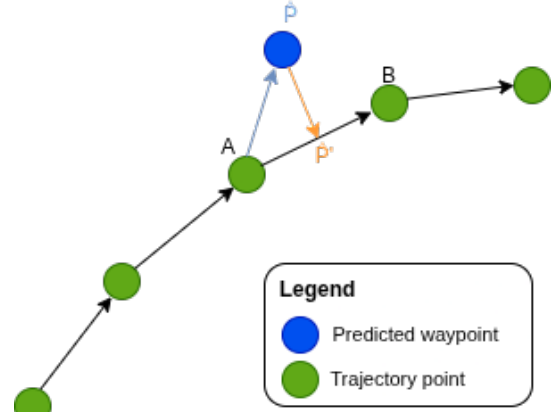


Fig. 28: Vectorized lateral offset calculation.

progression (longitudinal error), it is highly sensitive to incorrect projections, especially while cornering. When a model predicts a trajectory that cuts a corner, the predicted points $\hat{P}$ fall inside the apex of the turn. If we project these points without clamping ($t \in (-\infty, \infty)$), the mathematical projection $\hat{P}'$ will often lock onto the infinitely extended, invisible trajectory of an adjacent segment. Because the corner-cutting point $\hat{P}$ is physically close to the infinite segment lines, the calculated unclamped distance $\|\hat{P} - \hat{P}'\|$ becomes artificially small. This would trick the loss function into treating a dangerous corner-cutting maneuver as highly accurate, failing to penalize the undesired behavior. By clamping the projection ratio to $\hat{t}_{w,s} \in [0,1]$, we force $\hat{P}'$ to remain strictly bounded within the valid physical coordinates of the line segment between trajectory points, guaranteeing the model is accurately penalized for deviating from the actual expert trajectory.

C. Longitudinal path length loss

The longitudinal path length loss evaluates spatial progression by comparing the cumulative path length of the predictions with the ground-truth waypoints along the expert trajectory. Relying solely on Euclidean distance is insufficient, as it can directly reward the planner for stopping at sharp turns rather than following the path. This is because minimizing raw longitudinal displacement in the ego-frame without forward progression may yield artificially low error, as further discussed in Appendix I-B.

To mitigate this, the longitudinal error metric builds upon the vectorized projection of the lateral metric. Using the index of the closest trajectory segment s_c to the projected prediction $\hat{P}'$ and the clamped projection ratio $\hat{t}_{w,s}$, the cumulative path length of all full segments L_{s_c} up to the beginning of the closest segment s_c (where the projection point $\hat{P}'$ lies) is calculated as the sum of all consecutive segments:

$$L_{s_c} = \sum_{i=1}^{s_c-1} \|B_i - A_i\| \quad (18)$$

The total projected path length S_{pred} for a given projection point $\hat{P}'$ is calculated by adding the partial, clamped segment's length:

APPENDIX H E2E PLANNERS

$$S_{\text{pred}} = L_{s_c} + \hat{t}_{w,s_c} ||B_{s_c} - A_{s_c}|| \quad (19)$$

The above derivation is repeated for the ground-truth waypoint to calculate the target path length S_{target} . Because the target waypoint lies on the expert trajectory, the projection and clamping operations mathematically evaluate to the full length of the corresponding last segment.

To compute the final loss, a Huber Loss is used with a delta $\Delta = 1$, as it combines the benefits of Mean Absolute Error (MAE or L_1) and Mean Square Error (MSE or L_2). It is robust to outliers due to linear penalization of the error while maintaining the precision of small deviations.

$$\mathcal{L}_{\text{lon}}^w = \text{HuberLoss}(S_{\text{pred}} - S_w) \quad (20)$$

D. Roll and yaw loss

As described in Appendix E-B, roll and yaw Euler angles are encoded as 2D continuous vectors using sin and cos embeddings to avoid the periodic discontinuities of raw angles. Because the raw predictions $\hat{p}$ do not inherently satisfy the geometrical constraints (i.e., $\sin^2 \theta + \cos^2 \theta = 1$), the predicted vectors are strictly normalized to a unit circle.

$$\hat{u} = \frac{\hat{p}}{||\hat{p}||} \quad (21)$$

Subsequently, the angle error is computed by taking the sum of the squared errors between this normalized prediction and the target continuous vector $t = (\sin \theta, \cos \theta)$:

$$\mathcal{L}_{\text{angle}}^w = (\hat{u}_{\sin} - t_{\sin})^2 + (\hat{u}_{\cos} - t_{\cos})^2 \quad (22)$$

E. Velocity loss

The forward velocity is explicitly penalized as shown in the compound loss function. Although velocity can be implicitly defined by the longitudinal distance between temporally bounded waypoints, relying solely on spatial predictions often yields noisy implicit velocities. In a physical system, such as SenseBike, such noisy predictions may translate to high accelerations.

To compute the velocity error, a Huber Loss with delta $\Delta = 1$ is used. The MAE penalty (L_1) for large deviations prevents the model from predicting physically unfeasible velocity spikes. Conversely, the quadratic nature of MSE for small differences provides smooth gradients near the target velocity.

$$\mathcal{L}_{\text{vel}}^w = \text{HuberLoss}(\hat{v}_{\text{el}} - v_{\text{el}}) \quad (23)$$

A. Architectures

In the following section, we detail the architectures used in the ablation study of MLP (Figure 29a) and GRU (Figure 29b) based models. To evaluate the contributions of each architectural component, we gradually increase the complexity of our network. Starting from a baseline goal-conditioned planner (*Goal*), we progressively incorporate historical context, recurring planning head, and vehicle roll dynamics.

Goal: This model serves as the fundamental baseline for our experiments. It processes visual data using a single Convolutional Neural Network (CNN) with 3 convolutional layers. The network receives a single 4-channel RGB-D or 3-channel RGB image, a 3-dimensional state vector (forward velocity, yaw rate, and roll rate), and the target goal to output the predicted future waypoints directly via multi-layer perceptrons (MLPs).

Past: Building upon the baseline, this iteration introduces temporal context. We add a history branch to encode t_{history}/dt previous waypoints. These past waypoints are provided as 7-dimensional vectors, allowing the model to ground its future predictions based on the recent trajectory of the bicycle.

Gru0: To better capture the sequential nature of path planning, we replace a standard MLP with a Gated Recurrent Unit (GRU) planning head. The initial hidden state is generated from the fused multimodal features in the form of a 64-dimensional feature vector. The GRU step receives a 2-dimensional position vector (a spatial sub-vector of the waypoint) as input, and the first input is initialized with a zero vector.

Gru0 roll: This variation adapts the *Gru0* to explicitly account for bicycle leaning behavior. We expand the state vector from 3 to 5 dimensions to include encoded roll, providing the network with a complete dynamic ego state.

Gru0 7wp: Returning to the 3-dimensional state vector, this model modifies the recurrent input of the *Gru0* architecture. The full 7-dimensional waypoint is fed as input into the GRU at each sequence step instead of 2D spatial sub-vector.

Gru0 7wp roll: This model combines the architectural changes of the two previous iterations. It utilizes the full 7-dimensional waypoints as inputs to the GRU layers while simultaneously incorporating the expanded 5-dimensional state vector to maintain awareness of the current roll.

APPENDIX I EXPERIMENTS

In this Appendix, a detailed description of the performed experiments is provided, along with qualitative analysis.

A. Non-Learnable Geometric Baselines

To build a non-learnable baseline, we evaluate polynomials of 2nd, 3rd, and 4th degree on the validation set, its more challenging hard subset, and a variation containing only sharp turns. We perform the polynomial fitting between the ego pose and the goal. To ensure a fair comparison with the RGB and

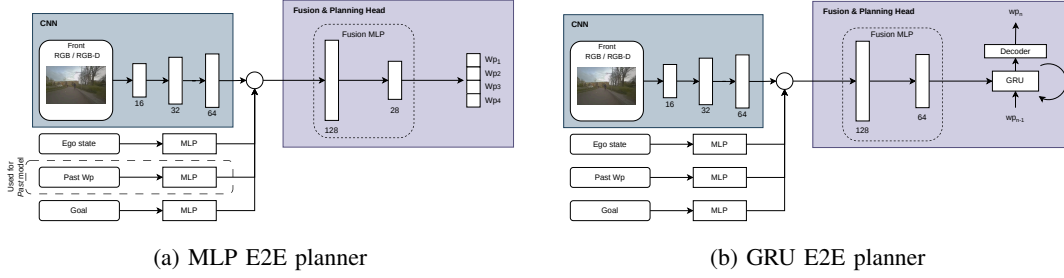


Fig. 29: The architecture comparison between E2E planners with MLP(a) and GRU(b) planning heads.

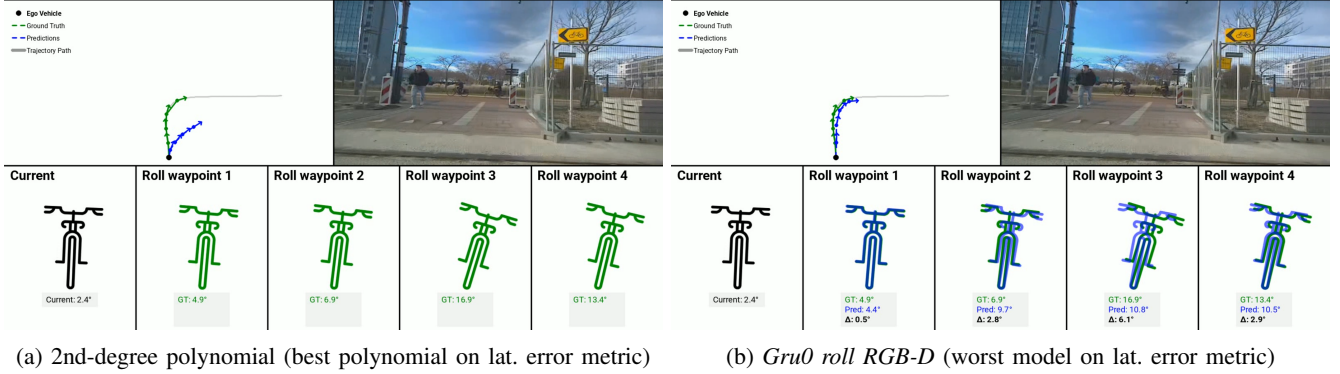


Fig. 30: The challenging right turn shows the superiority of the worst *Gru0 roll RGB-D* model over the basic polynomial fitting. While the best-performing polynomial cuts the corner and departs from the drivable area, the worst-performing E2E planner plans a safe path.

RGB-D models, the samples from a 4-second horizon, spaced at 1-second intervals, are isolated for evaluation. The polynomials are evaluated using the same lateral and longitudinal path length metrics as our models.

The 2nd-degree polynomial performs best on both metrics among all evaluated non-learnable methods. Additionally, this simple curve-fitting method scores better on the lateral error metric than the worst RGB-D model *Gru0 roll* (Lat. err = 61.48 cm in Table III). Unlike E2E planners, polynomial methods lack semantic understanding of the scene, as they directly connect the current ego pose (origin) with a goal. This may lead to catastrophic, corner-cutting behavior as shown in Figure 30a. The polynomial simply cuts corners and outputs the path through the non-drivable area. Conversely, the worst overall model, *Gru0 roll RGB-D*, correctly reasons that it is necessary to stay within the drivable area and turn where possible.

The straight path segments are present in all variations of the validation set, as the bike paths feature rather sharp than long turns, resulting in turns lasting a few seconds in 20-second-long scenes, as shown in Figure 2. The polynomial method exploits such scenarios and simply fits a straight line to the goal, exhibiting minimal lateral offset error. However, it cannot build a proper progression understanding, as its longitudinal path length error grows to a massive 1.28 meters on the hard set and almost 2 meters on the sharp turns subset, assuming constant velocity even while turning.

Although the lateral error is the most important metric, this experiment shows the need to consider the longitudinal

path length metric alongside it when evaluating E2E bicycle path planners as a secondary decoupled metric. The qualitative analysis in Fig. 30 provides empirical evidence of the superiority of our learnable method.

B. Path length loss vs Cartesian Loss

To evaluate the benefits of our proposed longitudinal path length loss over the Cartesian equivalent for evaluating E2E bicycle path planners, a comparison based on six metrics is presented. Progress ratio, path length error, and jerk demonstrate direct benefits through simple scalar metrics. Furthermore, to provide an in-depth analysis of cornering behavior, the velocity error calculated using spatial data (Vel. Error Correlation), the explicitly predicted speed (Pred. Explicit Speed Corr.), and the predicted speed calculated using spatial data (Pred. Spatial Speed Corr.) are correlated with the change in heading (turning). Pearson correlation is used to calculate the respective correlation in this study. The evaluation is performed on a manually crafted subset of the validation set containing only scenes with sharp turns, as such scenarios are safety-critical and uncover the underlying flaw of the Cartesian longitudinal loss. The results of this comparison are provided in the Table V.

Progress ratio: This metric denotes progress along the expert trajectory by calculating the path-length ratio between the last predicted waypoint projection on the expert trajectory and the last ground-truth waypoint. This effectively measures the ego-vehicle’s longitudinal progress along the trajectory. A progress ratio of 1.0 indicates that the model perfectly

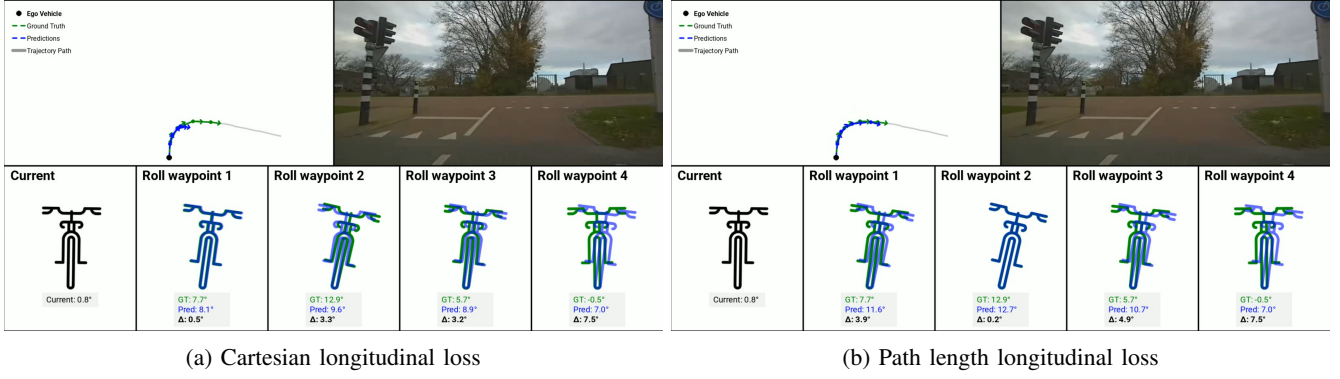


Fig. 31: The cornering planner behaviors comparison between Cartesian (a) and path length (ours) longitudinal loss (b). Both Figures show the predictions from the same time instance. The underlying flaw of the Cartesian longitudinal loss is shown as a congestion of the waypoints at the turn’s apex. On the contrary, our proposed path length loss mitigates the slowdown issue during cornering, yielding a feasible path.

Polynomial Baseline	Lat. err ↓	Lon. err ↓
Validation Set		
2nd Degree	31.90	60.06
4th Degree	92.73	68.95
3rd Degree	194.37	75.69
Gru0 7wp	18.02	9.00
Hard Validation Set		
2nd Degree	59.75	128.07
4th Degree	257.19	157.21
3rd Degree	600.99	180.46
Gru0 7wp	28.16	17.02
Sharp Turns Validation Set		
2nd Degree	79.08	199.39
4th Degree	455.45	253.99
3rd Degree	1099.55	314.27
Gru0 7wp	28.98	24.93

TABLE XV: Evaluation of non-learnable polynomial baselines and a direct comparison to the best RGB model, *Gru0 7wp* evaluated on the entire validation set and its more challenging subsets. The trajectories were generated using the 8-second goal coordinate and evaluated at the exact spatial fractions corresponding to 1, 2, 3, and 4 seconds to ensure fair comparison with the neural network models. Lat err - lateral error in centimeters, Lon. err - longitudinal path length error in centimeters.

matches the expert demonstration’s longitudinal progress. The model trained with our proposed path-length loss achieves superior performance on this segment-completion task, effectively mimicking expert longitudinal progression.

Path length error: This metric utilizes Mean Absolute Error (MAE) to calculate the difference between the ground truth and predicted path length for every waypoint in meters. Unlike the progress ratio, the path length error captures the intermediate steps, revealing detailed differences in longitudinal progression averaged across all waypoints. It effectively indicates how far ahead or behind the predicted waypoints are compared to the ground-truth data. The application of

our proposed loss function reduced this longitudinal error by 32.3%, achieving an average error of 0.534 m.

Jerk: Jerk quantifies the rate of change of acceleration and is calculated through the third derivative of the 2D spatial position predictions over time. This metric serves as a proxy for riding comfort by evaluating the physical smoothness of the predicted waypoints. The model optimized with our proposed path length loss demonstrates a reduction in spatial jerk, which corresponds directly to a smoother steering and milder accelerations compared to the Cartesian baseline. As the jerk is calculated using 4 predicted waypoints sampled at 1 Hz, the magnitude equal to 0.336 (Table V) achieved by the best RGB model, *Gru0 7wp*, exhibits a smooth riding behavior.

Velocity error correlation: This metric evaluates Pearson correlation between the velocity error, calculated through first derivative of spatial predictions, and the change in heading (turning). The model optimized with Cartesian loss exhibits a strong positive correlation of $r = 0.855$. This directly indicates the inaccurate spatial predictions during cornering maneuvers, demonstrating that the Cartesian loss allows the E2E planner to artificially minimize error by compromising longitudinal dynamics. Conversely, the model optimized with our proposed path length loss effectively mitigates this vulnerability, achieving a significantly lower velocity error correlation of 0.360 (a 57.9% reduction).

Predicted speed calculated using spatial data: This metric correlates the change in heading with the speed derived from spatial waypoints. The expert’s ground truth correlation of $r = -0.276$ corresponds to the human-like slowdown behavior while cornering. This evaluation further directly demonstrated the flaw in longitudinal waypoint prediction for model trained with Cartesian Loss. The correlation of the model optimized with Cartesian loss ($r = -0.617$) is significant, indicating closely spaced waypoints and, therefore, a major reduction in speed during cornering (Figure 31). In contrast, the proposed Path Length Longitudinal Loss achieves a correlation of -0.356 (a 42.3% reduction in magnitude), which closely approximates the expert behavior and therefore signals appropriate spacing between the waypoints.

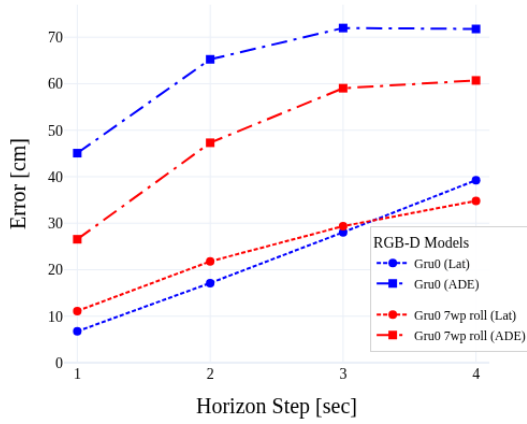


Fig. 32: Average ADE vs Lateral Error over the 4-second prediction horizon with a timestep of 1 second. The RGB-D models were evaluated on the validation set. The ■ denotes the scores obtained on the ADE metric, whereas ● denotes scores on the lateral error metric.

Explicitly predicted speed error correlation: Additionally to metrics utilized in detecting the flawed cornering behavior, the correlation between the explicit speed predictions and cornering is evaluated. Both the Cartesian and path length metrics exhibit similar, near expert-like correlation values. This demonstrates that, regardless of the longitudinal loss used in training, models successfully learn to predict a proper reduction in speed during cornering through their explicit speed outputs (the last entry in the 7-dimensional waypoint).

Conclusion and empirical proof: The *Pred. Spatial Speed Corr.*, *Vel. Error Correlation*, *Path Length Error (s_{mae})*, and *Progress Ratio*, metrics effectively highlight the inherent limitations of the baseline Cartesian loss. Optimizing models with Cartesian loss enables the network to artificially minimize error in the absolute longitudinal ego-x direction by clustering waypoint predictions at the apex of a turn. Conversely, the proposed Path Length Loss ensures accurate longitudinal progress along the trajectory by relaying on the cumulative path lengths rather than the absolute cartesian coordinates.

The predictions generated by the best RGB planner, *Gru0*, optimized using both losses respectively, are shown in Figure 31. The incorrect progression caused by Cartesian loss is visible in Figure 31a as a congestion of predicted waypoints at the turn’s apex. Because these waypoints are spread temporarily with a timestep of 1 second, this spatial clustering indicates that the network is effectively predicting a sudden stop mid-turn. In contrast, as shown in Figure 31b, the proposed longitudinal path length loss effectively prevents such behavior, ensuring continuous and realistic progression along the intended path.

C. ADE vs lat/lon decoupled loss

The lateral and longitudinal errors need to be decoupled when evaluating bicycle path planners, as described in Appendix G. To further support this claim, in addition to the data provided in Table VI, we present a detailed performance of the best RGB-D model on the ADE metric (*GRU0 7wp*

roll) and the best model on the lateral error metric (*GRU0*) evaluated on the validation set.

The averaged error on both ADE and lateral offset metrics over the 4-second prediction horizon, with a 1-second timestep, is shown in Figure 32. The predominant model for lateral error across RGB-D variations exhibits a significantly higher ADE across the entire horizon than the best ADE model. This may yield the incorrect conclusion that the *Gru0* model performs significantly worse overall, while exhibiting lower lateral error on average.

Moreover, because two planners perform similarly well, the longitudinal error directly causes the significant performance gap between the models evaluated on ADE, as shown in Figure 35d. As previously highlighted, the longitudinal error in the low-velocity range should not be the dominant metric; in the bicycle domain, minimizing lateral error is far more important than accurately matching the expert’s speed. Nevertheless, it is advised to evaluate these two metrics separately, as a coupled ADE representation can yield misleading results.

D. Lateral error in prediction horizon

To further investigate the change in lateral error over the 4-second predictions, temporally spaced at 1-second intervals, a horizon analysis is performed. Both RGB and RGB-D models were evaluated across the entire validation set and the hard subset. The results of the lateral error along the prediction horizon are shown in Figure 34.

All planners exhibit a gradual decrease in performance with increasing horizon, a phenomenon also observed in the automotive domain. This is mostly attributed to the compounding errors. The small deviation at timestep $t = 1$ second propagates the error throughout the entire horizon.

The RGB models with the MLP planning head exhibit noticeably worse performance on the lateral error metric, most clearly on the hard subset evaluation (Figure 34b), compared to memory-preserving GRU models. We argue that the recurrent nature of the GRU can learn the underlying bicycle dynamics and thereby improve the lateral error over the entire horizon. The single-shot MLP planning heads lack such an ability and therefore exhibit lower performance. Furthermore, on average, models using RGB-D inputs tend to benefit from the GRU planning head and achieve lower lateral error than single-shot MLPs, but the differences are less profound, with the exception of *Gru0 roll* planner. However, all of the planners’ scores are worse than their RGB variations.

E. Longitudinal path length error in prediction horizon

A detailed analysis of the longitudinal path length error across the four-second prediction horizon for both RGB and RGB-D models is presented in Figure 35. As shown in Figure 35a and 35b, the error for the RGB models increases linearly over time across all configurations. This error accumulation is an expected behavior in trajectory prediction, primarily due to compounding temporal errors.

While the RGB-D models exhibit similar linear degradation in performance on average, two exceptions highlight the impact of architecture on temporal reasoning in the longitudinal

direction. The simplest *Goal* exhibits a significant performance degradation along the horizon. We hypothesize that this is due to the single-shot nature of MLP, which, supplied with noisy RGB-D input data, cannot build a proper understanding of the bicycle motion. In contrast, the recurrent architectures, when paired with the proper ego-state dimensionality in their input layer, essentially flatten the error curve at later time steps. This indicates that such GRU architectures can successfully learn the correct longitudinal progression along the path.

F. RGB vs RGB-D

The RGB-D models performed significantly worse than their RGB equivalents, with a noticeable drop in performance across both the full validation set and the hard subset. For example, on the hard subset, the best-performing RGB-D model on a lateral error metric (*Gru0* at 35.60 cm) exhibited a 26.42% increase in error compared to the best RGB model (*Gru0 7wp* at 28.16 cm). To further investigate this degradation, we conducted a qualitative analysis of the spatial data. To collect the first iteration of the BikeScenes-planning (this Thesis), the cameras were recording at the lowest resolution (SVGA) due to insufficient computational resources to perform capture with higher resolution (Appendix C-A1). As a result, the SVGA capture resolution constrained the extraction of the depth images using ZED SDK to basic NEURAL depth model. As illustrated in Figure 33b, the resulting depth images were far from ideal: building edges appeared jagged, and thin structures like light poles exhibited the vanishing effect. This highly noisy and imprecise spatial input likely destabilized the E2E path planners, directly degrading their performance.

To mitigate this bottleneck, we changed our data collection process. After updating the ZED SDK and camera drivers to the latest release, we developed a custom recording pipeline that bypasses the cumbersome GUI and utilizes a multi-threaded script (Appendix C-A2). This allowed the cameras to operate at HD1200 resolution, enabling the use of the best NEURAL_PLUS depth model to extract 16-bit depth images. The resulting depth maps demonstrate improvements in both resolution and spatial accuracy - buildings contours appear naturally sharp, and the light pole is easily distinguishable in the visual comparison (Figure 33). While the SVGA and HD1200 resolution captures were performed on different recording dates, the perfect one-to-one comparison is challenging. However, our qualitative findings align with the official ZED SDK documentation: in the 10-12 meter range, the standard NEURAL model exhibits an error rate of $< 6\%$, whereas the NEURAL_PLUS model significantly reduces it to $< 2\%$.

G. Speed/Roll - turn correlation

To quantify the models' ability to correctly learn the underlying dynamics, a correlation study between speed/roll and cornering is conducted. We compare the RGB and RGB-D models against the expert behavior (GT). The results in Table XVI show that all models learned the coupled dynamics of bicycle cornering, successfully capturing the relationships between speed, roll, and change in heading. Models are

Model	Pred. Explicit Speed Corr.	Pred. Roll Corr.
GT	-22.91	-81.45
RGB Input Models		
Goal	-32.70	-86.19
Past	-28.53	-84.04
Gru0	-34.82	-82.70
Gru0 7wp	-31.13	-86.90
Gru0 roll	-31.65	-87.56
Gru0 7wp roll	-28.39	-76.98
RGB-D Input Models		
Goal	-35.38	-89.05
Past	-37.32	-90.51
Gru0	-16.38	-87.48
Gru0 7wp	-30.58	-82.88
Gru0 roll	-30.18	-77.00
Gru0 7wp roll	-31.51	-84.20

TABLE XVI: Roll and turn correlation metrics comparing RGB and RGB-D models' variations evaluated on a subset of the validation set containing only sharp turns. GT denotes the ground truth correlation, Pred. Explicit Speed Corr. - correlation between explicitly predicted speeds and cornering; Pred. Roll Corr. - correlation between roll predictions and turning. The negative correlation of *Pred. Explicit Speed Cor* denotes increased speed reduction in predictions while cornering, whereas the negative correlation in *Pred. Roll Corr.* denotes the roll angle increase while cornering (in the ROS-REP 103 coordinate system).

evaluated on a manually selected subset of the validation set, which contains only sharp turns.

Explicitly predicted speed correlation: All models except *Gru0 RGB-D* exhibit a stronger negative correlation between explicitly predicted velocity (last entry in 7D waypoint prediction) and cornering than the expert. This means that, on average, models tend to turn at lower speeds, acting as more cautious riders. The *Gru0 RGB-D* outlier exhibits a weaker correlation, resulting in greater speed predictions while cornering, coupled with a significant longitudinal error (see Fig. 35d). While its absolute difference to the ground-truth correlation is third-best overall ($\Delta = 6.53$), we argue that it struggles to build an appropriate understanding of cornering deceleration behavior. In contrast, the most expert-like model, *Gru0 7wp roll RGB*, with the correlation difference of $\Delta = 5.48$, most accurately mimics human rider speed while cornering.

Roll angle predictions correlation: As the coordinate frame of our dataset is in accordance with ROS-REP 103 (Right-handed, z-up, x-forward), turning left yields positive values while turning right yields negative values. Conversely, leaning into the left turn yields negative values, while leaning into the right turn yields positive values. Because of this inverse relationship between the leaning and turning direction, the results for roll correlations in Table XVI are negative.

The results show that all models learned the underlying bicycle cornering dynamics by correctly associating leaning with turning. The absolute correlation difference of $\Delta = 1.25$ shows that the most human-like roll and cornering association has been achieved by model *Gru0 RGB*. This indicates that



(a) SVGA Resolution - RGB



(b) SVGA Resolution - NEURAL Depth



(c) HD1200 Resolution - RGB



(d) HD1200 Resolution - NEURAL_PLUS Depth

Fig. 33: Comparison of stereo depth estimation under different sensor configurations. The top row shows severe edge degradation and object halos when using lower SVGA resolution and the standard NEURAL model from the ZED SDK to estimate depth. The bottom row highlights the improved spatial precision and edge preservation achieved with HD1200 resolution and the NEURAL_PLUS depth model.

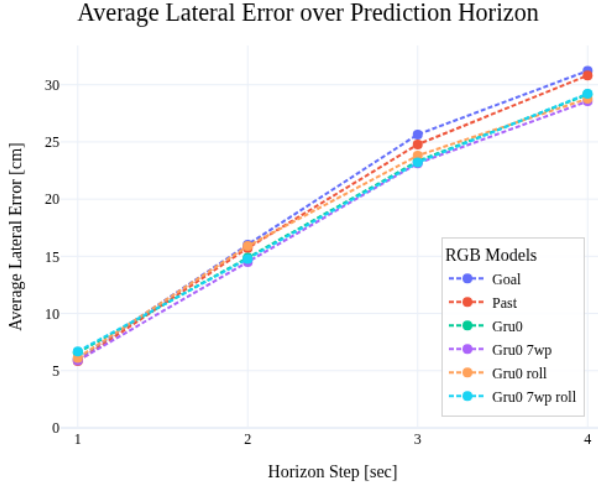
the model learned to predict the necessary roll for future waypoints while cornering almost identically to a human rider.

Conclusions: All model variations learned the underlying dynamics of bicycle cornering. However, to truly validate the performance on challenging turns, a closed-loop evaluation is required. Such an evaluation necessitates integration with an accurate, dynamically rigorous bicycle simulator.

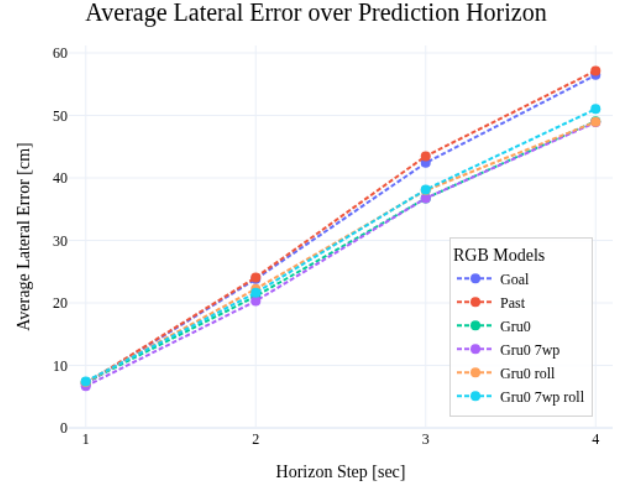
H. Qualitative Examples

In this subsection, a comprehensive qualitative analysis of the best RGB model, *Gru0 7wp*, is provided. Figures from 36 to 47 depict the model’s behavior in various scenarios. The predictions are marked in **blue**, whereas the ground truth data in **green** color. The top-left panel shows the dense expert trajectory in gray as well as ground-truth and predicted spatial components of the waypoints in the ego-centric frame. The top-right panel contains the front camera image (left lens), whereas the bottom panel shows the current lean, as well as predicted and ground-truth roll for each waypoint in degrees. The absolute difference between the predicted and ground-truth roll angles is shown for each waypoint prediction (Δ).

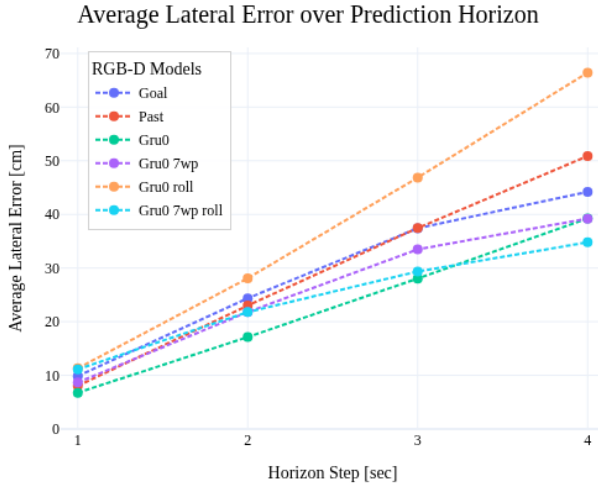
A failure case is depicted in Figure 45, where the planner deviates from the expert’s trajectory due to the uniform structure of the bike path and sidewalk. A corner-cutting behavior is provided on a scene with a slow 90° turn in Figure 46. The model successfully recovers the undesired behavior once the turn is initiated, as shown in Figure 47.



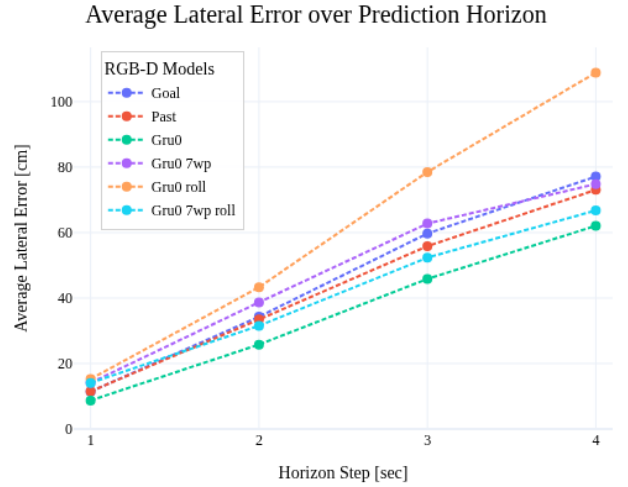
(a) RGB models evaluated on entire validation set



(b) RGB models evaluated on hard validation set

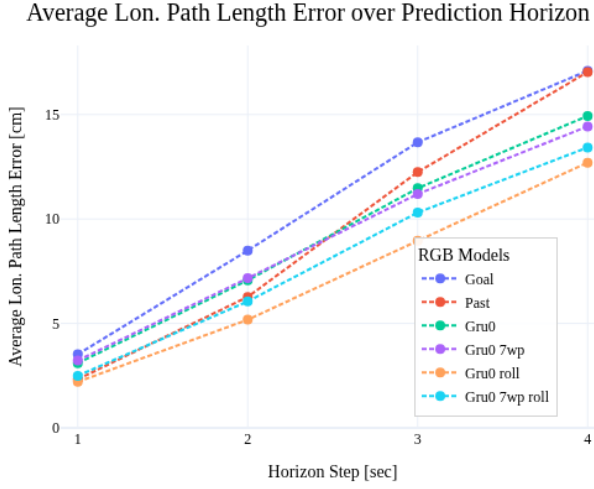


(c) RGB-D models evaluated on entire validation set

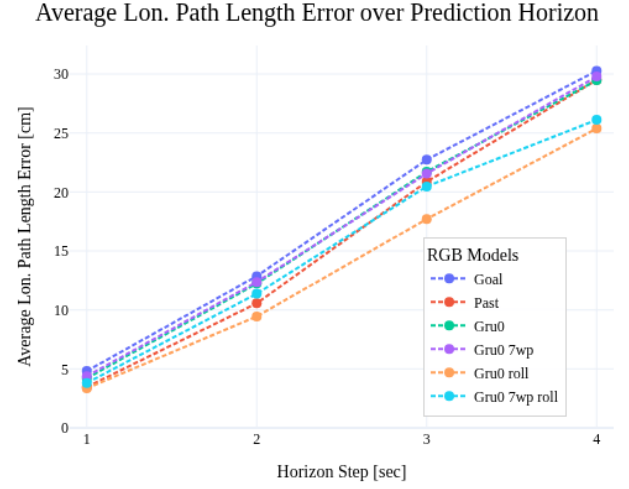


(d) RGB-D models evaluated on hard validation set

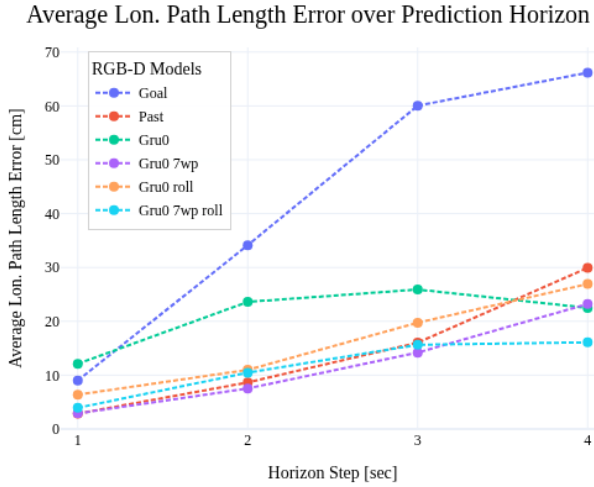
Fig. 34: The lateral error in the prediction horizon of 4 seconds in the 1-second timestep for the RGB (a,b) and RGB-D models (c,d) evaluated on both the entire validation set (a,c) and a hard subset of the validation set (b,d).



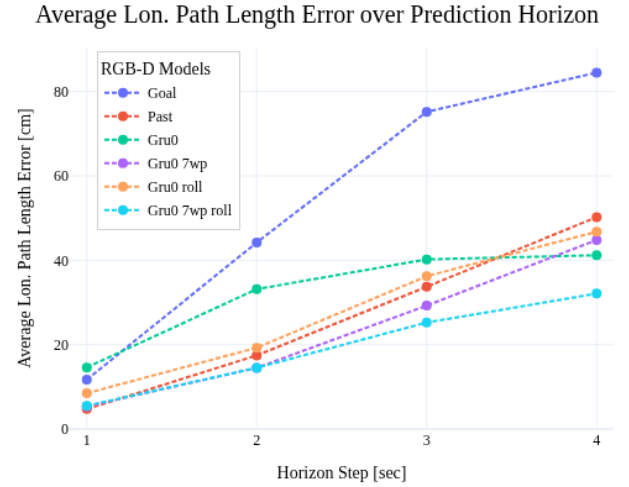
(a) RGB models evaluated on entire validation set



(b) RGB models evaluated on hard validation set



(c) RGB-D models evaluated on entire validation set



(d) RGB-D models evaluated on hard validation set

Fig. 35: The longitudinal path length error in the prediction horizon of 4 seconds in the 1-second timestep for the RGB (a,b) and RGB-D models (c,d) evaluated on both the entire validation set (a,c) and a hard subset of the validation set (b,d).

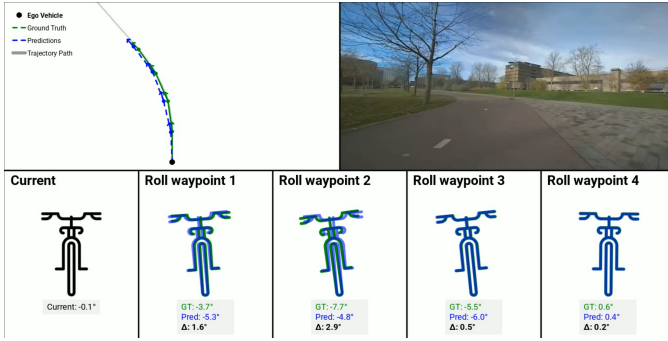


Fig. 36: Left long turn. The model correctly correlated roll with the change in heading (turning).

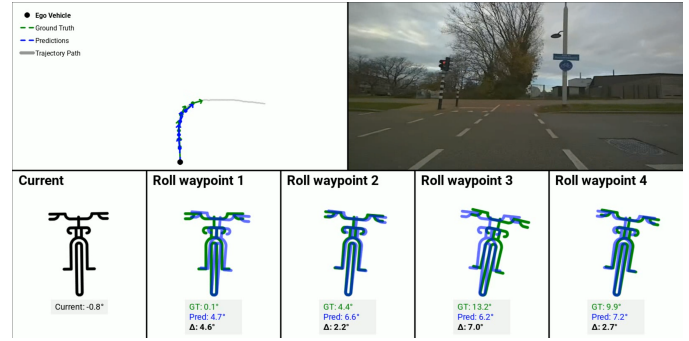


Fig. 40: Sharp right turn. Model correctly predicts the path within the drivable area (no corner-cutting).

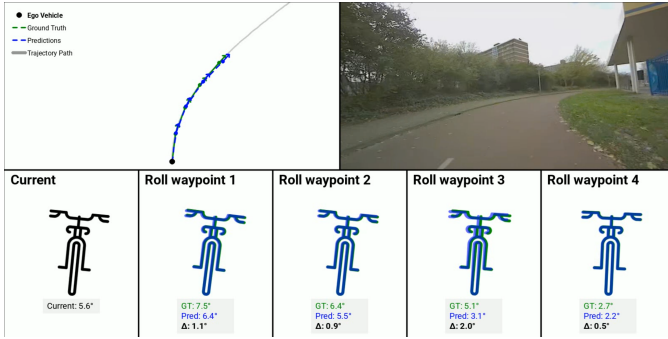


Fig. 37: Right long turn with accurate spatial, roll, and heading predictions.

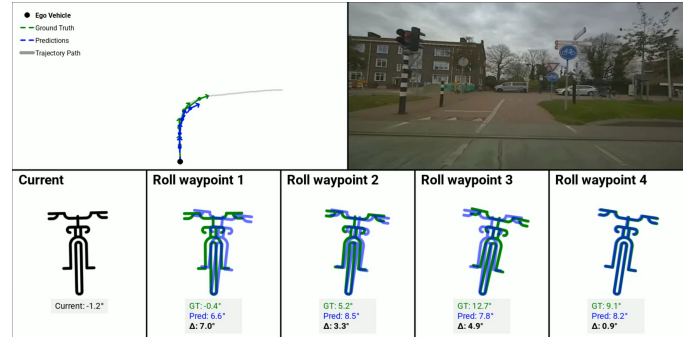


Fig. 41: Sharp right turn. Model correctly predicts the path within the drivable area (no corner-cutting).

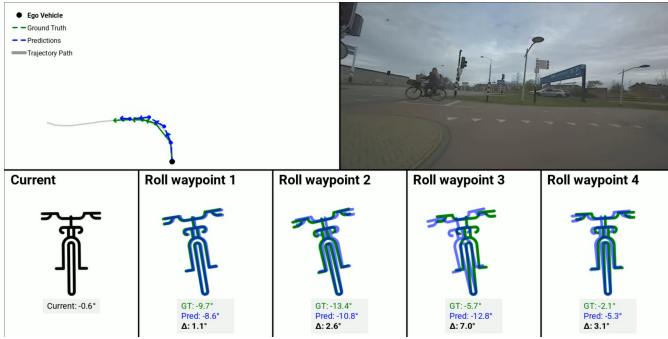


Fig. 38: Sharp 90° turn. The model accurately predicts the path and slightly more aggressive lean for the 3rd and 4th waypoints. A slowdown behavior while cornering is visible by the closer predicted waypoint spacing along the path.

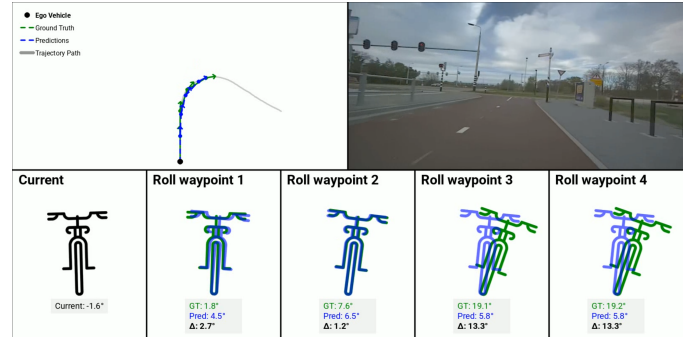


Fig. 42: Model correctly predicts the 7-dimensional waypoints while approaching a sharp right turn.

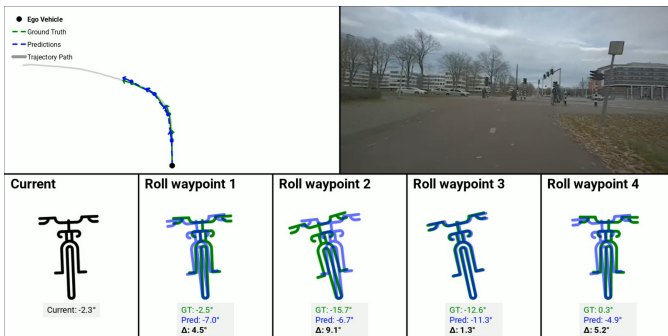


Fig. 39: Moderately long left turn.

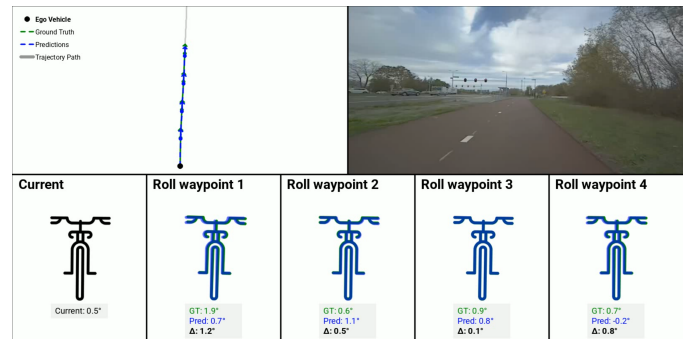


Fig. 43: For straight segments, the model predicts an accurate path with appropriately spaced waypoints while maintaining a steady horizontal position.

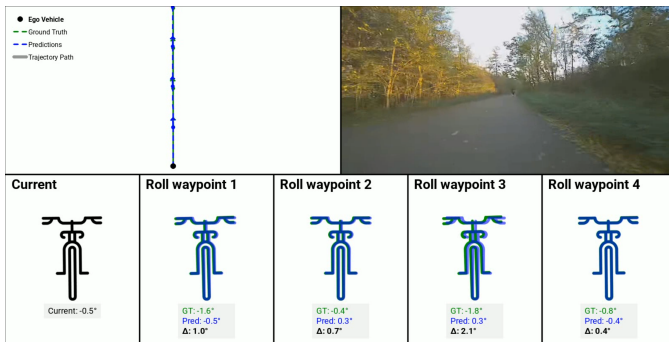


Fig. 44: Model correctly correlates the higher speeds with the greater spacing between predicted waypoints due to riding downhill.

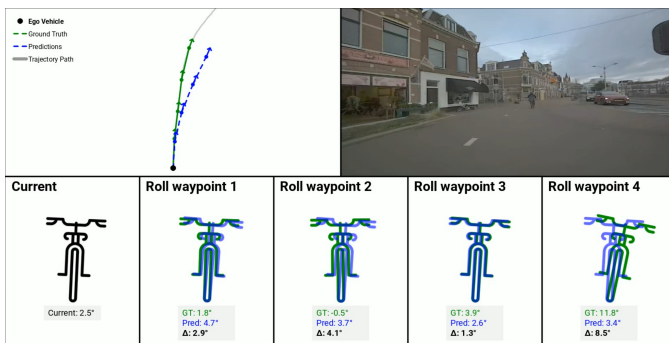


Fig. 45: The model fails to correctly identify the drivable area and therefore, is cutting corners. We hypothesize that it is because the bike path cannot be easily distinguished from the sidewalk.

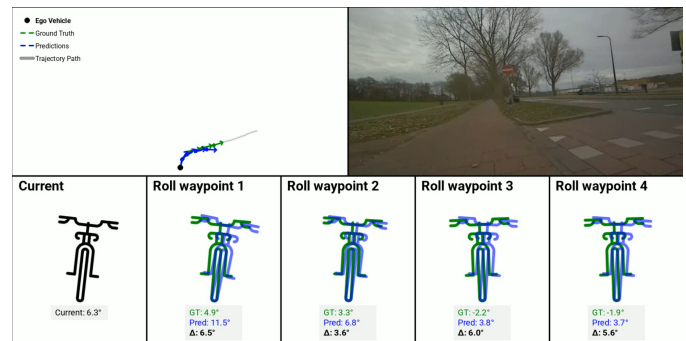


Fig. 47: The model recovers from corner cutting behavior (46) once the turn is initiated.

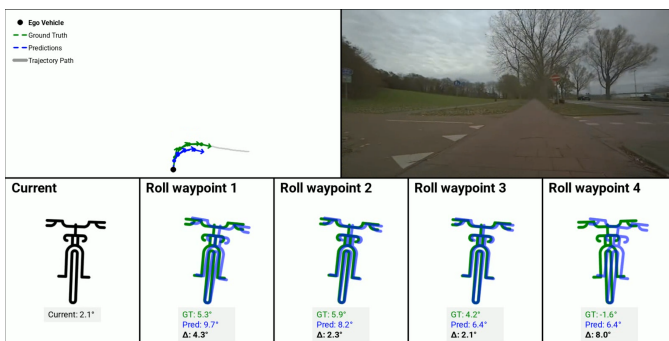


Fig. 46: The model fails by cutting corner. It eventually recovers after initiating the turn.