Fakultät für Informatik
Lehrstuhl für Echtzeitsysteme und Robotik

Constructing a Driving Dataset near a Bus Station

Ahmed Ben Hassen, Bassem Trifa, and Adem Yahmadi

Practical Course *Motion Planning for Autonomous Vehicles* WS 2025/26

Advisor:	Youran Wang
Supervisor:	Prof. Dr.-Ing. Matthias Althoff
Submission:	26.02.2026

Constructing a Driving Dataset near a Bus Station

Ahmed Ben Hassen, Bassem Trifa, and Adem Yahmadi
Technische Universität München
Email: ahmed.benhassen@tum.de, bassem.trifa@tum.de, adem.yahmadi@tum.de

***Abstract*—In this work, we present a computer vision-based pipeline that can extract the trajectories of all road users from drone recordings. The main goal of this method is to generate training data for the development of autonomous buses. To this end, videos were recorded near a bus stop in order to capture various scenarios that a bus may encounter. Our method is based on the pipeline used to create the MONA (Munich Motion Dataset of Natural Driving) dataset and adapts it to address the different challenges of our use case, including small objects and crowded scenes.**

I. Introduction

The development of autonomous buses requires accurate and realistic training datasets. These datasets can also be used to benchmark motion planners. Therefore, they should be derived from real-world scenarios that any bus may encounter when interacting with other road users. Drone recordings are used for this purpose because they offer several advantages, such as fewer occlusions and the ability to capture the entire scene. In this work, we use the same approach that was used to create the MONA [1] dataset and adapt it to our use case. In the initial stage of our approach, the trajectory extraction pipeline performs detection, pose estimation, multi-object tracking, and smoothing. In the second part, the trajectories are converted to the CommonRoad [2] format so that they can be used in motion planning simulations.

II. Related Work

Trajectory datasets remain an essential component in the development of autonomous systems. Depending on the use case and application area, several studies have focused on different scenarios to capture the behavior of road users in various locations. A common approach to collecting the data is to use drone footage. Using this approach, the highD [3] dataset extracts vehicle trajectories on highways and provides 16.5 hours of measurement data for safety validation. On the other hand, the inD (intersection Drone dataset) [4] and rounD [5] datasets focus on intersections and roundabouts, enabling the study of more complex scenarios that also include pedestrians and bicycles. In contrast, the MONA dataset [1] uses stationary cameras mounted in an elevated position to record videos and extract relevant information for a motion prediction and planning use case.

III. Trajectory Extraction Pipeline

Our trajectory extraction pipeline consists of five stages. It takes the drone recordings as input and outputs the trajectories of all traffic participants.

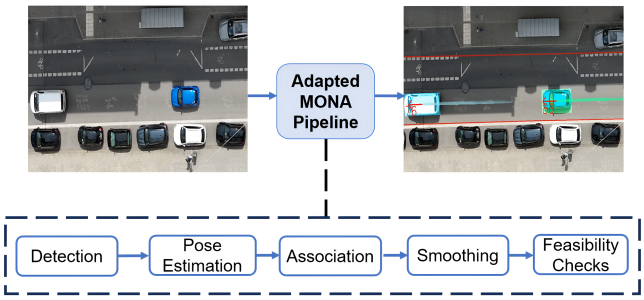

Fig. 1. Overview of the trajectory extraction pipeline.

A. Detection Stage

The detection stage, as shown in Figure 2, outputs the bounding box, segmentation mask, and the position of each detected object in pixel coordinates.

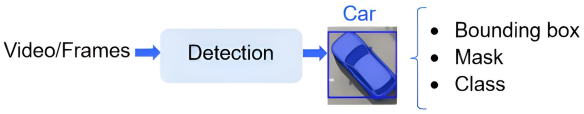

Fig. 2. Overview of the detection stage.[1]

1) Detector: In the MONA dataset pipeline [1], Detectron2[2] is used with pre-trained COCO [6] weights. Its performance on our data is suboptimal due to the domain-shifted viewpoint, where cars appear as roof-mounted rectangles, and the small size of certain object classes. While the domain shift can be resolved through fine-tuning, the main problem remains the small size of some road users, such as pedestrians and bicycles. These classes occupy very few pixels, especially when the drone is recording from a high altitude. To solve this problem, we adopted the Transformer Prediction Head YOLOv5 (TPH-YOLOv5) [7]. Compared to the original YOLOv5[3], it has been adapted to small objects by adding an additional prediction head and replacing some convolutional layers with transformer modules. The new detector was first pre-trained on the VisDrone dataset [8] and then fine-tuned on our data.

2) Segmentation Model: Masks are primarily used in the pipeline to estimate the heading of specific road users, such as cars, buses, and trucks. Due to the lack of ground truth masks,

[1]The figure quality was optimized using FigureLabs.
[2]github.com/facebookresearch/detectron2
[3]github.com/ultralytics/yolov5

fine-tuning a segmentation model is not possible. Instead, the bounding box of each detection can be used as a prompt for a segmenter to estimate the foreground. Therefore, the Segment Anything Model (SAM) [9] is used to generate the masks without additional training. Based on the trade-off between speed and accuracy, various variants of SAM were tested, such as ViT-H SAM, ViT-L SAM, ViT-B SAM, and MobileSAM [10]. Among all these variants, we chose MobileSAM because it offers a good compromise between speed and accuracy. On our data, it is approximately four times faster than ViT-H SAM while still maintaining good performance. As we pass each bounding box to MobileSAM, the number of bounding boxes in each image affects the segmentation speed. Thus, segmentation takes longer in crowded scenes. To avoid this problem, several techniques have been implemented:

- **Frame Skipping:**
  The generated masks are mainly needed to estimate the direction. Since we have 30 frames per second, the orientation angle of the detected objects does not change significantly between two consecutive frames. Therefore, skipping some frames does not affect the accuracy of our results, but improves the runtime.
- **Input Image Downscaling:**
  The number of pixels to be processed and embedded by the segmenter has a significant impact on the computational speed. Thus, reducing the resolution of the input image and reducing the number of pixels speeds up the process while keeping the segmentation masks accurate. The downscaling factor is a hyperparameter that must be set depending on the height of the drone.

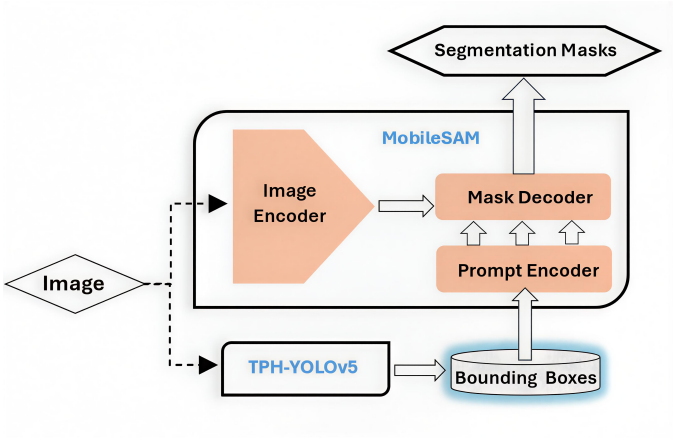

Fig. 3. Detection stage overview: The input image is forwarded to both the detector and the MobileSAM image encoder. The TPH-YOLOv5 outputs the bounding boxes, which are sent to the segmenter's prompt encoder. Based on the embeddings of the input image and the result of the prompt encoder, the mask decoder generates a segmentation mask for each detected object.[5]

B. Pose Estimation Stage

The pose estimation stage outputs the heading, length, width, and the position of each detection in world coordinates.

[5]The figure quality was optimized using FigureLabs.

1) Coordinate Transformation: Since only the videos are available as data, external sources such as Google Maps[6] and Google Earth[7] are used to extract the world coordinates. Several reference points, such as points on lane markings, are selected from the videos and these external sources to estimate a homography matrix that maps pixels to world coordinates with the lowest possible mean reprojection error. The homography matrix is computed using the `cv2.findHomography` function from the OpenCV[8] library. The best accuracy is achieved by the TUM2TWIN[9] project, which created an orthophoto of the streets, including our recording location.

2) Rotated Bounding Boxes: The axis-aligned bounding boxes provided by the detection stage do not always reflect the actual geometry of the object. Instead, the exact shape can be better estimated using the segmentation mask. A good example of this is a car in a curve, as shown in Figure 4. Therefore, we tightly enclose the segmentation mask with a rotated bounding box in order to better estimate the orientation of the detected objects.

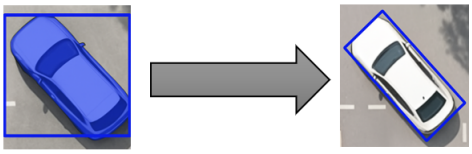

Fig. 4. Axis-aligned vs. rotated bounding boxes: On the left, we can see the axis-aligned bounding box and the segmentation mask; on the right, we can see the rotated box derived from the mask.[11]

3) Heading Estimation: Depending on the class of the detected object, two approaches were used for the heading estimation.

- **Heading Estimation for Vehicles, Buses, and Trucks:**
  The main idea is to estimate the orientation based on the rotated bounding box by calculating the angle between its length and the x-axis direction. All steps are shown in Figure 5.
- **Heading Estimation for Pedestrians, Bicycles, and Motorcycles:**
  The orientation of pedestrians, bicycles, and motorcycles is derived from their motion. Specifically, we first estimate a motion vector between two frames and then calculate its direction. This approach requires that detections are linked to a single track ID. Therefore, the initial orientation for these classes is set to NaN. After the association stage, the heading is calculated, and these NaN values are replaced with the correct angle values.

[6]www.google.com/maps
[7]earth.google.com
[8]github.com/opencv/opencv
[9]tum2twin
[11]The figure quality was optimized using FigureLabs.
[13]The image of the car was taken from VectorStock.com.

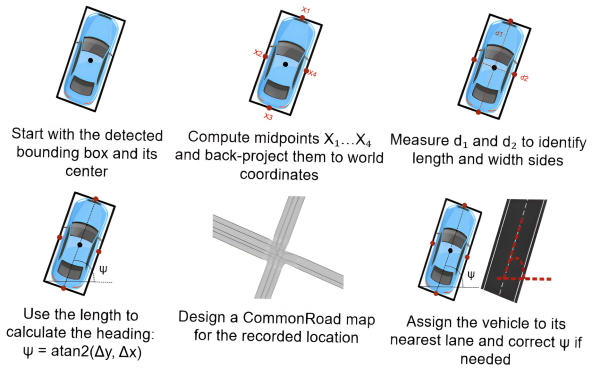

Fig. 5. Overview of the steps for vehicle heading estimation. Δx and Δy are the projections of the length vector onto the x- and y-axes [2].[13]

C. Association Stage

The aim of the association stage is to link detections belonging to the same object to a unique track ID. The trajectory extraction pipeline of the MONA dataset uses the DeepSORT tracker [11]. It consists of a two-stage matching logic. In the first step, a Kalman filter predicts the next position of the track in world coordinates and attempts to link detections to it based on the Mahalanobis distance. In the second step, all unassigned tracks or detections are processed using a fallback guided by the intersection over union (IoU). To avoid ID switches in

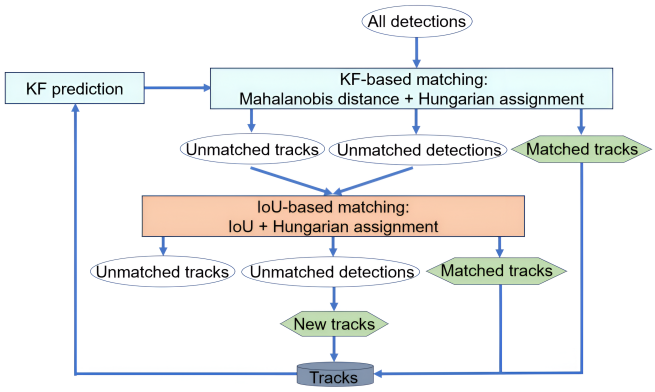

Fig. 6. Overview of the association stage.

crowded scenes or noisy detections, two adjustments are made for bicycles and pedestrians:

- **Kalman Filter Matching Margin:**
  In the Hungarian assignment, a detection is only paired with a track if the best Kalman filter cost significantly exceeds the second-best by at least a specified margin.
- **Distance Gate:**
  If the distance between the detection and the Kalman filter mean in world coordinates exceeds a predefined threshold, the candidate detection is rejected.

IV. CommonRoad Dataset Conversion

The vehicle trajectories from our tracking pipeline require transformation into CommonRoad format for motion planning research. CommonRoad provides standardized scenario representations. Key implementation challenges include maintaining trajectory continuity during state discretization, aligning coordinate systems between tracking output and CommonRoad format, and mapping heterogeneous vehicle types to standardized obstacle representations. Our conversion implementation addresses these challenges through a pipeline that maintains trajectory accuracy while ensuring full CommonRoad compliance, enabling direct integration with motion planning benchmarks and autonomous vehicle testing frameworks.

A. Architecture and Core Components

The conversion system uses a two-layer architecture. A factory wrapper accepts configuration parameters such as input/output directories, time step size, coordinate transformation options, and planning problem settings, then creates a standalone converter instance. The standalone converter performs trajectory-to-scenario conversion and optionally adds planning problems via scenario re-reading and enhancement. This approach separates configuration handling from the conversion logic while providing flexibility for different usage scenarios.

B. Data Processing Pipeline

The converter processes vehicle trajectories in two main stages. The parquet files consumed here are produced by the preceding detection/segmentation/association stage and contain per-frame, columnar trajectory records (e.g., `track_id`, `frame_id`, `x`, `y`, `vx`, `vy`, `psi_rad`, `length`, `width`, `class`). These files provide the canonical per-frame vehicle observations that the converter verifies and transforms into CommonRoad dynamic obstacles. The converter checks that required columns are present and skips files that do not match the expected schema, logging a warning that includes the file name and, when available, the affected `track_id`. Next, the converter reads trajectory data from the parquet files and lanelet/map information from CommonRoad XML files. For each vehicle, trajectory points are sorted by `frame_id`; tracks with fewer than two points are excluded. After conversion to CommonRoad dynamic obstacles, the converter recenters the scenario by subtracting the dataset midpoint

$$(x_c, y_c) = \left(\frac{x_{\min} + x_{\max}}{2}, \frac{y_{\min} + y_{\max}}{2} \right)$$

from every position. The translation is applied using CommonRoad's `translate_rotate()` with zero rotation. Recentering reduces absolute coordinate magnitudes (improving numerical stability) and centers the scene for visualization. Data processing errors are handled via exception handling and are logged with file and track identifiers.

C. Dynamic Obstacle Generation

Vehicle trajectories are converted into CommonRoad dynamic obstacles through state sequence generation. Each vehicle is represented as a rectangular obstacle with dimensions

(length and width) taken from the trajectory data. The converter uses time-varying position, orientation, and velocity for each state.

Vehicle classification is performed by mapping the data source's vehicle type to CommonRoad obstacle types: cars, trucks, bicycles, and buses. The converter processes each vehicle trajectory by sorting data points chronologically. Vehicles with fewer than two data points are excluded since they cannot form a meaningful trajectory. For each valid vehicle, an initial state is created from the first data point, and a trajectory prediction is generated from all data points. Each trajectory state contains position coordinates, velocity magnitude, orientation angle, and a time step index. The resulting dynamic obstacles are added to the scenario with unique obstacle IDs generated by applying an offset to the original track identifiers, avoiding conflicts with map elements.

D. Planning Problem Generation

A planning problem specifies a motion planning task by defining an initial state (starting position, velocity, orientation) and a goal region (desired end state with tolerance intervals). By generating planning problems from recorded trajectories, we create realistic benchmark scenarios for algorithm evaluation.

The implementation selects vehicles with trajectories longer than 5 time steps ($\Delta t = 0.033\,\mathrm{s}$, so 5 steps $\approx 0.165\,\mathrm{s}$), ensuring sufficient trajectory context, and randomly chooses 20% of them as ego vehicles. For each selected vehicle, the initial state is extracted from the first trajectory point. A goal region is constructed from a future state (3–10 steps ahead, ≈ 0.10–$0.33\,\mathrm{s}$), specifying acceptable position ($5\,\mathrm{m}^2$), velocity ($\pm 2.0\,\mathrm{m/s}$), orientation ($\pm 0.1\,\mathrm{rad}$), and forward time tolerance (+10 steps). These planning problems are then used to test motion planning algorithms against realistic driving scenarios.

E. Output Generation and Serialization

The converter produces CommonRoad scenarios in XML format, including the lanelet network, dynamic obstacles, and metadata such as scenario identifiers and time step information.

Output filenames are derived from the input parquet filenames by replacing the substring `trajectories` with `scenario`. For example, `trajectories_123.parquet` becomes `scenario_123.xml`. This preserves the dataset identifier while clearly indicating the file contains a CommonRoad scenario.

Comprehensive logging provides feedback on conversion progress, obstacle generation, and errors for each trajectory, enabling quality assurance and debugging.

F. Converter for the OBACHT Dataset

To improve structure and maintainability, the standalone converter was migrated to the standard CommonRoad dataset pipeline. The migration follows the factory design pattern and integrates the dataset through three dedicated components:

- **`ObachtConverterFactory`:** Creates an interface between the OBACHT dataset and the standard CommonRoad conversion pipeline. It initializes all required components and ensures that preprocessing is completed before scenario generation.
- **`ObachtRecordingGenerator`:** Loads the trajectory files, maps tracks to CommonRoad obstacle types, filters unsupported or invalid tracks, and organizes the data into time windows using a fixed time step.
- **`ObachtMetaScenarioCreator`:** Provides the static road network for each generated time window. The CommonRoad XML map is loaded once and deep-copied for every scenario to ensure independence.

To avoid numerical issues caused by large absolute coordinates in the original dataset, the coordinate transformation was moved to a separate preprocessing step. Both the reference map and trajectory data are transformed before entering the pipeline, resulting in stable and consistent scenario generation.

V. Visualization Toolbox

To support structured validation and debugging of the generated CommonRoad scenarios, a dedicated visualization toolbox was developed. It enables spatial inspection of converted scenarios, temporal analysis of vehicle dynamics, and trajectory evaluation in both Cartesian and curvilinear coordinate systems. The toolbox consists of three main components: scenario rendering, time-based state analysis, and curvilinear trajectory visualization.

A. Scenario Rendering

The `ScenarioVisualizer` renders CommonRoad scenarios from XML files. It enables a structured inspection of both the static road network and the dynamic obstacle behavior after scenario conversion. Two output modes are available:

- **Snapshot mode:** Generates a static PNG image at time step $t = 0$. Dynamic obstacles are disabled to emphasize the static road layout, which supports verification of map alignment, coordinate transformation correctness, and overall spatial consistency.
- **Animation mode:** Generates a GIF file that visualizes scenario progression over time with dynamic obstacles enabled. The animation horizon can be specified manually; otherwise, it is determined from the maximum `final_time_step` among all dynamic obstacles (with a default fallback). Obstacle IDs can optionally be displayed to support tracking of individual vehicles.

To keep the visual output readable, additional rendering elements (e.g., bounding boxes, trajectory traces, and selected lanelet network details) are disabled by default.

B. Time Profiles of State Variables

The `StatePlotter` performs per-obstacle dynamic analysis based on the trajectory state sequences stored in each scenario. Only dynamic obstacles whose type is contained in `VEHICLE_TYPES` are considered. Obstacles without valid trajectory predictions are skipped to ensure robust processing.

For each obstacle, the discrete state sequence is evaluated using the scenario time step Δt. Each state provides the Cartesian position $(x(t), y(t))$ and may additionally contain dynamic quantities such as velocity, orientation, acceleration, and yaw rate.

If certain state variables are not explicitly available, they are computed numerically from the position data. Velocity is derived from the position gradients, orientation from the direction of motion, acceleration from the velocity gradient, and yaw rate from the temporal change of orientation. This fallback strategy ensures consistent and comparable analysis even when state information is incomplete.

Three visualization modes are supported:

- **dynamics:** Velocity and acceleration over time.
- **trajectory:** Yaw rate over time and the Cartesian trajectory in the x–y plane, where the path is color-coded by speed and annotated with start and end markers.
- **all:** Combined view of both representations.

The profiles of each obstacle are processed individually and stored as a separate PNG file. This enables structured inspection of motion continuity, speed profiles, and dynamic consistency across trajectories.

C. Curvilinear Coordinate System Representation

The `CLCSVisualizer` provides a visual representation of vehicle trajectories in a curvilinear coordinate system, enabling structured inspection of longitudinal and lateral motion relative to a reference path.

For this purpose, a reference path is computed from the planning problem and preprocessed to ensure a smooth and numerically stable representation. Based on this path, a curvilinear coordinate system is constructed using the `commonroad-clcs` library [12].

Dynamic obstacle trajectories are then projected from Cartesian coordinates (x, y) into curvilinear coordinates (s, d), where s denotes the arc length along the reference path and d represents the lateral deviation from the centerline. The resulting (s, d) trajectories are visualized to analyze vehicle movement along the road.

Two visualization modes are supported:

- **Combined mode (default):** All trajectories are rendered in a single (s, d) plot. Line segments are color-coded by velocity, providing a compact overview of traffic flow and speed distribution while reducing visual clutter in dense scenarios.
- **Individual mode:** Each vehicle trajectory is plotted separately in the (s, d) plane, including start and end markers as well as an average speed annotation. This mode enables a detailed inspection of individual motion behavior.

The curvilinear representation explicitly separates longitudinal s and lateral d motion components, enabling structured traffic flow analysis.

VI. Drivability Checker

To ensure that converted scenarios are not only visually consistent but also physically valid, an automated drivability validation module was developed. The checker is built on top of the `commonroad-dc` library [13] and verifies the three core drivability criteria:

- Collision freedom,
- Road compliance,
- Kinematic feasibility.

While the underlying validation routines are provided by `commonroad-dc`, additional diagnostic and scenario-level evaluation logic was implemented to support structured debugging and dataset validation.

A. System Architecture

Trajectory data are first standardized and converted into CommonRoad `PlanningProblem` and `Solution` objects. The evaluator then performs collision checking, kinematic feasibility verification, and road compliance testing. Each validation routine is implemented in a separate module to ensure clear separation of concerns. If violations are detected, the results can be visualized to support debugging and analysis.

B. Validation Scopes

Two validation scopes are supported:

- **Ego trajectory validation:** A single selected trajectory is validated with respect to safety, kinematic feasibility, and road compliance.
- **Scenario integrity validation:** All vehicle trajectories in the scenario are evaluated. This includes a global vehicle-vehicle collision scan followed by per-vehicle feasibility and road compliance checks.

C. Vehicle Collision Checking

Collision detection is initially performed using the built-in function from the `commonroad-dc` library. As illustrated in Fig. 7, the internal pipeline consists of a preparation stage followed by a two-stage geometric verification process (broad phase and narrow phase), which finally yields a boolean collision decision.

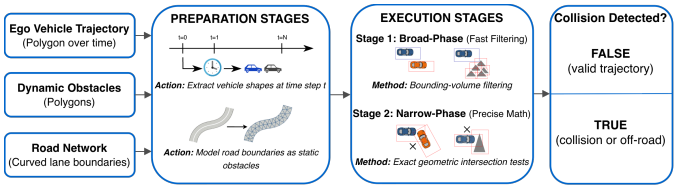

Fig. 7. General collision checking pipeline of the `commonroad-dc` library.

In the collision checking pipeline, vehicles and road boundaries are represented as geometric objects. First, a fast filtering step reduces potential collision candidates using simplified bounding volumes. Then exact geometric intersection tests are executed only for the remaining candidates. The final result indicates whether a collision or off-road violation exists.

Since this function provides only a Boolean outcome and no diagnostic metadata, additional discrete-time procedures are implemented to extract structured diagnostic information. These custom extensions rely on polygon intersection tests using the `Shapely` geometry library.

a) Ego scope: If a collision is reported, a discrete-time analysis is performed. The ego trajectory is reconstructed as a `DynamicObstacle`, and for each time step t its polygonal occupancy is extracted and tested against all dynamic obstacle polygons using `Shapely`'s `.intersects()` method.

A structured crash report is returned containing:

- First collision time step,
- IDs of involved vehicles,
- Collision positions,
- Individual velocities,
- Relative speed.

b) Scenario integrity scope: For global validation, the same discrete-time principle is extended to all vehicles. A direct pairwise comparison would require n^2 checks per time step for n vehicles, which becomes computationally expensive in dense scenarios.

To improve efficiency, an additional broad-phase filtering step is introduced before the exact intersection tests, while the final narrow-phase verification remains identical to the ego case.

Broad phase: At each time step, active vehicle polygons are inserted into a `Shapely STRtree`. The tree organizes polygons using bounding rectangles in a hierarchical spatial index. Each vehicle polygon is fully contained within its axis-aligned bounding rectangle. An overlap of two bounding rectangles therefore only indicates a potential collision but does not guarantee that the actual vehicle polygons intersect, since the true shapes are subsets of these rectangles. The call `tree.query(active_polys)` returns only such candidate pairs, which are then verified in the narrow phase using exact polygon intersection tests.

Narrow phase: For candidate pairs, exact polygon intersection tests are performed using `.intersects()`. Only true geometric overlaps are confirmed to be collisions. As in the ego scope, only the first detected collision frame per vehicle pair is stored to avoid duplicate reports.

This extension preserves detailed diagnostic output while enabling efficient global collision detection in larger scenarios.

D. Road Compliance Check

Road compliance is verified by testing whether the ego vehicle intersects a road boundary obstacle derived from the lanelet network. Instead of evaluating the complete scenario, a reduced *mini scenario* is constructed around the trajectory.

For this purpose, all ego positions are enclosed in an axis-aligned rectangular region. The rectangle is defined by the minimum and maximum x and y coordinates of the trajectory and extended by a fixed safety margin in all directions. Its edges are parallel to the global coordinate axes and its center, length, and width are computed from these bounds. Only lanelets intersecting this rectangular region are selected using `find_lanelet_by_shape()` and copied into a reduced local scenario with simplified connectivity.

A road boundary obstacle is then generated using `create_road_boundary_obstacle()` and tested against the reconstructed ego vehicle by collision checking. If an intersection occurs, the trajectory is classified as off-road.

E. Kinematic Feasibility Check

A collision-free trajectory may still be physically unrealizable. Therefore, kinematic feasibility is evaluated using `solution_feasible()` from the `commonroad-dc` library. The routine checks whether the motion is consistent with the selected vehicle model and remains within its physical limits.

Two representations are supported:

- **Input sequence:** If acceleration and steering values are provided directly, the vehicle is simulated forward and all inputs are verified to remain within admissible physical bounds.
- **State trajectory:** If only states (position, velocity, orientation) are given, the required control inputs between consecutive states are reconstructed and checked against the vehicle's physical constraints.

The trajectory is considered feasible only if all implied or provided inputs stay within physical limits. Otherwise, it is classified as infeasible.

VII. Results

Trajectory Extraction Pipeline Results

Unfortunately, since we do not have ground-truth trajectories, a detailed evaluation of the trajectory extraction pipeline's performance is not possible. Instead, we can present the qualitative results. The outputs of the different stages are shown in Figures 8, 9, and 10. In Figure 8, each detected object is represented by a detection ID, a class, a bounding box, a segmentation mask, and a confidence value. To reduce computation and improve the trajectory extraction pipeline's runtime, we define a detection zone, indicated by the red rectangle. Any detected object outside the detection zone is not considered in the subsequent pipeline steps. Figure 9 illustrates the results of the pose estimation stage. For vehicles, trucks, and buses, each bounding box is rotated by the computed heading. For bicycles and pedestrians, we have visualized the heading angle with an arrow pointing in the estimated direction. The results of the association stage are finally shown in Figure 10. Each object is assigned a unique track ID across all frames. For the evaluation, we set the downscaling factor to 0.35.

Converter Implementation Results

1) Visualization: The conversion pipeline successfully transforms raw trajectory data from parquet files into CommonRoad scenarios.

The conversion preserves trajectory accuracy while adding CommonRoad-specific representations including rectangular obstacle geometries, temporal discretization into discrete time steps, and seamless integration with the road network structure.

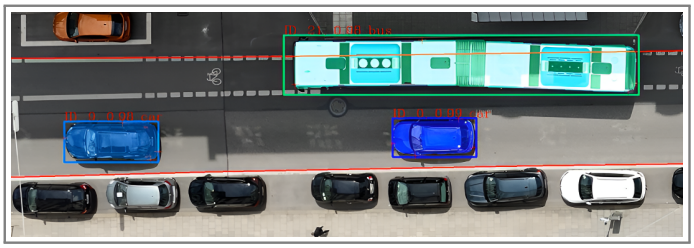

Fig. 8. Visualization of the detection stage.

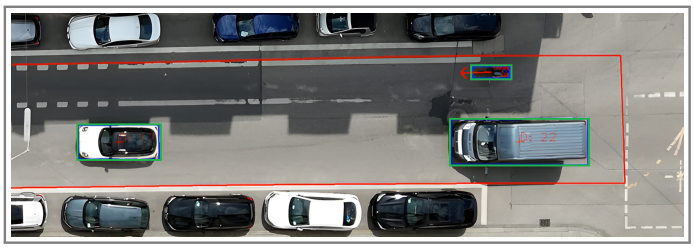

Fig. 9. Visualization of the pose estimation stage.

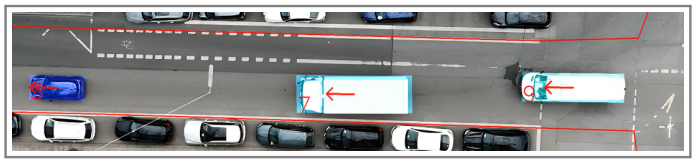

Fig. 10. Visualization of the association stage.

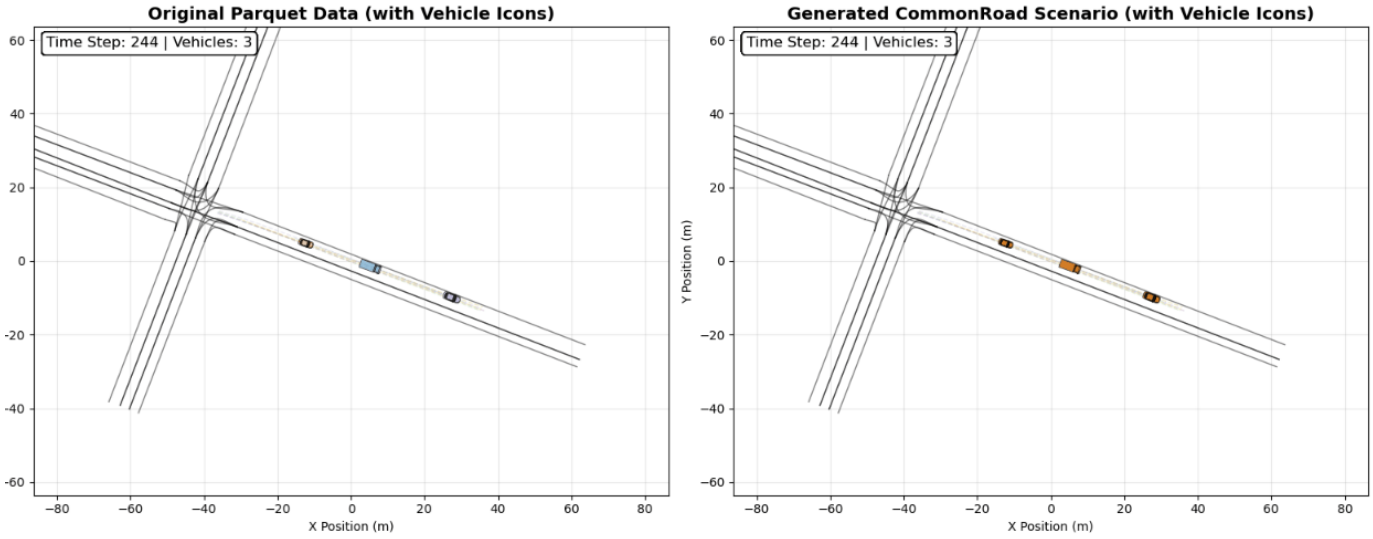

Fig. 11. Conversion result comparison: (left) original tracking data from parquet file showing raw vehicle trajectories, (right) converted CommonRoad scenario with dynamic obstacles, rectangular shapes, and lanelet network integration

All vehicles are correctly classified according to their types and represented with appropriate dimensions. The resulting scenarios are directly compatible with motion planning algorithm evaluation and autonomous vehicle testing frameworks.

2) Trajectory Analysis: Individual vehicle trajectories are extracted and visualized to demonstrate the detailed motion data captured by the detector. Figure 12 shows a representative car trajectory spanning 17.2 seconds with 522 recorded frames. The color gradient encodes instantaneous vehicle speed, revealing acceleration and deceleration patterns. Coordinates are transformed to match the scenario center reference frame for consistency with generated CommonRoad scenarios.

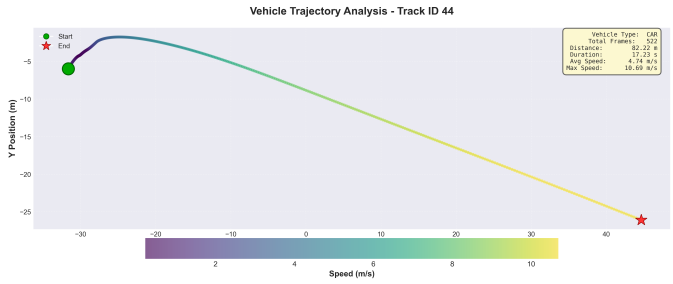

Fig. 12. Exemplary vehicle trajectory from Obacht dataset. Color gradient represents instantaneous speed

Visualization Toolbox Results

1) Scenario Rendering: The scenario rendering module successfully visualizes the converted CommonRoad scenarios. Figure 13 shows a representative snapshot extracted from the

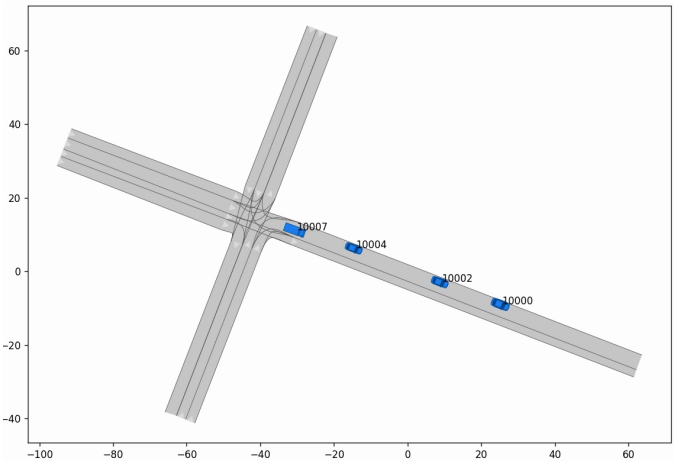

Fig. 13. Snapshot of the animation mode showing dynamic obstacle rendering within the lanelet network. The image represents a single frame extracted from the generated GIF animation.

generated animation.

The lanelet network is shown together with dynamic obstacles rendered using their vehicle shapes. Obstacle IDs are displayed directly in the scene, which makes it easy to track individual vehicles over time. The animation confirms correct temporal progression and proper spatial alignment with the road network. The snapshot further indicates consistent coordinate transformation and accurate vehicle placement within the intersection.

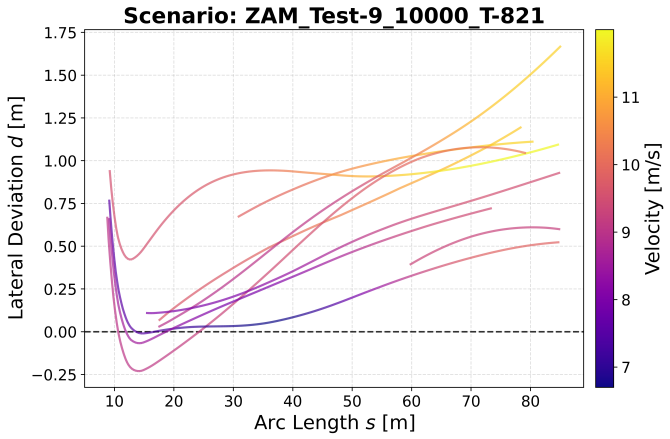

Fig. 14. Curvilinear trajectory representation of all vehicles. Trajectories are projected to (s, d) coordinates and color-coded by velocity.

2) Curvilinear Trajectory Visualization: Figure 14 shows the projection of all dynamic obstacle trajectories into the curvilinear coordinate system (CLCS).

The longitudinal coordinate s denotes the arc length along the reference path, while the lateral coordinate d represents the deviation from the centerline. The dashed horizontal line at $d = 0$ corresponds to the reference path. Velocity is encoded using a continuous color scale. This representation separates longitudinal motion from lateral displacement and enables structured analysis of traffic progression and speed distribution along the road.

The projected trajectories appear continuous and consistent over the full scenario.

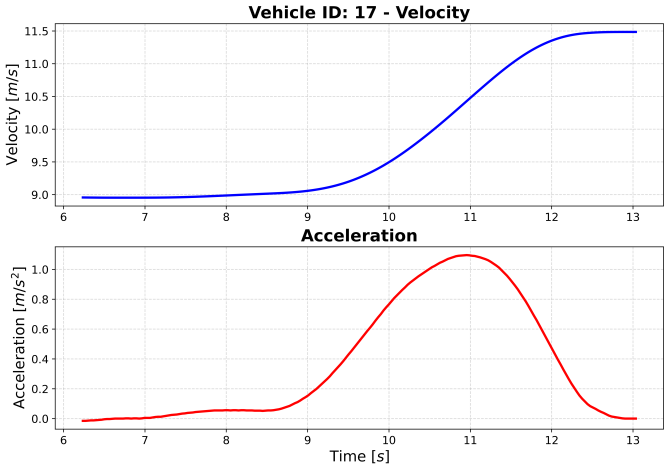

Fig. 15. Dynamic state profiles for vehicle ID 17. Shown are velocity and acceleration over time.

3) Time Profiles of State Variables: Figure 15 shows the temporal evolution of velocity and acceleration for a representative vehicle.

The velocity increases smoothly over time, while the acceleration curve changes continuously without abrupt jumps or unrealistic peaks. The relationship between velocity and acceleration is consistent and physically reasonable.

Overall, the plots confirm plausible vehicle dynamics and allow straightforward inspection of motion consistency over the entire trajectory.

Drivability Checker Results

1) Collision Detection: For the OBACHT scenario evaluated in the *scenario integrity scope*, no vehicle-vehicle collisions were detected. All trajectories were collision-free under global validation.

To demonstrate the diagnostic capabilities of the collision module, Figure 16 shows a representative collision case from a different dataset. The visualization highlights both involved vehicles and reports the first collision frame, collision location, and relative speed.

2) Road Compliance: Within the OBACHT dataset, several vehicles failed the road compliance check. Figure 17 shows a representative violation, where the driven trajectory leaves the valid lane area and intersects the constructed road boundary.

Trajectories that remain within lane boundaries pass the validation, while boundary intersections are classified as violations. The visualization clearly highlights the off-road segment and allows direct inspection of the invalid motion.

3) Kinematic Feasibility: For all evaluated datasets, the kinematic feasibility check passed successfully.

All trajectories remained within the physical limits of the selected vehicle model. No dynamically inconsistent or phys-

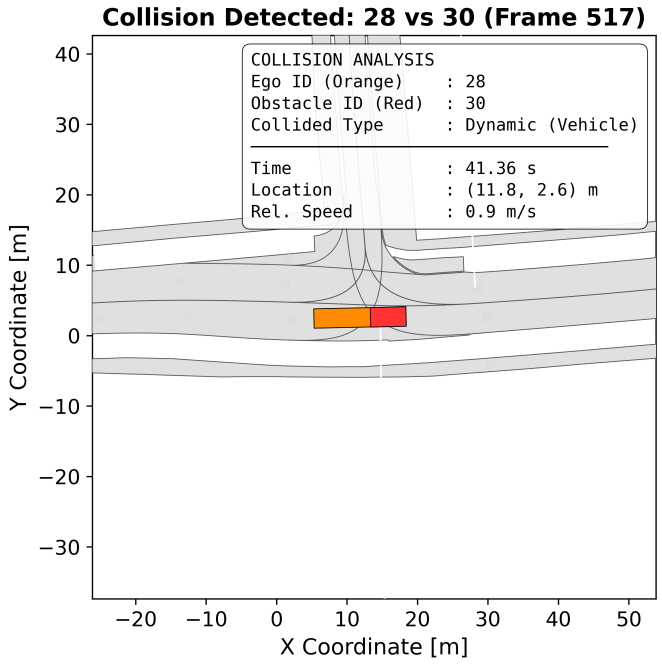

Fig. 16. Detected vehicle-vehicle collision. The ego vehicle (orange) collides with a dynamic obstacle (red). The first collision frame and diagnostic information are displayed.

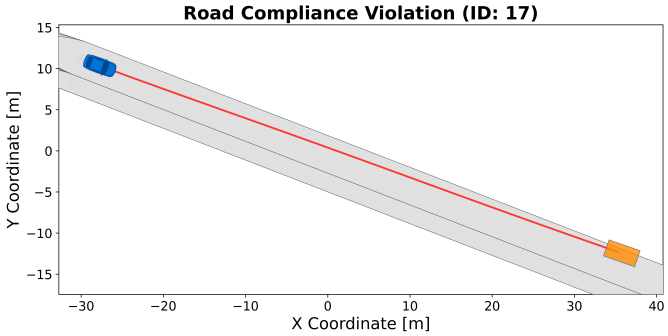

Fig. 17. Road compliance violation for vehicle ID 17. The trajectory intersects the road boundary and is classified as off-road.

ically infeasible motions were detected, indicating that the generated trajectories are physically executable.

4) Computational Performance: The runtime of the validation in the *scenario integrity scope* was evaluated for multiple scenarios with different numbers of vehicles (see Table I).

Collision checking accounts for the largest share of computation time in scenarios with many vehicles due to repeated geometric intersection tests.

Kinematic feasibility verification remains lightweight, as it is evaluated individually per trajectory.

Road compliance checking shows low execution times in most cases, with runtime depending on the spatial extent of the evaluated region.

Overall, the drivability checker operates in the millisecond range and scales reliably with increasing scenario complexity.

VIII. Conclusion

This work presents a trajectory extraction pipeline from drone recordings for autonomous bus development. By adapting the MONA framework with TPH-YOLOv5 for small object detection and MobileSAM for efficient segmentation, we address the challenges of top-down views and crowded scenes. The pipeline includes enhanced tracking with additional constraints to reduce ID switches, and improved pose estimation using rotated bounding boxes for vehicles and motion-based direction for pedestrians and bicycles.

The extracted trajectories are successfully converted into CommonRoad-compliant scenarios through a two-stage architecture that handles coordinate transformation, dynamic obstacle generation, and optional planning problem creation. The results demonstrate effective detection, tracking, and conversion across all road user types, producing scenarios ready for motion planning research and autonomous vehicle testing.

To support structured validation, a dedicated visualization toolbox was developed, enabling scenario rendering, temporal state analysis, and curvilinear trajectory representation for detailed inspection of spatial alignment, motion consistency, and speed behavior. In addition, an integrated drivability checker verifies collision freedom, road compliance, and kinematic feasibility, ensuring that the generated scenarios are physically consistent and computationally efficient to validate.

References

[1] L. Gressenbuch, K. Esterle, T. Kessler, and M. Althoff, "MONA: The Munich Motion Dataset of Natural Driving," in *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, 2022, pp. 2093–2100.

[2] M. Althoff, M. Koschi, and S. Manzinger, "CommonRoad: Composable benchmarks for motion planning on roads," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 719–726.

[3] R. Krajewski, J. Bock, L. Kloeker, and L. Eckstein, "The highD Dataset: A Drone Dataset of Naturalistic Vehicle Trajectories on German Highways for Validation of Highly Automated Driving Systems," 2018. [Online]. Available: https://arxiv.org/abs/1810.05642

[4] J. Bock, R. Krajewski, T. Moers, S. Runde, L. Vater, and L. Eckstein, "The inD Dataset: A Drone Dataset of Naturalistic Road User Trajectories at German Intersections," *CoRR*, vol. abs/1911.07602, 2019. [Online]. Available: http://arxiv.org/abs/1911.07602

[5] R. Krajewski, T. Moers, J. Bock, L. Vater, and L. Eckstein, "The rounD dataset: A Drone Dataset of Road User Trajectories at Roundabouts in Germany," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, 2020, pp. 1–6.

[6] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft COCO: Common Objects in Context," 2015. [Online]. Available: https://arxiv.org/abs/1405.0312

[7] X. Zhu, S. Lyu, X. Wang, and Q. Zhao, "TPH-YOLOv5: Improved YOLOv5 Based on Transformer Prediction Head for Object Detection on Drone-captured Scenarios," in *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, 2021, pp. 2778–2788.

[8] P. Zhu, L. Wen, D. Du, X. Bian, H. Fan, Q. Hu, and H. Ling, "Detection and tracking meet drones challenge," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 11, pp. 7380–7399, 2021.

[9] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick, "Segment Anything," *arXiv:2304.02643*, 2023.

[10] C. Zhang, D. Han, Y. Qiao, J. U. Kim, S.-H. Bae, S. Lee, and C. S. Hong, "Faster Segment Anything: Towards Lightweight SAM for Mobile Applications," *arXiv preprint arXiv:2306.14289*, 2023.

TABLE I
RUNTIME OF THE DRIVABILITY CHECKS IN MILLISECONDS (MS)

Scenario	Obstacles	Collision (ms)	Feasibility (ms)	Road Compliance (ms)
ZAM_Test-9_10000_T-821	9	134.8	595.3	3.6
DEU_MUC-1_1_T-899	22	569.8	25.6	1867.0
DEU_LocationBUpper2-1_21_T-900	25	397.0	3.6	0.9
CHN_TIANJIN-8021_10_T-899	48	448.5	754.7	15.7

[11] N. Wojke, A. Bewley, and D. Paulus, "Simple Online and Realtime Tracking with a Deep Association Metric," 2017. [Online]. Available: https://arxiv.org/abs/1703.07402

[12] G. Würsching and M. Althoff, "Robust and efficient curvilinear coordinate transformation with guaranteed map coverage for motion planning," in *2024 IEEE Intelligent Vehicles Symposium (IV)*, 2024, pp. 2694–2701.

[13] C. Pek, V. Rusinov, S. Manzinger, M. C. Üste, and M. Althoff, "Commonroad drivability checker: Simplifying the development and validation of motion planning algorithms," in *2020 IEEE Intelligent Vehicles Symposium (IV)*, 2020, pp. 1013–1020.