Department of Informatics
Technical University of Munich

Master's Thesis in Informatics

Monocular 3D Object Detection Using HD Maps

Monokulare 3D Objekterkennung mit HD-Karten

Supervisor Prof. Dr.-Ing. habil. Alois C. Knoll

Advisor Walter Zimmer, M.Sc.

Author Joseph Birkner

Date May 15, 2023 in Garching

Disclaimer

I confirm that this Master's Thesis is my own work and I have documented all sources and material used.

Garching, May 15, 2023

(Joseph Birkner)

Abstract

Due to their low cost and high output information density, monocular RGB cameras are a popular sensor choice for many perception tasks, including 3D object detection. The low cost of the sensor is especially important for the large-scale deployment of road-side infrastructure, as envisioned by the Providentia project. Prior work within the project determined that a two-stage detection approach using the bottom contour of $2D$ instance masks is viable in a highway setting. However, the applicability of the $2D \rightarrow 3D$ lifting approach via the mask in urban settings has been unclear. In this work, we propose an augmented L-Shape fitting algorithm which solves the monocular 3D object detection task for urban settings. The algorithm is augmented using HD maps to inform likely heading values, and tracking, to further improve the heading value selection. We evaluate our algorithm on the Providentia Intersection Scenario dataset. The augmented algorithm improves over basic L-Shape fitting by 22% in *mAP*, 8.36% in *IoU*, and reaches 5.37° of orientation error and $0.9m$ in translation error. We conclude that the approach is useful for real-time application in road-side infrastructure sensing tasks.

Zusammenfassung

Aufgrund ihrer geringen Kosten und hohen Informationsdichte sind monokulare RGB-Kameras eine beliebte Sensorwahl für viele Wahrnehmungsaufgaben, einschließlich der 3D-Objekterkennung. Die niedrigen Kosten des Sensors sind besonders wichtig für den großflächigen Ausbau von straßenseitiger Infrastruktur, wie er im Rahmen des Providentia-Projektes vorgesehen ist. Frühere Arbeiten im Rahmen des Projekts ergaben, dass ein zweistufiger Erkennungsansatz, der die untere Kontur von $2D$-Objektinstanzmasken verwendet, für den Einsatz an Autobahnen geeignet ist. Unklar blieb jedoch die Anwendbarkeit dieses $2D \rightarrow 3D$ Lifting-Ansatzes in städtischen Szenarien. In dieser Arbeit schlagen wir einen erweiterten L-Shape-Fitting Algorithmus vor, der die Aufgabe der monokularen 3D-Objekterkennung auch in diesen Szenarien löst. Der Algorithmus wird durch HD-Karten erweitert, um wahrscheinliche Ausrichtungswinkel zu ermitteln, und durch Tracking, um die Selektion der Ausrichtungswinkel weiter zu verbessern. Wir evaluieren unseren Algorithmus anhand des urbanen Providentia Kreuzungs-Szenario-Datensatzes. Der erweiterte Algorithmus verbessert das grundlegende L-Shape-Fitting um 22% in *mAP*, 8.36% in *IoU*, und erreicht sowohl 5.37° Rotationsfehler als auch $0.9m$ Positionsfehler. Wir kommen zu dem Schluss, dass der Ansatz für die Echtzeitanwendung für 3D-Sensoren an der straßenseitigen Infrastruktur geeignet ist.

Contents

Acknowledgements

I would like to thank my thesis advisor Walter Zimmer for his relentless pursuit of perfection for our monocular perception algorithms, as well as Suren Sritharan, wo completed some essential groundwork towards adopting SORT and L-Shape-Fitting into our codebase.

Chapter 1

Introduction

1.1 The Providentia++ Project

The Providentia Project [Krä+22] aims to create a large-scale Intelligent Infrastructure System (IIS) that can provide autonomous vehicles and traditional vehicles with complementary information about each road user and the overall traffic situation. By observing and detecting all road users from multiple perspectives, an IIS can greatly extend a vehicle's perception range, enabling it to plan its maneuvers more safely and proactively. The primary goal of Providentia is to improve the safety and comfort of autonomous vehicles by reducing their reliance on on-board sensors and providing them with a better and spatially extended understanding of their surrounding scene.

There are several related projects in the field of IIS, including the Test Area Autonomous Driving Baden-Württemberg, the MEC-View project, and the local highway operator in Austria's road operator system [CK21]. While these projects have similar goals, they are smaller in scale than Providentia and use different sensor types. Additionally, many research contributions propose methods of making algorithmic use of the information provided by an IIS or optimizing their function.

Providentia uses a combination of multimodal sensors, including radars, LiDARs, and cameras. The radars and LiDARs provide distance measurements and can detect objects that are out of the camera's field of view. The cameras capture a more distant environment than the LiDARs, but objects that are too far away appear small on the image and cannot be reliably detected. The combination of these sensors provides redundant road coverage with overlapping field of views, accurate calibration, and robust detection and data fusion algorithms.

The RGB camera sensor captures visual information about the environment, which is used to generate the digital twin. It can capture a more distant environment than the LiDARs but is prone to severe occlusions due to its low perspective. The RGB camera sensor is essential for fusing multiple sensor perspectives and updating the digital twin, which includes information such as position, velocity, vehicle type, and a unique identifier for every observed vehicle. By providing this digital twin to an autonomous driving research vehicle, Providentia demonstrates that it can be used to extend the limits of the vehicle's perception far beyond its on-board sensors. The update rate for the digital twin depends on the fusion-setup and the type of the used object detection network, and it can vary between 13.1Hz to 24.6Hz depending on the sensor setup.

Figure 1.1: Overview of the Providentia A9 Test Stretch (Graphic produced using Google Earth).

1.2 The A9 Testfield

At the heart of the Providentia Project is an expansive test area with multiple sensor stations. The sensor stations are each equipped with RGB Camera, LiDAR and RADAR sensors. These are used to test the sensor calibration-, road user detection-, sensor-fusion-, tracking-, and communication-solutions that are developed as part of the project. An overview of the test area along the A9 highway near Garching Hochbrück, Germany is provided in Figure 1.1.

This work focuses on the RGB Camera sensors, which are installed at the designated stations. The goal of this work is to develop a perception algorithm which can calculate a three-dimensional digital twin of any visible road user from the two-dimensional RGB camera input frames. Previous work on the Monocular 3D Object Detection (*Mono3D*) task in the scope of Providentia focused on solving this task for the sensor stations along the A9 highway [Blu22]: `S40` and `S50`. In this work, we aim to generalize the Mono3D solution to cover the more urban sensor stations, in particular the cameras which are installed at the `S110` intersection. This urban intersection setting is more challenging because it does not allow for a fixed orientation of road users to be assumed by the monocular detector. Instead, the detector must induce a contextual bias for each road user to determine their orientation angle.

Crucially, a high-definition (HD) map of the test area was also developed as part of the Providentia project using the OpenDRIVE format [DSG10]. An overview of the map with a more detailed view of the S110 intersection is provided in Figure 1.2. Within this project, we aim to explore how the HD map of the test area can serve as auxiliary sensory input to monocular detector. We hypothesize that the highly detailed map might provide the monocular detector with information about viable road user orientations.

For the demonstration purpose of this work, we focus our efforts on two cameras which are installed at the S110 sensor station: `S110-S1` and `S110-S2`. Both cameras have a native resolution of 1920x1200 pixels and run at 60 Hz. However, the `S110-S2` camera has a much

longer detection range, as it is angled towards the horizon, whereas the S110-S1 camera is focused downwards onto the intersection.

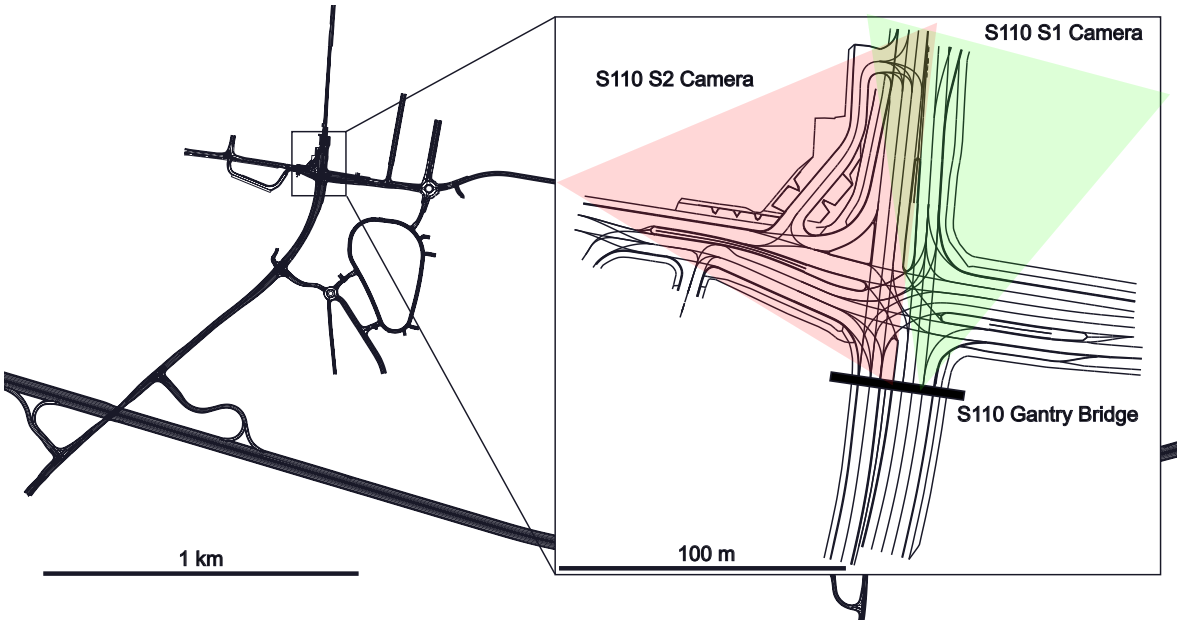


Figure 1.2: Overview of the OpenDRIVE map of the Providentia test area, with zoomed in cut-out of the S110 intersection with its S1 and S2 cameras.

1.3 The Providentia Intersection Scenario Dataset

To evaluate any object detection method, labeled data are required. For this purpose, the *Providentia Intersection Scenario* dataset was developed under the umbrella of Providentia. The dataset annotates both LiDAR point-clouds with categorized 3D object bounding boxes, and RGB Camera frames with 2D cuboid labels. Specifically, four scenes in the dataset provide the crucial combination of 3D LiDAR bounding box labels for 2D camera frames. This labeling pair allows us to evaluate our monocular 3D object detector. The four scenes in the dataset which do provide this required annotation mode are listed in Table 1.1.

Table 1.1: Intersection Scenario dataset scenes which provide camera frames annotated with 3D detection bounding boxes from LiDAR labels.

Scene	Perspective	#Frames	#Detections	Weather	Time of Day
Scene 4	S110 Camera S1	300	2918	Sunny	Daytime
Scene 4	S110 Camera S2	300	2810	Sunny	Daytime
Scene 5	S110 Camera S1	300	2958	Sunny	Daytime
Scene 5	S110 Camera S2	300	2302	Sunny	Daytime
Scene 8	S110 Camera S1	1200	8661	Sunny	Daytime
Scene 8	S110 Camera S2	1200	11064	Sunny	Daytime
Scene 9	S110 Camera S1	619	2327	Rain	Night
Scene 9	S110 Camera S2	619	4668	Rain	Night

The dataset frame-rate is 10 Hz. The combination of 3D LiDAR Labels and RGB camera frames introduces an additional challenge in the form of synchronization delay —there may be up to 200 ms of difference between the 2D RGB frame and the 3D LiDAR annotation timestamps. We will discuss this problem in more detail in Chapter 5. Each road user is also annotated with a category; one of `CAR`, `BUS`, `TRUCK`, `TRAILER`, `VAN`, `MOTORCYCLE`, `BICYCLE`, `PEDESTRIAN`, `EMERGENCY_VEHICLE`, or `OTHER`.

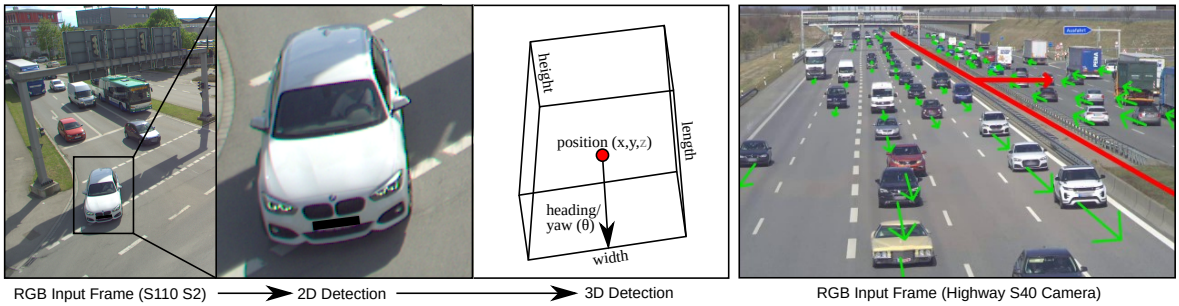


Figure 1.3: On the left: General dataflow in the Monocular 3D Object Detection task. For each instance of a recognized object in the RGB input frame, the detector must estimate a 3D pose parameter-set. The more variables the detector must estimate, the harder the task becomes. For example, elevation (i.e., the position z component) may be omitted. **On the right:** Figure 4.4 from [Blu22]. Frame of a highway scene. In such a scenario, the detector may also omit the calculation of the heading (yaw) orientation angle and assume a fixed value.

1.4 Monocular 3D Object Detection

1.4.1 Variables to Estimate

Monocular 3D Object Detection (*Mono3D*) is the process of detecting three-dimensional objects from a single two-dimensional RGB camera output frame. However, the term *3D Object Detection* might be misleading in the context of this work. The number of variables which a "3D" object detector may derive for an object far surpasses three, as can be seen in the following Table 1.2.

We observe that a minimal 3D Object Detector should actually estimate six dimensions per object: Birds-eye-view (BEV) positions X/Y, BEV size L/W, heading angle θ and category C. Earlier work on the *Mono3D* task for the Providentia project [Blu22] focused on solving this task for the highway scenario. On top of the minimum six variables, this early detector also calculated the object height. However, as the early detector focused on the highway scenario, it was able to assume a fixed value for the θ variable. This is illustrated on the right side of Figure 1.3. As this work strives to generalize the *Mono3D* solution beyond the highway, it must treat the θ value as non-fixed. Interestingly, we facilitate this task by also estimating the identity I variable, which then also allows us to estimate the planar speed $\delta X/\delta Y$ variables. So, we will end up with a 10-D detector in this work (although we do not make an attempt to evaluate the speed estimates).

1.4.2 Two-Stage Detection

While many approaches are viable, the earlier Providentia *Mono3D* detector by [Blu22] used a so-called two-stage detection model. As this work builds on top of [Blu22], we use the same

Table 1.2: Parameterization options for the output of a Monocular 3D Object Detector per object.

Variable	Description	Optional
X/Y	Position along the longitudinal/lateral axes.	**No**
L/W	Extent (length/width) along the longitudinal/lateral axes.	**No**
$\delta X/\delta Y/\delta Z$	Speed. Derivatives of the position/elevation variables.	**Yes**
Z	The elevation of the object over the road surface.	**Yes**
H	The height of the object.	**Yes**
θ	Yaw (heading) angle, determines travel direction.	**No**
ϕ/γ	Tilt/Roll angles, e.g. terrain slope or crash scenarios.	**Yes**
I	Identifier for the object across multiple frames.	**Yes**
C	Category of the object, e.g. `CAR` or `PEDESTRIAN`.	**No**

approach. Generally, the two-stage detector splits the detection task into two separate steps: a $2D$ instance segmentation step, and a $3D$ lifting step. There are again many implementation options for these individual steps. The steps are illustrated in Figure 1.4

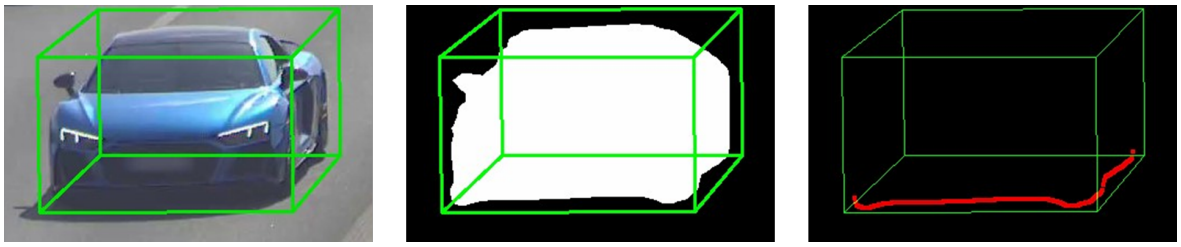


Figure 1.4: Two-stage detection from camera image (left) via instance segmentation (middle) and $2D \rightarrow 3D$ lifting via the instance mask bottom contour (right). Graphic from [Blu22].

Within the taxonomy provided by [Ma+22], these are also called *Result-based Lifting Methods*. The primary advantage of this idea is that the 3D detector can benefit from highly polished, interchangeable, off-the-shelf instance segmentation models. For example, the [Blu22] detector exchanged *Mask-RCNN* [He+17] in favor of *Yolact-Edge* [Liu+21] in the first stage for performance reasons. In this work we switch again, from *Yolact-Edge* to *YOLOv7* [WBL22] to improve the detection quality. Another benefit is that stage two, the $2D \rightarrow 3D$ lifting stage, can be independently optimized, as the problem shifts from identifying the image areas occupied by relevant objects to estimating the three-dimensional properties of these objects. The method proposed by [Blu22] of lifting the $2D$ mask into $3D$ space via a $3D$ back-projection of the instance mask's bottom contour is applied and refined in this work.

The biggest hurdle towards a correct $3D$ vehicle pose estimation in an urban setting was identified by [Blu22] as the θ (heading/orientation) value calculation. The process by which an orientation angle can be estimated from the bottom contour is called *L-Shape-Fitting* [Zha+17]. The vehicle bottom contour, which we want to use as an input to this calculation, can be quite noisy for many reasons, such as visual obstructions, upstream instance mask detection errors, or cropping at the image border (just to name a few). Hence, an additional contextual bias is required for each object to correctly estimate its orientation angle.

1.4.3 Use of Neural Networks

A Neural Network would most likely be a great choice for an algorithm to estimate not just the vehicle orientation θ, but to perform the whole $2D \rightarrow 3D$ lifting stage. For example, *UrbanNet* [CW21] implemented this approach with a Neural Network trained on synthetic data. For this work, we are staying with a "rule-based", non-neural approach for three reasons:

1. Research continuity: The bottom-contour-based two-stage approach proposed by [Blu22] is undoubtedly viable, there is no obvious reason to change the research direction.
2. Explainability, Extendability, Maintainability: The "rule-based" approach allows for diagnosing and fixing particular failure modes. This is much harder with a neural network.
3. Performance: A neural network estimator would most certainly require GPU resources to perform adequately in a real-time setting as required by the Providentia IIS. On the other hand, the non-neural approach can function purely on the CPU, leaving the sparse GPU resources to other tasks.

For these reasons, this work explores exclusively non-neural "Software 1.0" [1] solutions for the $2D \rightarrow 3D$ lifting stage.

1.5 Relevance of HD Maps

High-definition (HD) maps model the road network down to the detail of individual driving lane geometries. In this work, we hypothesize that the HD map can serve as a useful additional sensor to the *Mono3D* detector. In particular, we explore whether the HD map can serve as a viable source of vehicle orientation (heading) values. In the first step, we developed a method to derive the likely heading value at a particular position on the road surface from the geometry of the enclosing lane boundaries. This method is described with more detail in Chapter 4.7. By converting the calculated $\overrightarrow{xyz}$ heading vectors at each position to RGB colors, we can proof the general idea. This is visualized in Figure 1.5.

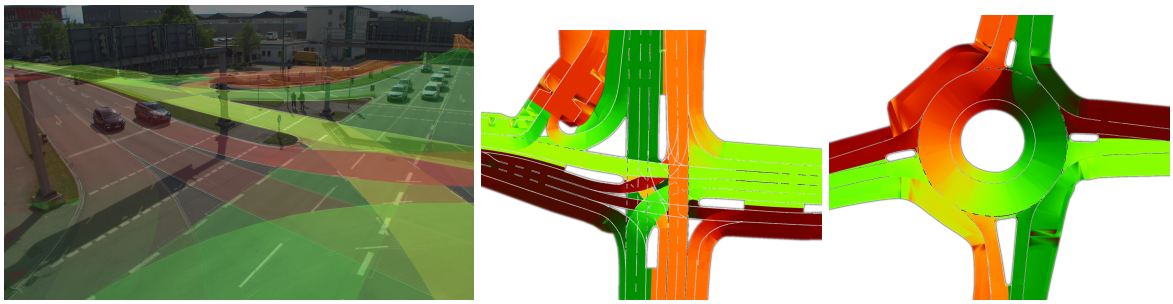


Figure 1.5: Visualisation of heading vectors as RGB values, smoothly interpolated from the lane boundary geometry. **Left:** Colored heading overlays for the S110-S2 camera perspective. In the bottom-right corner, there are four overlapping lanes. **Middle:** RGB heading visualization for the whole S110 intersection. **Right:** RGB heading visualization for a roundabout.

From this proof-of-concept visualization of the heading vectors, it becomes apparent that the map-derived headings may serve well as an additional per-pixel feature for the *Mono3D*

[1] https://karpathy.medium.com/software-2-0-a64152b37c35

detector. But it also becomes apparent, that there may always be many possible overlapping heading options provided by the HD map, as there may always be many overlapping lanes —especially in the context of urban intersection scenarios. The left-most image of Figure 1.5 illustrates this problem quite well. Here we see color-coded headings overlaid on top of a frame from the `S110-S2` camera. Towards the bottom-right corner of the image, there are locations with up to four overlapping lanes, which provide four distinct heading options! Therefore, an additional information source is needed to resolve ambiguities among multiple heading options.

1.6 Tracking Enters the Picture

1.6.1 Screen-Space Tracking

In face of the mentioned uncertainties when deciding between ambiguous heading options, it might be useful to consider previously known spatial orientations of a vehicle. Consider a scene such as the one shown in Figure 1.6.

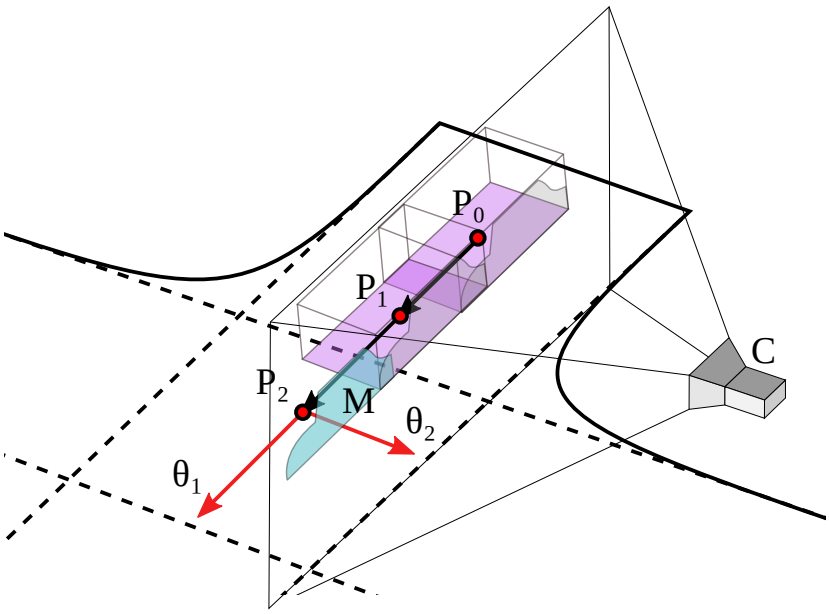


Figure 1.6: Visualization of a *Mono3D* detection scenario where tracking resolves a heading ambiguity.

In this scene, the *Mono3D* detector C observes the instance mask of vehicle M. Through the screen-space bounding box of M, the detector can associate this instance mask with detections from previous frames —P_0 and P_1. The mask M also allows for the calculation of the 3D position P_2. The detector then performs a lookup for θ (heading) options at location P_2 and obtains two possible headings: θ_1 and θ_2. As the detector must now pick one of these heading options, it may use the given knowledge of previous historical positions (P_0 and P_1) for M to bias towards θ_1 as the more likely orientation. This is because the detector assumes that vehicles move in a continuous direction more often than not, and erratic turns are less likely than forward motion.

To express this intuition formally, given a detection M at time t which is associated with

a set of historical positions $P = \{p_t, \ldots, p_{t-T}\}$, we wish to pick $\theta(M)$ such that

$$\sum_{\delta=1}^{\delta=T} |\theta(M) - \text{atan2}(p_t - p_{t-\delta})|$$

is minimal. In this work, the storage and retrieval of historical positions for the screen-space bounding box of a detection is accomplished using the *SORT* [Bew+16] object tracking algorithm.

1.6.2 Birds-Eye-View Tracking

We use screen-space tracking to aid in the ranking of heading options. However, the same tracking algorithm can also be used at a later stage for a different purpose: In Birds-Eye-View *(BEV)* tracking, objects are tracked in $3D$ space on the X/Y axes rather than in camera image space. This is where the *Kalman-Filters* [WB+95] which are at the heart of the *SORT* tracking algorithm really develop their potential as denoisers of physical measurements. Fully implemented, they could denoise every variable of a tracked detection. However, in this work, we will only use them to stabilize position and speed estimates.

1.7 Inherent Functional Limitations

As the origins, goals and methods of this work are now introduced, we can already identify functional limitations which are going to be inherent to our approach. These limitations must be tackled by future work (see Chapter 7).

1. **Bad detection quality for road users with highly cropped or obstructed instance masks**: Both the bottom-contour based L-Shape-Fitting algorithm, and the estimation of the vehicle position from the instance mask require an unobstructed full view of the road user to function optimally. Any obstruction of the view will degrade the estimation quality. Such an obstruction may be a stationary object in front of the road user, another road user, a weather condition like snow/rain/fog, or simply the field-of-view *(FOV)* limit of the sensor.

2. **Bad detection quality for road users in legal or physical peril:** Our *Mono3D* detection approach assumes that road users do not fly, do not lie on their side, do not move sideways, and are always aligned with a legal traffic flow direction as parsed from the HD map. Therefore, the 3D pose estimation for any road user which violates one of these assumptions will be as bad as the the day this road user is probably experiencing.

1.8 Research Objective

In summary, the research objective for this work is to address the truthfulness of the following hypothesis: *A two-stage Mono3D detector with an instance-mask bottom-contour based estimator in the second stage can function well within an urban intersection setting if the estimator is assisted by additional information from an HD map*.

1.9 Contributions

This work presents the following research contributions:

1. An augmented *L-Shape-Fitting* algorithm which supports tracking and HD Map confidence inputs.
2. A lookup strategy for vehicle bottom contours within the HD Map to derive yaw options histograms with associated confidence values.
3. An HD map rasterization algorithm to support fast bulk lookup of heading options for a vehicle bottom contour.
4. A formula to derive the plausibility of a yaw hypothesis for a detected vehicle from a series of historical positions for the same vehicle.
5. An algorithm to calculate the physical position and height of a vehicle, given the instance mask height, physical width/length, and predicted orientation angle.
6. An algorithm to calculate 3D bounding boxes for Vulnerable Road Users (Pedestrians and Bicycles) from their Instance Mask Bottom Contour.

Chapter 2

Related Work

2.1 Earlier Work Within the Providentia Project

This work is deeply rooted in the earlier groundwork which was laid out for the *Mono3D* task in highway scenarios [Blu22] in the scope of the Providentia project. Our earlier work took inspiration from the *Cooperative Vehicle Infrastructure System* [Guo+21] in the design and implementation of the two-stage detector based on the bottom contour of the vehicle instance mask, but also differs in key aspects. The detection flow of our early *Mono3D* highway detector is illustrated in Figure 2.1.

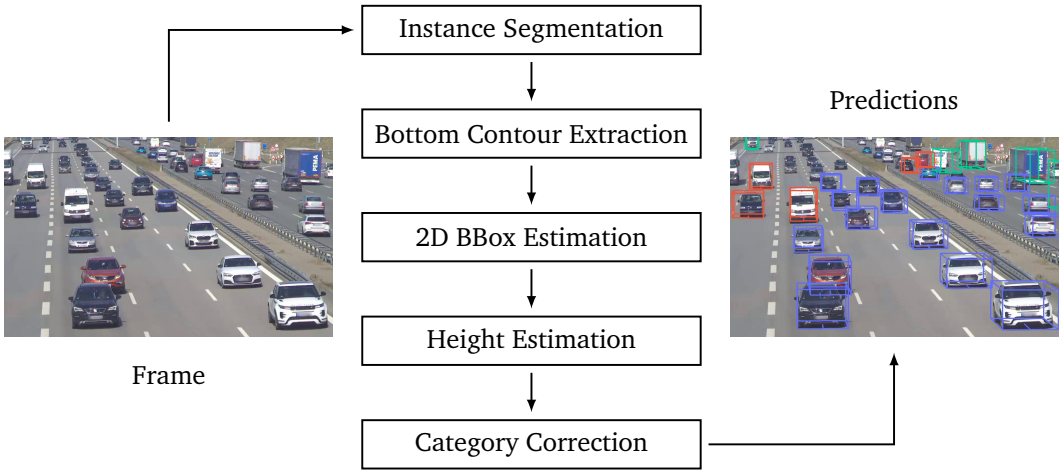


Figure 2.1: Figure 4.1 from [Blu22], illustrating the stages of the monocular 3D detection process for highway scenes.

Note, that while this work adopts the first two elements of the detection flow —*Instance Segmentation* and *Bottom Contour Extraction* —we have completely reengineered all subsequent processes. This earlier detector works well on highways because it assumes a fixed orientation for all vehicles. This also allows for a very straight-forward linear equation to solve for vehicle heights given their distance from the camera, as well as their instance mask heights. In this work, the vehicle orientation is dynamically derived from the HD map and the vehicle's positional history. The height estimation algorithm is revised accordingly.

2.2 Detection of Vehicles in Cooperative Vehicle Infrastructure Systems

The earlier Providentia highway detector was inspired by the *Cooperative Vehicle Infrastructure System (CVIS)* [Guo+21], which also uses a two-stage detector with a $2D \rightarrow 3D$ lifting stage based on the vehicle bottom contour. Their approach is illustrated in Figure 2.2.

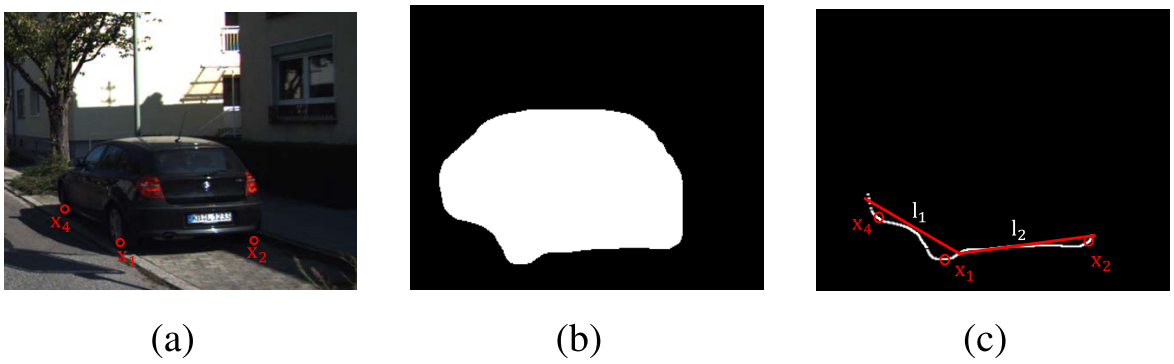


Figure 2.2: Figure 1 from [Guo+21], illustrating their approach to keypoint estimation based on two lines which are regressed to the vehicle bottom contour: *(a)* RGB image; *(b)* vehicle segmentation mask; *(c)* the contour point of the bottom edge of the vehicle and the contact points between vehicle and ground are represented by white dots and red circles.

While their approach has provided the basis for our research on *Mono3d* in the Providentia project, it exhibits some flaws which we are hoping to have overcome (to some extent) in this work:

1. Previous frames are not considered when a vehicle pose is estimated, potentially leading to bad yaw angle choices in ambiguous situations.
2. They do not mention the effects of shadows or noise due to obstructions or weather on their bottom contour line regression algorithm. The L-Shape-Fitting algorithm which is used in this work might be more robust in such situations, especially since we also apply *DBSCAN*-based outlier filtering on the bottom contour.
3. Their height estimation solution assumes that the closest vehicle top corner and bottom corner share the same distance from the camera, which leads to *leaning* boxes in traffic monitoring situations where the camera is stationed highly above the vehicles.
4. They do not consider non-vehicle *Vulnerable Road Users (VRUs)* in their detector, such as bicyclists or pedestrians. This work is also detecting VRUs.

Finally, no code was published alongside their approach. This inherently necessitates research to reproduce their results.

2.3 The L-Shape Fitting Method for Vehicle Pose Detection

Branching off *CVIS*, this work (like [Blu22]) makes use of *L-Shape-Fitting (LSF)* to estimate vehicle size and orientation from the instance mask bottom contour. This approach can also be interpreted as a *pseudo-lidar* [Ma+22] approach because the back-projection of the instance mask bottom contour yields a (very flat) point-cloud. Underlining the "LiDAR-esque" origins of the LSF algorithm is the fact that it was introduced in the scope of an object

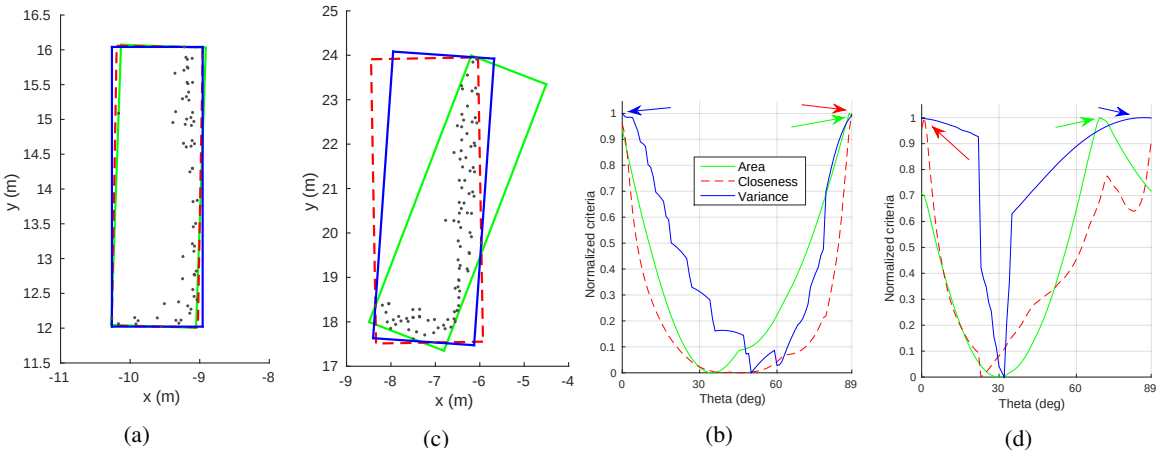


Figure 2.3: Figure 2 from [Zha+17]. In (a) and (c), the grey dots represent the laser range scan points and the rectangles in green, red, and blue are the fitted rectangles by using criteria area minimization, closeness maximization, and variance minimization. The normalized scores for the three criteria over the searched directions are plotted in (b) and (d), respectively. In example (a), the fitting results from the three criteria are very similar, and the maxima of the three curves in (b) are very close (marked by arrows and achieved at 88◦, 89◦, and 0◦, respectively). In example (b), the fitting result from the area criteria is different from the other two, and its maximum in (d) is away from the other two (achieved at 69◦, 1◦, and 86◦, respectively).

detection method from LiDAR measurements, in the work *The L-Shape Fitting Method for Vehicle Pose Detection from LiDAR* [Zha+17]. Their method is illustrated in Figure 2.3.

By regressing a rectangular shape to the bottom contour, the LSF algorithm can estimate orientation, BEV position and BEV size simultaneously. The regression is performed by maximizing one of three error criteria: `Area`, `Closeness` or `Variance`. The algorithm and the criteria are explained in section 3.7. In this work, we further adapt and develop this approach to stabilize size and heading estimates in complex traffic observation scenarios using tracking and the HD map.

2.4 TrafficNet

The *TrafficNet* detector [RAM21] is a very thorough two-stage monocular traffic monitoring solution, encompassing both vehicle and VRU detection, tracking, speed estimation, and even road geometry prediction from satellite images. This is illustrated in Figure 2.4.

As can be seen from the right side of Figure 2.4, they make use of road curb geometry which is extracted from a satellite image to make estimates of vehicle headings, substituting the role of the HD map in this work. Note that heading estimation via the most proximate curb would not be applicable in this work, as curbs within a complex intersection with many overlapping lanes are not a good indicator of orientation. However, their approach alleviates the need for rather hard-to-source high-definition maps, which is a point of weakness in this work. Another strong point of *TrafficNet* is their extensive use of Kalman Filters [WB+95] to de-noise the vehicle poses and category estimations. They also fine-tuned a custom instance mask segmentation model in the first detector stage based on YOLOv5 [Joc+20]. This is done both to detect additional object categories, and to speed up inference by removing convolutions which are only needed to detect large objects. One weak point of their work is the extensive use of “magic values” in the second detector stage: They make use of predefined sizes based on vehicle categories and calculate 3D object height based on a precalculated factor of 0.6 from the instance mask height. Also, unfortunately, no code is publicly

Figure 2.4: Figures 12 and 13(c) from [RAM21], illustrating their approach to heading estimation based on the geometry of proximate road curbs which have been extracted from sattelite imagery. This is analogous to our use of the HD map in this work (to some extent).

available for *TrafficNet*.

2.5 UrbanNet: Urban Maps for Long Range 3D Object Detection

Another case of a two-stage object detector which uses Urban maps to augment the performance of the $2D \rightarrow 3D$ lifting-stage is *UrbanNet* [CW21]. This solution is unique, because it applies the HD map as an auxiliary feature to a small Feed-Forward Neural Network. The architecture is illustrated in Figure 2.5.

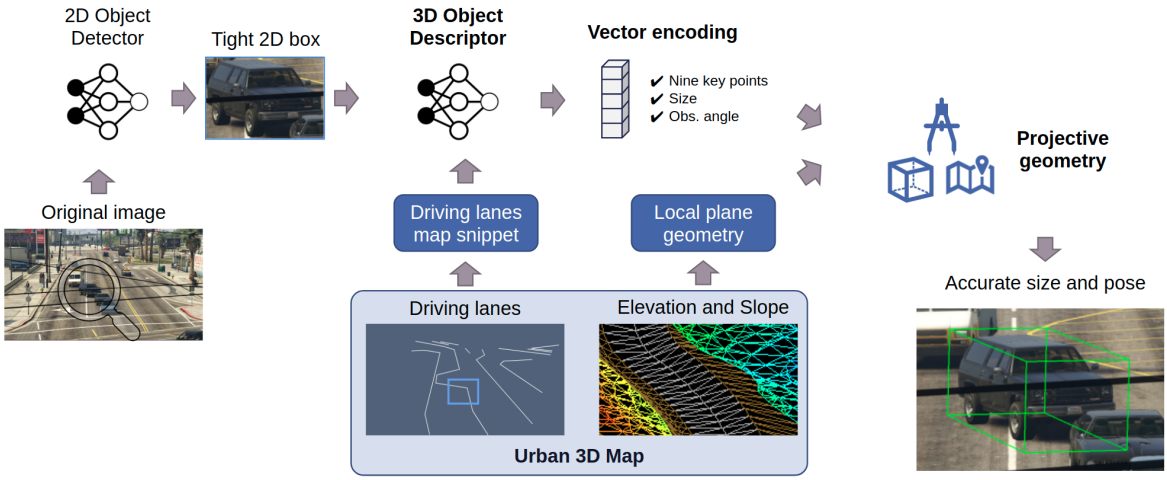


Figure 2.5: Figure 2 from [CW21], describing how the Urban HD map assists the 3D object descriptor estimation model. Center-lines are *painted* as a per-pixel feature into the 3D object descriptor network's single input image.

This approach provides an interesting way to remediate some of the shortcomings of L-Shape-Fitting in the second detector stage. The neural network can learn to recognize obstructed road user poses, or even highly uncommon poses, such as a car laying on its side in a crash scenario. This is not possible with a bottom-contour based detector. Furthermore, the runtime performance of this approach might be overall better than ours, as there is no requirement for the first detector stage to output and compute expensive per-instance image masks. Notably, they also incorporate slope estimation into their prediction, which is espe-

cially useful for long-range vision in mountainous terrain. The bottleneck in this case is the availability of training data, something which is solved in UrbanNet by training and testing exclusively on artificial computer-generated road scenes.

2.6 Cooperative Roadside Vision Systems in Complex Scenarios

Finally, the *Complex Scenario Cooperative Roadside Vision System* [Mas+21] is related as prior work due to its two-stage monocular 3D object detection architecture and usage of HD maps in all stages of the detection processes.

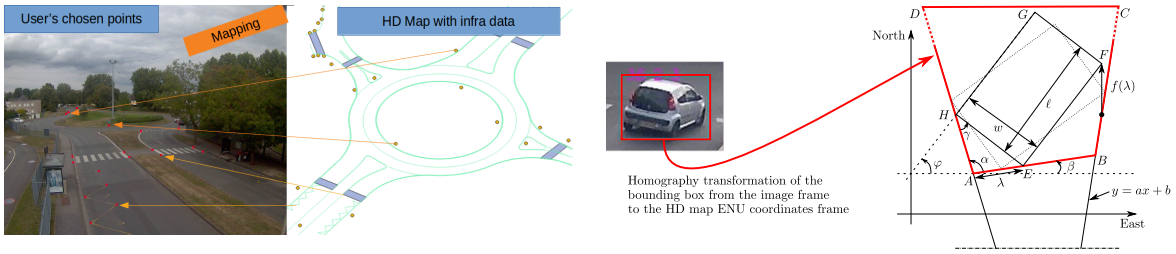


Figure 2.6: Figures 2 and 3 from [Mas+21], describing how the HD map is used for calibration (left) and how the vehicle size is calculated from its 2D bounding box (right).

Their system uses the HD map for calibration (as illustrated in Figure 2.6), heading calculation, and tracking. On the other hand, their paper is not clear on when and how exactly they make use of the HD map for heading estimation. In some or most cases, they fall back to a projection of the 2D screen-space bounding box onto the 3D ground plane, from which a vehicle's BEV width/length and heading may be calculated. They do not make any attempts at calculating vehicle heights or detecting VRUs.

2.7 Survey Studies

In the conclusion of this related work study, several survey papers were consulted, some of which shall be highlighted here for reference. First, the survey *3D Object Detection from Images for Autonomous Driving* [Ma+22] was a great resource to learn a taxonomy for the multitude of architectural approaches towards the *Mono3d* task. Furthermore, the survey on *Object Detection in Traffic Videos* provides an in-depth overview of the spectrum of techniques which are used in 2D object detection neural networks, such as YOLO [WBL22]. Research on *Infrastructure-Based Object Detection and Tracking for Cooperative Driving Automation* [Bai+22] is very connected to this work, and the cited survey includes many of the previously mentioned related works. Finally, the *Review on Cooperative Perception and Control Supported Infrastructure-Vehicle Systems* [Yu+22] is the only survey which mentions HD maps as an auxiliary data source and shared spatial reference frame for cooperating autonomous vehicles.

Chapter 3

Technical Foundations

This chapter provides additional technical details for specific algorithms and techniques that are later referenced in our System Design (Chapter 4). Feel free to skip this chapter if you are roughly familiar with the following methods: 2*D* object instance segmentation, the *OpenDRIVE* and *Lanelet2* map data formats, the *Robot Operating System (ROS)*, the *DBSCAN* point clustering algorithm, the *SORT* object tracking algorithm, the *L-Shape-Fitting* algorithm (as previously mentioned in Section 2.3), and common object detection evaluation metrics, such as *mAP* and *IoU*.

3.1 The YOLACT Instance Segmentation Model

YOLACT (You Only Look at Coefficients) [Bol+19] is a real-time instance segmentation method that focuses on processing static images. It simplifies the instance segmentation problem by separating it into two parallel tasks, enabling faster processing while maintaining competitive accuracy. The functionality of *YOLACT* can be described in three main steps (as illustrated in Figure 3.1):

1. **Generating Prototype Masks:** *YOLACT* employs a fully convolutional network called *ProtoNet* to generate a fixed set of K prototype masks, which are used as the base for constructing the final object instance masks. These prototype masks are shared across all object instances in the image.

2. **Predicting Per-instance Mask Coefficients and Bounding Boxes:** In parallel to generating prototype masks, *YOLACT* predicts per-instance mask coefficients and bounding boxes for potential objects using an anchor-based approach. Each anchor is associated with a mask coefficient vector, which has the same length as the number of prototype masks (K). Additionally, the model predicts the class probabilities and bounding box coordinates for each anchor.

3. **Assembling Final Instance Masks:** To obtain the final instance masks, *YOLACT* linearly combines the prototype masks using the predicted mask coefficients. The model selects the top scoring instances based on their class probabilities and applies a non-maximum suppression (NMS) algorithm to remove duplicate detections. The result is a set of instance masks, each associated with a specific object class and bounding box.

YOLACT's architecture consists of a feature backbone (e.g., *ResNet* [TAL16] or *MobileNet* [How+17]), a Feature Pyramid Network (FPN) for multiscale feature extraction, and separate prediction heads for class, bounding box, and mask coefficients. This architecture

allows it to achieve real-time instance segmentation on static images, with a good balance between speed and accuracy. However, it is designed for large GPUs, such as the *TitanX* or *RTX-2080-Ti*, and its performance on lower-power edge devices is limited without further optimization or adaptations, which is where *YolactEdge* [Liu+21] comes in to address these limitations.

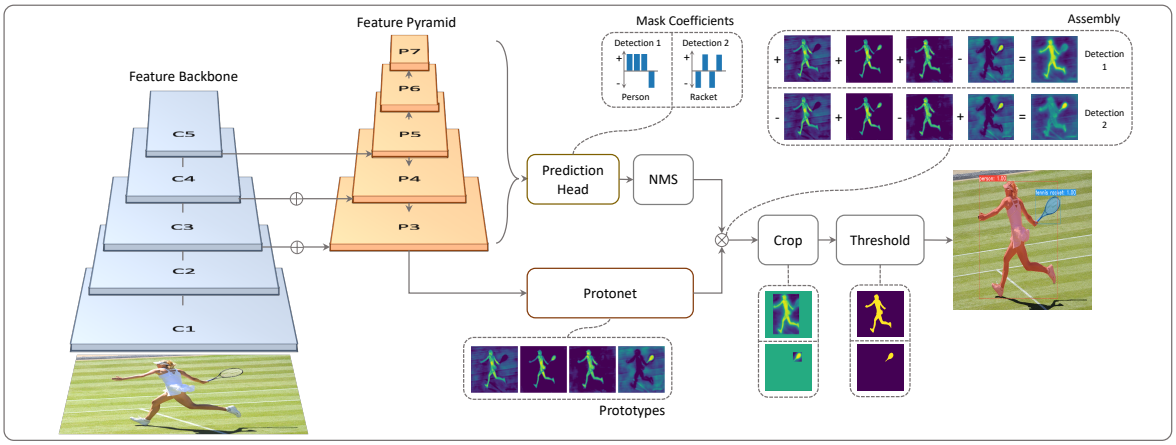


Figure 3.1: Figure 2 from *YOLACT*. The figure shows how generated prototype masks are combined with instance-specific coefficients to produce the final instance masks.

YolactEdge is a real-time instance segmentation method designed for edge devices, focusing on video processing. Its functionality can be broken down into two main improvements over the original *YOLACT* method:

1. ***TensorRT* Optimization:** *YolactEdge* utilizes NVIDIA's *TensorRT* [Van16] inference engine to optimize the neural network. *TensorRT* provides mixed-precision support, optimal tensor layout, layer fusion, and kernel specializations. It quantizes model weights to `INT8` or `FP16` precision, which can speed up the processing while preserving accuracy. *YolactEdge* explores the optimal mix between INT8 and FP16 weights for different model components, maximizing speed without significant degradation in accuracy. The *TensorRT* optimization results in around a 4x improvement in speed when working with static images.

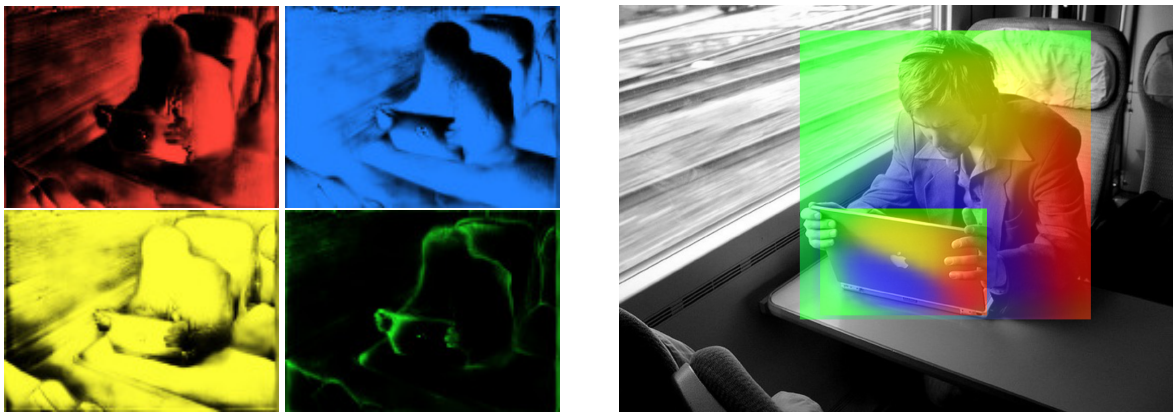
2. **Exploiting Temporal Redundancy in Video:** *YolactEdge* takes advantage of the temporal redundancy in videos, which means that neighboring frames in a video sequence are often highly correlated. Instead of computing expensive backbone features for every frame, YolactEdge divides the frames into two groups: keyframes and non-keyframes. For keyframes, the model computes all backbone and pyramid features. For non-keyframes, only a subset of features is computed, while the rest are transformed from the temporally closest previous keyframe.

By combining *TensorRT* optimization and exploiting temporal redundancy, YolactEdge can achieve real-time instance segmentation on edge devices such as Jetson AGX Xavier with a high frame rate and competitive accuracy. This makes it an ideal solution for applications like robotics, autonomous driving, security, and augmented reality that require real-time processing and low latency.

3.2 The YOLOv7 Instance Segmentation Model

YOLOv7 [WBL22], as a state-of-the-art real-time object detection model, demonstrates significant quantitative improvements in object detection performance. It not only excels in this primary task but also showcases its adaptability and effectiveness in related computer vision tasks, such as instance segmentation.

To achieve high-performance real-time instance segmentation, *YOLOv7* is integrated with BlendMask, a technique introduced in the paper "BlendMask: Top-Down Meets Bottom-Up for Instance Segmentation". BlendMask builds upon the foundation laid by *YOLACT*, but with a key difference in its approach to blending masks within bounding boxes. While *YOLACT* predicts a single scalar coefficient for each prototype mask, BlendMask predicts a low-resolution $(7 * 7)$ attention map for the same purpose, as described in Figure 3.2.



(a) Bottom-Level Bases **(b)** Top-Level attentions

Figure 3.2: Figure 3 from *BlendMask*, showcasing how it replaces *YOLACT's* Prototype Masks with Bottom-Level Bases which are combined using per-instance Top-Level Attention maps rather than coefficients.

This attention map, which functions as a high-dimensional feature, is attached to each bounding box. The blending of masks using the attention map results in a more precise and refined instance segmentation. To leverage *YOLOv7* for instance segmentation, the *YOLOv7* object detection model is fine-tuned on the *MS COCO* [Lin+14] instance segmentation dataset. The resulting *YOLOv7*-mask model achieves a Mask AP (average precision) of 43.1%, while *YOLACT* achieves a Mask AP of 35.4% when using a *ResNet-101* backbone. The architecture of *YOLOv7*-mask, as well as some of the qualitative corresponding results, are illustrated in Figure 3.2.

3.3 Map Data Formats

This work depends on HD maps to inform viable heading choices for road user detections. As map data formats for this task, both the *OpenDRIVE* [DSG10; AUK18] and *Lanelet2* [Pog+18] map data formats are considered. In both *OpenDRIVE* and *Lanelet2* formats, lane geometry representation is crucial for accurate road modeling and navigation. *OpenDRIVE* is an XML-based data format that describes road networks hierarchically. It consists of roads, lanes, junctions, objects, and signals, with each road having a unique identifier and geometric information. Lanes are classified into driving lanes, sidewalks, and shoulders, while junctions define connections between roads. *Lanelet2*, on the other hand, uses a combination of XML and OSM formats for data representation. It is based on the concept of "lanelets", which

are individual driving lanes with associated traffic rules. *Lanelet2* includes left and right boundaries, regulatory elements, and routing graphs.

In *OpenDRIVE*, lane geometry is represented using a combination of planar and lateral geometric information. The geometry of a road is defined by a reference line, which is a parametric curve. This curve is specified using a series of points and can be described by different types of geometry, such as straight lines, spirals, arcs, and clothoids. The reference line represents the road's centerline, and the lanes are defined relative to it. Lanes in *OpenDRIVE* are divided into three categories: driving lanes, sidewalks, and shoulders. Each lane has attributes such as width, type, and road marks. The lateral position of a lane is described by a set of functions, which determine the lane's width as a function of its longitudinal position along the road. This information, along with the road's reference line, is used to compute the geometry of individual lanes.

Lanelet2 uses a simpler approach for representing lane geometry. It employs polygons, with each lanelet defined as a drivable polygon consisting of left and right boundaries. The boundaries are sequences of nodes (points with longitude and latitude), which form linear or curved segments. The nodes are ordered along the lanelet's direction, and the shape of the lanelet is derived by connecting corresponding nodes from the left and right boundaries. Lane boundaries in *Lanelet2* can be solid or dashed lines, indicating whether lane changes are allowed. Unlike *OpenDRIVE*, which uses parametric curves for defining road geometry, *Lanelet2* uses a more straightforward representation based on sequences of points. This approach can be less accurate in some cases, but it simplifies map data handling and provides better compatibility with OpenStreetMap data.

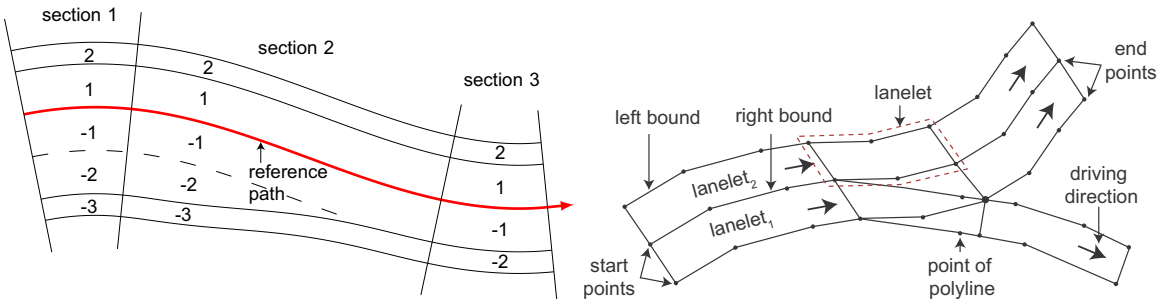


Figure 3.3: Figures 2 and 3 from [AUK18]. The left-hand-side image shows a typical *OpenDRIVE*-based model with multiple lane-sections based on a common continuous reference line. The left-hand-side image shows the simpler modeling approach of *Lanelet2*, where lanes are based on self-contained pre-triangulated shapes.

In summary, *OpenDRIVE* represents lane geometry using parametric curves and a reference line, while *Lanelet2* uses polygonal shapes defined by sequences of nodes. *OpenDRIVE* offers more accurate road geometry, but *Lanelet2's* simpler representation can be easier to work with and integrate with other mapping data sources like OpenStreetMap. This modeling difference is also highlighted in Figure 3.3. In this work, we are compelled to use *OpenDRIVE* as the input format for the HD map, as this format was used by Providentia's map data supplier. However, the *Lanelet2* format would be more suitable, as their approach to lane modeling is directly compatible with our heading interpolation algorithm, as described in Section 4.7.

3.4 The Robot Operating System

The *Robot Operating System (ROS)* [Qui+09] is a flexible software framework for developing robotic applications. Its main goal is to simplify the creation of complex and robust robot

behavior across various platforms.

ROS follows a graph architecture, where modular components called nodes communicate with one another through a publish-subscribe messaging model. Nodes are single-purpose components responsible for specific tasks, such as controlling a sensor or processing data. They exchange information by sending messages over named channels called *topics*. Another central component in *ROS* is the Master, which manages the entire system by allowing nodes to find each other, register topics and services, and maintain a list of active nodes.

The framework provides several advantages for robotics applications, including hardware abstraction, low-level device control, efficient message passing, package management, and a rich ecosystem of tools and libraries. These features make it easier for developers to build distributed robotics applications and promote modularity and code reusability. In our case, we divide separate detector stages into different nodes, as described in Section 4.1.

3.5 SORT Object Tracking

The *Simple Online and Realtime Tracking (SORT)* [Bew+16] algorithm is an object tracking method that is specifically designed for tracking multiple objects in real-time. It employs a combination of detection and tracking to maintain the identity of objects as they move across video frames. *SORT* is lightweight and computationally efficient, making it suitable for real-time applications.

At the core of the *SORT* algorithm is the *Kalman Filter* [WB+95], a recursive estimation technique that helps to predict the state of a dynamic system over time, even in the presence of noise. In the context of object tracking, the *Kalman Filter* is used to estimate the position, velocity, and other parameters of objects in the video frames.

Here is a step-by-step description of the *SORT* algorithm and its use of *Kalman Filters*:

1. **Object Detection:** First, an object detection algorithm, such as YOLO (You Only Look Once) or Faster R-CNN, is applied to the input video frames to identify objects of interest and generate bounding box coordinates for each detected object.

2. **Initialization:** For each detected object, a *Kalman Filter* is initialized with the bounding box coordinates as the initial state. The state vector typically includes the position (x, y) and velocity (vx, vy) of the object's center. Additionally, a unique ID is assigned to each object.

3. **Prediction:** For each object, the *Kalman Filter* predicts its state in the next frame. This is done by applying a state transition model to the current state, which usually involves updating the position based on the velocity.

4. **Data Association:** In the subsequent frame, the newly detected objects are matched with the predicted states of the existing tracked objects. This association is achieved using a distance metric, typically the Intersection over Union (IoU) or the Mahalanobis distance. A bipartite graph matching algorithm, such as the Hungarian algorithm [Kuh55], is employed to find the optimal assignment between detections and tracked objects.

5. **Update:** Once the data association is completed, the *Kalman Filter* is updated with the new measurements (bounding box coordinates) of the associated detected objects. This step incorporates the new observations into the existing state estimate, considering both the prediction and the measurement uncertainties.

6. **Handling Occlusions and New Objects:** If a tracked object is not detected in the new frame, its *Kalman Filter's* state is still updated based on the prediction alone. If an object is missing for a certain number of consecutive frames, it is removed from the tracking list. Conversely, if a new object is detected, a new *Kalman Filter* is initialized and added to the tracking list.

7. **Output:** The final output of the *SORT* algorithm is a list of tracked objects, each with a unique ID and an updated bounding box for each frame.

In summary, the *SORT* algorithm uses *Kalman Filters* to predict and update the state of objects in a video sequence, maintaining their identities while tracking their positions and velocities over time. The data association step ensures that the correct detections are matched with the corresponding tracked objects, making the tracking robust and efficient.

3.6 The DBSCAN Algorithm

DBSCAN, which stands for *Density-Based Spatial Clustering of Applications with Noise* [Sch+17], is a popular unsupervised machine learning algorithm designed for cluster analysis. It works by identifying dense regions in the input data, separating them into distinct clusters, and treating low-density regions as noise.

DBSCAN has the following key functionalities:

1. **Density-based clustering:** The algorithm groups data points based on their density in the feature space. A dense region is defined as an area with a high concentration of data points, while a low-density region has fewer data points.

2. **Automatic detection of the number of clusters:** Unlike some other clustering algorithms, like K-means, *DBSCAN* does not require the user to specify the number of clusters beforehand. The algorithm automatically determines the number of clusters based on the input data.

3. **Robustness to noise:** *DBSCAN* can identify and separate noise from the data. Noise is defined as data points that do not belong to any dense region and are not part of any cluster.

4. **Handling arbitrary shapes:** *DBSCAN* can identify and create clusters with varying shapes and sizes, unlike some other clustering algorithms that assume clusters have a spherical or circular shape.

The *DBSCAN* algorithm works in the following steps:

1. **Initialize:** Two main parameters need to be defined: ϵ, which is the radius around a data point, and N_{min}, the minimum number of data points required to form a dense region.

2. **Iterate through the data points:** For each unvisited data point in the dataset, perform the following steps:

   Mark the current data point as visited.

   Find all neighboring data points within the ϵ radius.

If the number of neighbors is greater than or equal to N_{min}, mark the current data point as a core point and create a new cluster. Recursively add all the connected neighbors (directly or indirectly) to the cluster using a depth-first search approach.

If the number of neighbors is less than N_{min}, mark the current data point as noise.

3. **Cluster assignment:** Assign data points to their respective clusters. Core points are assigned to a specific cluster, border points are assigned to the nearest core point's cluster, and noise points are not assigned to any cluster.

The *DBSCAN* algorithm is particularly useful for datasets with spatial or geographical data, as well as for datasets with complex structures and noise. In the case of this work, the *DBSCAN* algorithm is applied to denoise vehicle bottom contours which have been projected out from the vehicle's instance mask into 3*D* space. This is further explained in Section 4.4.

3.7 The L-Shape Fitting Algorithm

The *L-Shape-Fitting* algorithm is used to find the best-fitting rectangle for a segmented cluster of points. The algorithm assumes that the vehicle being tracked can be approximated by an L-Shape rectangle model, which consists of two perpendicular lines that intersect at a corner. The goal is to find the optimal disjunction of the m points into two sets (P and Q) and the optimal parameters (θ, c_1, c_2) for the two perpendicular lines corresponding to the points in P and Q, respectively.

Algorithm 2 Search-Based Rectangle Fitting

Input: range data points $X \in R^{n\times 2}$
Output: rectangle edges $\{a_i x + b_i x = c_i | i = 1, 2, 3, 4\}$
$Q \leftarrow \emptyset$
for $\theta = 0$ **to** $\pi/2 - \delta$ **step** δ **do**
  $\hat{e}_1 \leftarrow (\cos\theta, \sin\theta)$ ▷ rectangle edge direction vector
  $\hat{e}_2 \leftarrow (-\sin\theta, \cos\theta)$
  $C_1 \leftarrow X \cdot \hat{e}_1^T$ ▷ projection on to the edge
  $C_2 \leftarrow X \cdot \hat{e}_2^T$
  $q \leftarrow$ CalculatecriterionX(C_1, C_2)
  insert q into Q with key (θ)
end for
select key (θ^*) from Q with maximum value
$C_1^* \leftarrow X \cdot (\cos\theta^*, \sin\theta^*)^T, C_2^* \leftarrow X \cdot (-\sin\theta^*, \cos\theta^*)^T$
$a_1 \leftarrow \cos\theta^*, \quad b_1 \leftarrow \sin\theta^*, \; c_1 \leftarrow \min\{C_1^*\}$
$a_2 \leftarrow -\sin\theta^*, \quad b_2 \leftarrow \cos\theta^*, \; c_2 \leftarrow \min\{C_2^*\}$
$a_3 \leftarrow \cos\theta^*, \quad b_3 \leftarrow \sin\theta^*, \; c_3 \leftarrow \max\{C_1^*\}$
$a_4 \leftarrow -\sin\theta^*, \quad b_4 \leftarrow \cos\theta^*, \; c_4 \leftarrow \max\{C_2^*\}$

Algorithm 3 Area Criterion.

function CalculateArea(C_1, C_2)
  $c_1^{max} \leftarrow \max\{C_1\}, \quad c_1^{min} \leftarrow \min\{C_1\}$
  $c_2^{max} \leftarrow \max\{C_2\}, \quad c_2^{min} \leftarrow \min\{C_2\}$
  $\alpha \leftarrow -(c_1^{max} - c_1^{min}) \cdot (c_2^{max} - c_2^{min})$
  return α
end function

Algorithm 4 Closeness Criterion.

Parameter: d_0
function CalculateCloseness(C_1, C_2)
  $c_1^{max} \leftarrow \max\{C_1\}, \quad c_1^{min} \leftarrow \min\{C_1\}$
  $c_2^{max} \leftarrow \max\{C_2\}, \quad c_2^{min} \leftarrow \min\{C_2\}$
  $D_1 \leftarrow \arg\min_{v\in\{c_1^{max}-C_1, C_1-c_1^{min}\}} ||v||_{l_2}$
  $D_2 \leftarrow \arg\min_{v\in\{c_2^{max}-C_2, C_2-c_2^{min}\}} ||v||_{l_2}$
  $\beta \leftarrow 0$
  for $i = 1$ **to** length(D_1) **step** 1 **do**
    $d \leftarrow \max\{\min\{D_{1(i)}, D_{2(i)}\}, d_0\}$
    $\beta \leftarrow \beta + 1/d$
  end for
  return β
end function

Algorithm 5 Variance Criterion.

function CalculateVariance(C_1, C_2)
  $c_1^{max} \leftarrow \max\{C_1\}, \quad c_1^{min} \leftarrow \min\{C_1\}$
  $c_2^{max} \leftarrow \max\{C_2\}, \quad c_2^{min} \leftarrow \min\{C_2\}$
  $D_1 \leftarrow \arg\min_{v\in\{c_1^{max}-C_1, C_1-c_1^{min}\}} ||v||_{l_2}$
  $D_2 \leftarrow \arg\min_{v\in\{c_2^{max}-C_2, C_2-c_2^{min}\}} ||v||_{l_2}$
  $E_1 \leftarrow \{D_{1(i)} | D_{1(i)} < D_{2(i)}\}$
  $E_2 \leftarrow \{D_{2(i)} | D_{2(i)} < D_{1(i)}\}$
  $\gamma \leftarrow -$ variance$\{E_1\} -$ variance$\{E_2\}$
  return γ
end function

Figure 3.4: Algorithms 2-5 from *L-Shape-Fitting* [Zha+17]. Algorithm 2 shows the main theta search loop, while Algorithms 3-5 are possible implementations of the `CalculateCriterionX` function.

The algorithm uses a search-based approach (see Algorithm 2 of Figure 3.4) to find the best-fit rectangle approximately. It iterates through all possible directions of the rectangle and for each direction, it calculates the distances of all the points to the rectangle's four edges. Based on these distances, the points are split into P and Q, and the corresponding errors

are calculated as the objective function. After iterating through all directions, the algorithm selects the optimal direction which achieves the smallest error, and fits the rectangle based on that direction.

The algorithm offers three different criteria for selecting the best-fitting rectangle: rectangle area minimization, points-to-edges closeness maximization, and points-to-edges squared error minimization. Each criterion can be chosen to play the role of the `CalculateCriterionX` function in the algorithm provided in Figure 3.4. The choice of criterion depends on the specific requirements of the application.

The experimental results show that the algorithm is highly accurate and efficient, with a small mean and standard deviation of the estimation error. Albeit computationally the most expensive, the variance criterion achieves the highest correctness, while the area minimization criterion sometimes fails to estimate the correct heading.

However, the algorithm may not always achieve perfect results. For instance, the algorithm may fail to estimate the correct heading if the scan points are too sparse, or if there are points outside or inside of the L-Shape model. This is where the previously mentioned (see Section 3.6) *DBSCAN* [Sch+17] algorithm comes in to denoise the bottom contour points. Nevertheless, the impact of such imperfect cases on vehicle tracking can be minimized by biasing the algorithm's search space towards correct solutions using tracking and the HD map, as is done in this work.

3.8 Evaluation Metrics

Like in 2D object detection, the *Average Precision (AP)* is also used as the main evaluation metric in 3D object detection. The predictions are first assigned to their corresponding ground truths according to a specific similarity measure to compute AP. The most used similarity measure is the *Intersection over Union (IoU)*, which is the geometric overlap between the ground truth 3D bounding box A and the estimated 3D bounding box B. IoU is defined like this:

$$\text{IoU}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

The *IoU* measure is used to judge a matched prediction as a *True Positive (TP)* or a *False Positive (FP)* by comparing it with a certain threshold. Then, the recall r and precision p can be computed from the ranked detection results according to the following (FN denotes False Negatives):

$$r = \frac{\text{TP}}{\text{TP} + \text{FN}}, p = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

The precision can be regarded as a function of recall, i.e., $p(r)$: As recall approaches zero, the precision will rise, as there are fewer and fewer False Positives in the denominator. Conversely, a rising recall is usually associated with a drop in precision, as the reduced number of False Negatives leads to new False Positives. The interpolated precision values across a range of recall values is defined as AP:

$$\text{AP} = \frac{1}{\mathbb{R}} \sum_{r \in \mathbb{R}} p(r)$$

,

where the set $\mathbb{R}$ refers to a discrete set of values in practice. The mean of this value across all object classes is called *mAP*. The AP metric is commonly used to evaluate object detection

performance, but it only considers the localization of objects and does not account for other factors such as dimension and orientation. To address this limitation, *nuScenes* [Cae+20] proposed a set of True Positive (TP) metrics that measure other prediction errors from the true positives.

The five TP metrics are defined as positive scalars, including the *Average Translation Error (ATE)*, *Average Scale Error (ASE)*, *Average Orientation Error (AOE)*, *Average Velocity Error (AVE)*, and *Average Attribute Error (AAE)*. The *ATE* measures the Euclidean distance between the object center on the 2D ground plane in meters. The *ASE* is the 3D Intersection over Union (IoU) error after aligning orientation and translation. The *AOE* is the smallest yaw angle difference between the predictions and ground truths in radians. The *AVE* is the absolute velocity error as the L2 norm of the velocity differences in 2D in meters per second. Finally, the *AAE* is defined as 1 minus attribute classification accuracy ($1-acc$).

To obtain a single scalar score, *nuScenes* [Cae+20] computes the mean TP metric (mTP) over all object categories $\mathbb{C}$ for each TP metric using the following equation:

$$\text{mTP}_k = \frac{1}{\mathbb{C}} \sum_{c \in \mathbb{C}} \text{TP}_{k,c}$$

where $\text{TP}_{k,c}$ denotes the k-th TP metric (e.g., $k = 1$ means the ATE) for category c. To integrate all the mentioned metrics into a scalar score, *nuScenes* proposes the *nuScenes* Detection Score (NDS) equation that combines the mAP and mTPk scores. The NDS is calculated using the following equation:

$$\text{NDS} = \frac{1}{10} \left[5 * \text{AP} + \sum_{k=1}^{5} (1 - \min(1, \text{mTP}_k)) \right]$$

where AP is the mean average precision defined previously. The term $1 - \min(1, \text{mTP}_k)$ is used to ensure that each TP metric contributes to the score positively, and the division by 10 is used to ensure that the score is in the range $[0, 1]$.

Chapter 4

System Design

4.1 Distributed Architecture

In this chapter, we are going to present technical details of our *Mono3d* solution. We will start by laying out the high-level architecture, as displayed in Figure 4.1.

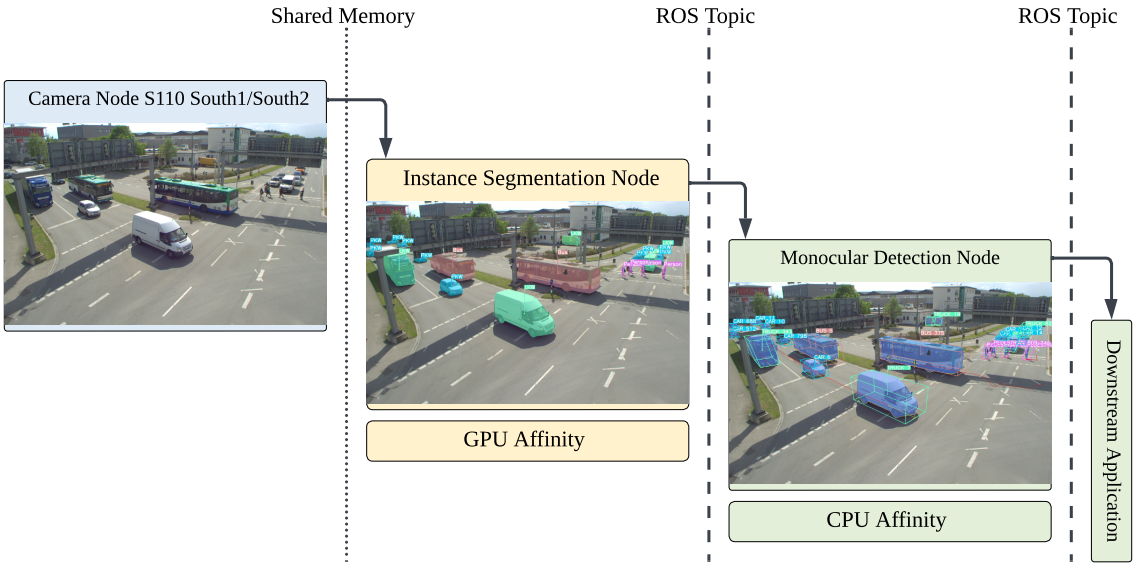


Figure 4.1: High-level sketch of the implemented *Mono3d* solution. The Camera Driver Node publishes camera frames to Shared Memory. They are picked up from there by the $2D$ Instance Segmentation node, which requires a machine with strong GPU resources. The detected instances, including their instance masks, are published for each frame via *ROS*, and received by the $3D$ detector node, which does not require a GPU.

We have divided our *Mono3d* architecture into three independent process nodes. A camera driver node, a $2D$ instance segmentation node, and a $3D$ detection node. The camera driver node publishes full-resolution $1920p$ RGB image frames at $60Hz$, requiring a throughput of (roughly) 360MiB per second. This makes shared memory a good transport medium. For our solution, we make use of the *Eclipse eCAL* shared memory library [1] to facilitate the communication of camera frames. The frames are continuously received by the the $2D$ Instance Segmentation node, which is bottlenecked by the performance of the used GPU. From there, the instance masks of detected objects are published to downstream processes via *ROS*. In this setup, the second stage our two-stage monocular $3D$ detector is just another consumer of the detected $2D$ instances —many other consumers may exist for different purposes. In

[1] https://eclipse-ecal.github.io/ecal/index.html

the second detector stage (the $2D \rightarrow 3D$ lifting stage), $2D$ instance detections are received via ROS and annotated with hypotheses regarding their $3D$ pose. Again, via *ROS*, these $3D$ detections are then broadcast to downstream applications, such as sensor fusion, visualization, or an autonomous vehicle. This separation of the detection pipeline into multiple concurrent, networked processes offers numerous advantages:

1. **Increased robustness**: A distributed multi-process solution enhances system robustness by enabling fault tolerance and redundancy. When a process fails or encounters an error, other processes in the system can continue to function, minimizing the impact of the failure. This capability to recover and adapt to faults makes the overall system more reliable, preventing single points of failure from causing widespread disruptions.

2. **Separation of functional concerns**: In a distributed system, each process can focus on a specific functionality, promoting modularity and maintainability. This separation of concerns simplifies the design and development of the system, making it easier to understand, debug, and extend. It also facilitates reusability, as individual components can be easily replaced or integrated into other systems without affecting the overall system's functionality.

3. **Improved assignment of hardware**: The networked solution allows for better resource allocation and hardware utilization. By spreading tasks across multiple processes and physical machines, the system can allocate specific resources, such as GPUs, to each process based on its requirements. This flexibility enables more efficient use of available hardware, reducing resource contention and potential bottlenecks, while also providing the opportunity to optimize cost and energy consumption.

4. **Increased performance**: The asynchronous process collaboration enables parallelism, allowing multiple tasks to be executed concurrently. This parallel execution can significantly improve the overall performance and throughput of the system. By leveraging the processing capabilities of multiple machines, distributed systems can handle larger workloads and scale more effectively than a single-process system. Additionally, this architecture allows for load balancing, which can help distribute work evenly and prevent overloading individual components.

Natural tradeoffs of this architecture are increased complexity and a nonzero communication overhead, as data must be serialized and deserialized into discrete messages. However, the advantages of the distributed approach tend to outweigh the negatives, especially where scalability is a concern.

4.2 The 2D Object Detector

The $2D$ Object Detector receives camera input frames from a camera driver node via shared memory. It uses an off-the-shelf Object Recognition and Instance Segmentation solution (currently *YOLOv7/Blendmask* [WBL22; Che+20]) to detect $2D$ bounding boxes and pixel masks of Vehicles and Vulnerable Road Users (VRUs). As the performance of such an object detection solution inversely scales to the square of the input image resolution, the $2D$ object detector may also downsize the incoming frames from $1920 * 1200$ to $1280 * 800$ or even $640 * 400$ using *OpenCV* [Bra00]. The performance impact of this downscaling is further explored in Section 5.2. As the instance masks of detected objects must be efficiently passed via *ROS* messages, bit packing [BBB59] is used to efficiently serialize the pixel mask arrays for each

instance. Earlier work within Providentia [Blu22] went so far as to conclude that the throughput demands of the instance masks is far too high for a distributed solution to be viable, but this turned out to be overly pessimistic. Each detected instance consists of a pixel mask, a 2*D* screen-space bounding box, and an assigned object category. As we are currently using the *YOLOv7* detector, which was trained on the *MS COCO* dataset, our 2*D* object detector is able to distinguish between six different object classes: `CAR`, `BUS`, `TRUCK`, `MOTORCYCLE`, `BICYCLE`, `PEDESTRIAN`. The `VAN`, `EMERGENCY_VEHICLE`, and `OTHER` classes from the *A9 Dataset* cannot yet be recognized.

4.3 The 3D Object Detector

In the second stage of our *Mono3d* solution, 2*D* object detections are received as an array D_t^{2D} of detection triplets for each frame:

$$D_t^{2D} = \{(\texttt{category}_n, \overrightarrow{\texttt{bbox}}_n^{2D}, \texttt{mask}_n)_{n=0}^{n<N}\}$$

The per-frame detection array is received in a single *ROS* topic update message from the upstream 2*D* object recognition node. Now, the task of the receiving 3*D* object detector is to generate a best-effort physical pose hypothesis for each 2*D* detection. The estimated 3*D* pose must include the objects bottom-center $\overrightarrow{\texttt{xyz}}$ position, its `length`, `width`, and `height` in meters, and its `yaw` heading angle. The full process of estimating these pose parameters from a 2*D* detection triplet is illustrated in Figure 4.2.

The high-level process which turns the instance mask and category inputs into a 3D pose hypothesis for each 2*D* detection is summarized in the following: First, the instance mask is filtered to extract its bottom contour (see Section 4.4). This yields a line of 2*D* pixel coordinates $\{\overrightarrow{uv}_i\}_{i=0}^{i<N}$. These are projected back into 3*D* map space using a simple ray-cast, whereas the z (altitude) coordinate is assumed to be 0. The resulting 3*D* point cloud $\{\overrightarrow{\text{xyz}}_i\}_{i=0}^{i<N}$ is optionally filtered using the *DBSCAN* algorithm to remove outlier points. If the category of the detected object is that of a *Vulnerable Road User* such as a pedestrian or bicycle, the bottom contour will be converted to a 3*D* pose estimation with a special algorithm as explained in Section 4.5. Otherwise, the *HD Map Lookup Grids* (see Section 4.7) are queried at the positions of the 3*D* points. This query provides a set of $\{(\texttt{lane\_id}_k, \theta_k)\}_{k=0}^{k<K}$ for each point where the bottom contour touches a particular lane's surface. These tuples are averaged and aggregated into a histogram per unique `lane_id` (see Section 4.8). This histogram provides θ_k values with associated confidences $\text{Score}_{\text{map}}(\theta_k)$ derived from the number of bottom contour points which touch the associated lane.

Thereafter, for each yaw option θ_k, the *L-Shape-Fitting (LSF)* algorithm is used to calculate the physical length and width of the given bottom contour assuming the given θ_k. *LSF* also returns an error score $\text{Score}_{\text{lsf}}(\theta_k)$ indicating $p(\theta_k|\{\overrightarrow{\text{xyz}}_{0\leq i<N}\})$ (see Section 4.6). The proposed length and width values are also sanity-checked against limits per the detected object category. Using these assumed spatial extents, the spatial position and height can be regressed from the 2*D* instance bounding box (see Section 4.10).

Finally, the 2*D* instance bounding box is also matched against historical detections via *SORT* object tracking to find previous detections of the same vehicle. According to the historical vehicle trajectory, a θ_k proposal also receives a historical plausibility/fitness score $\text{Score}_{\text{track}}(\theta_k)$ (see Section 4.9). Multiplied together, the three scores $\text{Score}_{\text{map}}(\theta_k)$, $\text{Score}_{\text{lsf}}(\theta_k)$ and $\text{Score}_{\text{map}}(\theta_k)$ yield the final $\text{Score}(\theta_k)$. Now, for the object in question, we hypothesize that the best θ_k yaw choice (and associated pose parameters) is the one which finally receives the highest score. This is calculated for every detected 2*D* object to perform the $2D \rightarrow 3D$ lifting operation.

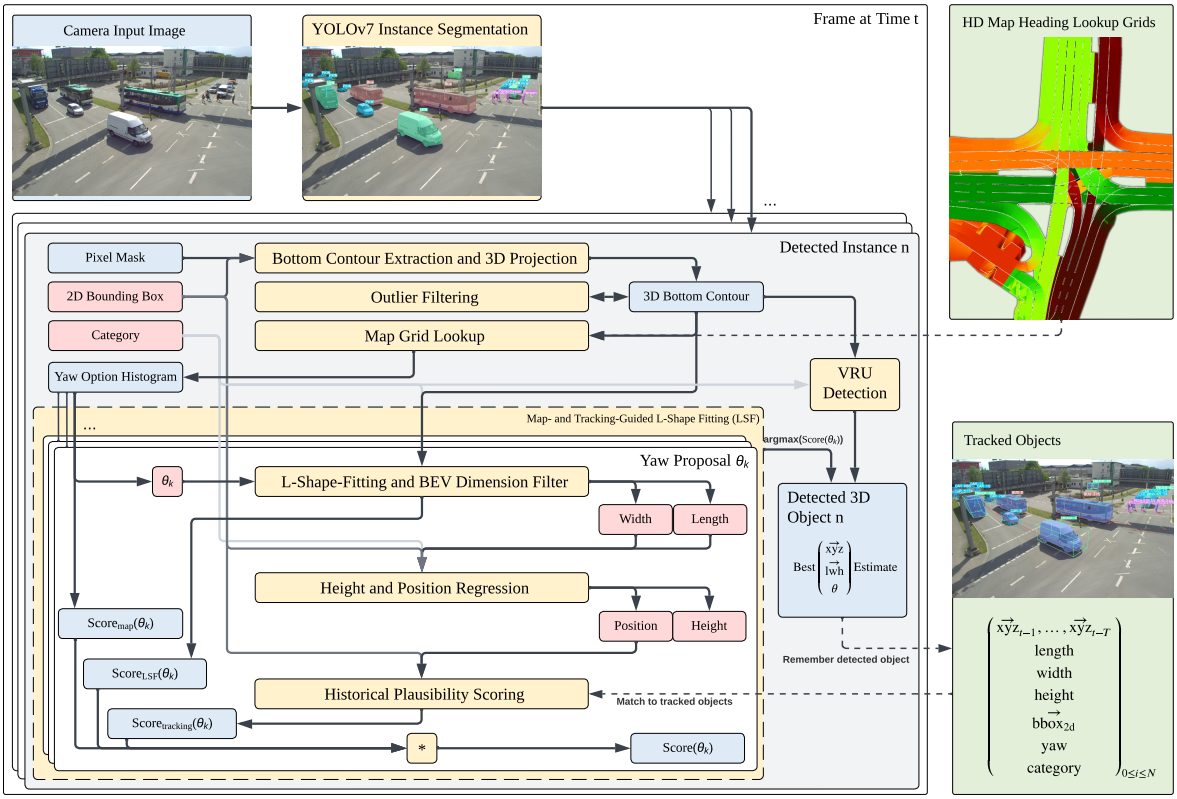


Figure 4.2: Low-level illustration of our *Mono3d* pipeline. To determine a 3*D* pose for a non-VRU 2*D* object instance, its associated pixel mask and category are passed through six processing steps: **(1)** *Bottom Contour Projection*, **(2)** *Outlier Filtering*, **(3)** *Map Grid Yaw Option Lookup*, **(4)** *L-Shape-Fitting*, **(5)** *Height and Position Regression*, **(6)** *Historical Plausibility Scoring*. **Legend:** Blue boxes mark normal I/O data, red boxes mark I/O variables which are primary components of a final pose hypothesis. Yellow boxes mark processes. Green boxes with dashed arrows mark auxiliary data flow which is not restricted to the scope of the current frame.

The following sections of this chapter serve to explain each step in detail.

4.4 Bottom Contour Extraction and Filtering

In the first processing step for each 2*D* detection, the detection's pixel mask is converted to a 3*D* bottom contour. We can express the image mask for detection n as a function $\texttt{Mask} : \mathbb{N}^2 \rightarrow \mathbb{B}$, as it maps uv pixel coordinates to a boolean value which indicates whether the object occupies the given pixel. The pixel coordinates are modeled as $\overrightarrow{uv}$ vectors with u indicating the pixel column, v indicating the pixel row, and $\overrightarrow{00}$ as the top-left image corner. In this model, the 2*D* bottom contour points B^n_{2D} are extracted as follows:

$$B^n_{2D} = \{\overrightarrow{uv} | \texttt{Mask}_n(\overrightarrow{uv}) \wedge \neg \texttt{Mask}_n(\begin{pmatrix} u \\ v+1 \end{pmatrix})\}$$

In practice, the operation is realized using a *numpy* [Har+20] *argmax* call. Successively, the 2*D* bottom contour points are projected into 3*D* map space via a raycast operation. We assume that the ground plane is located at $z = 0$, P is the camera's intrinsic (projection) matrix, R is the camera's spatial rotation and $T = \begin{pmatrix} T_x & T_y & T_z \end{pmatrix}$ is the camera's spatial translation relative to the ground plane. Therefore, the bottom contour 3*D* ground points $B^n_{3D,\text{ground}}$ are calculated as follows:

$$B^n_{3D,\text{init}} = \bigcup_{\overrightarrow{uv} \in B^n_{2D}} \{\overrightarrow{\text{xyz}} | \overrightarrow{\text{xyz}} = R^T * P^{-1} * \overrightarrow{uv1}\}$$

$$B^n_{3D,\text{ground}} = \bigcup_{\overrightarrow{\text{xyz}} \in B^n_{3D,\text{init}}} \{\overrightarrow{\text{xyz}_{\text{ground}}} | \overrightarrow{\text{xyz}_{\text{ground}}} = T + \overrightarrow{\text{xyz}} * -T_z/z\}$$

This point set is then filtered using the *DBSCAN* [Sch+17] algorithm to get rid of outlier points which do not belong to the true $3D$ bottom outline.

4.5 Detection of Vulnerable Road Users

Vulnerable Road Users (VRUs), such as pedestrians or bicyclists, receive special treatment in our *Mono3d* pipeline. This is, because several assumptions which are made for optimized road vehicle detection are not valid for VRUs:

1. **Legal Heading Assumption**: For road vehicles, the monocular detector makes use of the HD map to derive yaw options from legal traffic flow directions. This is not possible for bicyclists or pedestrians, as these often move along unmapped territory beyond the road, and perpendicular to normal traffic directions on mapped vehicle lanes.
2. **Box Shape Assumption**: Cars, trucks and other motorized road users usually exhibit an instance mask bottom contour shape which indicates the occupied ground area of the object, and therefore allows for L-Shape-Fitting as an effective algorithm to derive the object's length and width. This is usually not the case for pedestrians and bicycles, which might have major errors in their back-projected bottom contours (see Figure 4.3).

For these reasons, we have implemented a simplified detection flow for VRUs: We do not attempt to detect their orientation, instead they are always assigned $\theta = 0$. Furthermore, their dimensions are set to fixed values. The only variables which remain to be estimated are the VRU's BEV coordinates. This is done by averaging the positions of 10 percent of the road user's bottom contour $B^n_{3D,\text{ground}}$ points, which are closest to the camera from the BEV perspective.

The reasoning for this strategy is illustrated in Figure 4.3: The camera C detects pedestrian A and cyclist B. The projection of their instance mask bottom contour (highlighted in red) to the ground is highlighted in blue. Points on the $3D$ bottom contour which are close to the camera BEV position and therefore considered to estimate the position of A and B are respectively shown.

4.6 Bottom Contour L-Shape Fitting

The process of estimating θ, width and length for motorized road vehicles is realized using the *L-Shape-Fitting (LSF)* algorithm [Zha+17] with an augmented score calculation function. From the standard LSF implementation, we use the *Variance* criterion, as it was demonstrated to perform the best in the original paper. The value of the LSF variance score $\texttt{Variance}_{\text{LSF}}(\theta_k)$ is in the range of $[-\infty, 0]$. We normalize this value to the range of $[0, 1]$ as $\text{Score}_{\text{lsf}}(\theta_k) = 1/(1 - \texttt{Variance}_{\text{LSF}}(\theta_k))$. Now, we combine it using multiplication with the map confidence

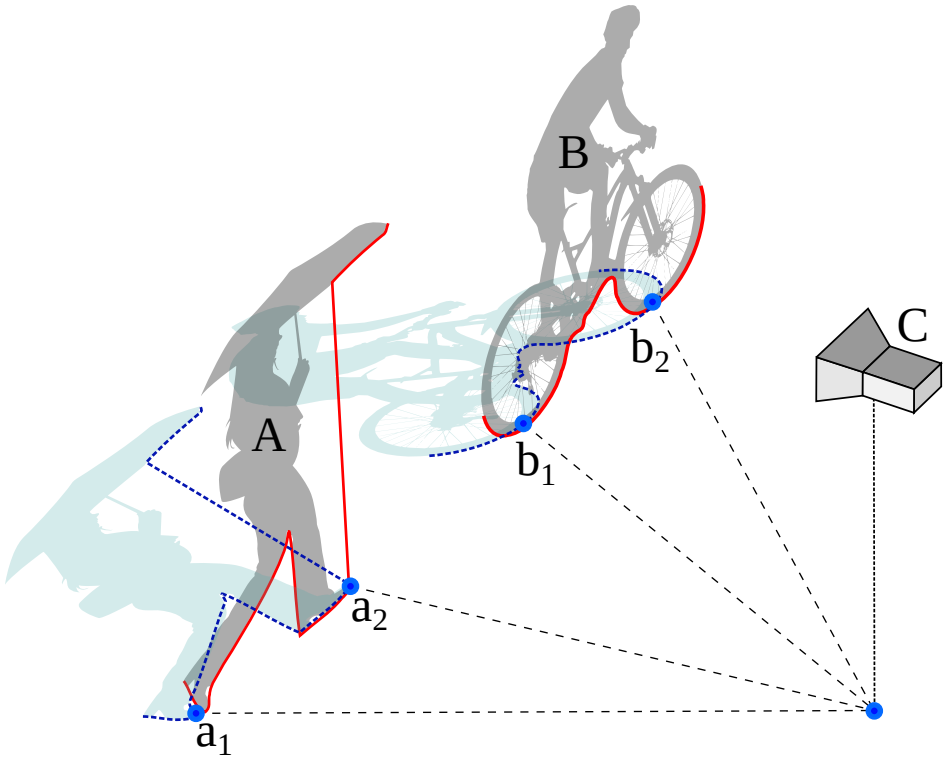


Figure 4.3: Illustration of our proposed solution for VRU detection. Camera C observes pedestrian A and cylist B. Their 2D bottom contour outline is highlighted in red, the projection of the outline to the ground is highlighted in dashed blue lines. Only the highlighted points on the respective bottom contour projections which remain close to the camera may be considered to estimate the position of A/B.

score (see Section 4.8) and the historical plausibility score (see Section 4.9), to arrive at the final Score(θ_k). Unlike in the original *LSF* algorithm, we constrain the choice of θ_k to values which are derived from the HD map.

4.7 HD Map Lookup Grids

In this work, we propose to select the θ heading value for each $3D$ vehicle detection from a traffic flow direction that is derived from the HD map. However, so far it has been left up to the reader to imagine how the HD map is exactly queried to produce heading options. The following two sections explain this step in more detail. First and foremost, we propose a spatial grid lookup structure for heading options at fixed planar increments, covering the *field-of-view (FOV)* of each sensor. This approach allows us to pre-compute the heading options for a discrete subset of representative map locations, and the subsequent real-time lookup becomes computationally trivial.

4.7.1 Lane Tesselation

The precise mechanic of rendering the previously outlined heading lookup buffers begins by converting the *OpenDRIVE* [DSG10] map lane surfaces into triangles. This process is called *tesselation*. An *OpenDRIVE Road* is defined as a sequence of *Lane Sections* along a reference (center-)line. A *Lane Section* is a bundle of adjacent lanes. In the *OpenDRIVE* format, a *Lane* is enclosed between an inner and an outer boundary line. A *Lane Section* is therefore a sequence

of $M+1$ lane boundary lines which enclose M parallel lanes. We make use of the *OpenDRIVE Converter Tool* [2] to sample the *OpenDRIVE* lane boundaries from *Bezier Curves* into polylines. The tool discretizes the lane boundaries as an array of $\overrightarrow{\text{xyz}}$ coordinates at an interval of one meter.

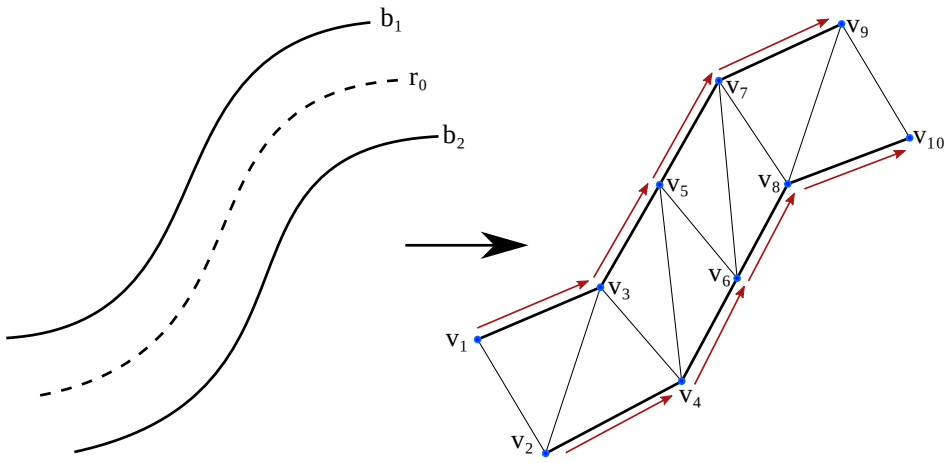


Figure 4.4: Illustration of our lane tesselation algorithm for a single lane section. The OpenDRIVE lane boundary lines are sampled at constant intervals to convert them to polylines. Orientations are calculated for each derived poly-line vertex. For each lane, the enclosing polylines are then "zipped" together with a triangle-strip.

As illustrated in Figure 4.4, the result of the lane section point sampling is a matrix $V \in \mathbb{R}^{(M+1)\times N\times 4}$, with N being the number of points sampled along each boundary line. Each vertex $v_{(m,n)} \in V$ has four scalar components: Three spatial dimensions for its location, and one extra dimension θ to indicate the heading at its location. In Figure 4.4, the heading at the vertex position is indicated with a red arrow. It is calculated as $\theta(v_{(m,n)}) = \texttt{arctan2}(v_{(m,n+1)} - v_{(m,n)})$. The last vertex assumes the theta value of the second-to-last. Each lane $l_m \in \{l_1, \ldots, l_M\}$ is then tessellated as a triangle strip via a sliding window of three vertices over the sequence $\{v_{(m,1)}, v_{(m+1,1)}, \ldots, v_{(m,N)}, v_{(m+1,N)}\} \subseteq V$. This results in $2(N-1)$ triangles covering the surface of each lane.

4.7.2 Lane Rasterization

As lanes are converted into batches of triangles, it is possible to use fast rasterization algorithms to "paint" them into grid buffers. Rasterization is the process of converting a vector representation of a shape into a pixel or voxel representation. Both representations have distinct advantages. A vector representation is usually the most efficient for storage. But it is comparatively slow to determine whether it contains a specific point, and how that point is situated in relation to the vertices of the geometry. This is where a pixel or grid-based representation has a lot of potential. The grid can store which locations are covered by the geometry and can also serve as a cache for vertex attributes of the geometry at each grid cell location. In case of the lane surfaces, each grid cell stores the heading value of the lane at the grid cell location. To account for overlapping lanes, we instantiate one grid per *OpenDRIVE* road. Only lane sections which belong to the same road are rendered into the same grid.

The spatial coverage of each heading lookup grid for a camera is determined by iterating through the available lane sections, and selecting those which are both in view of the camera and not further than 200 meters away. Each cell in each grid then covers a $10 * 10$ cm square. This yields a grid resolution of $898 * 1140$ cells for the *South-1* camera, and $1436 * 1224$

[2] https://github.com/Brucknem-TUMProjects/OpenDRIVE

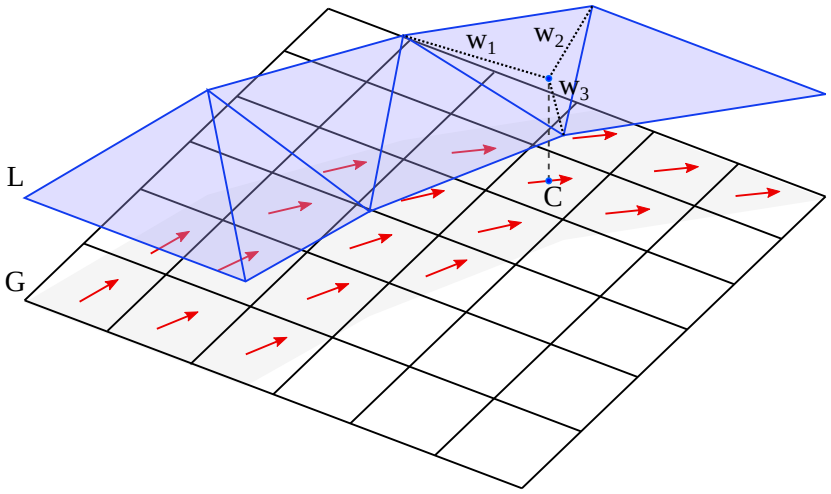


Figure 4.5: Illustration of lane heading rasterization using barycentric coordinates. Lane L is rasterized into grid G, and the barycentric coordinates w_1, w_2 and w_3 are shown for one specific gric-cell center point C.

cells for the *South-2* camera. For the highway, we extend the grid range to 1000 meters and increase the grid cell size to $50 * 50$ cm.

In detail, the process of rasterization means to loop over every "pixel" (grid cell) and determine if it lies inside one of the triangles of the tessellated lane sections for the road of the tested grid. If it does, we use the barycentric coordinates of the grid cell with respect to the triangle to interpolate the corresponding heading value for the point inside the triangle. This process is also illustrated in Figure 4.5.

The algorithm involves the following steps:

1. Given a triangle with vertices v_1, v_2, and v_3, and a grid with width w and height h, loop over every cell which is within the bounding box of the triangle.

2. For each candidate cell $C = (x, y)$, compute the barycentric coordinates (w_1, w_2, w_3) of the cell with respect to the triangle using the following formulas:

$$
\begin{aligned}
\boldsymbol{v_1 v_2} &= \boldsymbol{v_2} - \boldsymbol{v_1} \\
\boldsymbol{v_1 v_3} &= \boldsymbol{v_3} - \boldsymbol{v_1} \\
\boldsymbol{v_1 c} &= C - \boldsymbol{v_1} \\
w_1 &= \frac{\boldsymbol{v_1 v_3} \times \boldsymbol{v_1 c}}{\boldsymbol{v_1 v_3} \times \boldsymbol{v_1 v_2}} \\
w_2 &= \frac{\boldsymbol{v_1 c} \times \boldsymbol{v_1 v_2}}{\boldsymbol{v_1 v_3} \times \boldsymbol{v_1 v_2}} \\
w_3 &= 1 - w_1 - w_2
\end{aligned}
$$

   Note that the order of the vertices in the cross product determines the winding order of the triangle. If the winding order of the triangle is reversed, the signs of the cross products will also be reversed.

3. If all three barycentric coordinates are non-negative, the pixel lies inside the triangle. We can then use the barycentric coordinates to interpolate the corresponding heading value θ_C of the grid cell C inside the triangle using the following formula:

$$\theta_C = w_1 \theta(\boldsymbol{v_1}) + w_2 \theta v_2 + w_3 \theta(\boldsymbol{v_3})$$

where $\theta(\boldsymbol{v})$ is the heading for a boundary vertex as described in Section 4.7.1.

4. Repeat steps 1–3 for every grid cell of every road grid.

In practice, we have implemented this algorithm in *numpy*, which allows the parallel execution of all grid cell tests for one triangle. A triangle covers 99.85 cells on average. This makes the process of rendering the heading lookup grids quite fast, with an average wall-time of 1.48s to render 6812 triangles on a Ryzen 5600 processor. This process can be comfortably executed during the startup of the 3D detector process. At an average of 25 *OpenDRIVE* roads per camera view on the S110 intersection, and five bytes stored per cell, the final grid-set for one camera occupies 165.78MiB of RAM on average. The resulting heading fields are visualized in Figure 4.6.

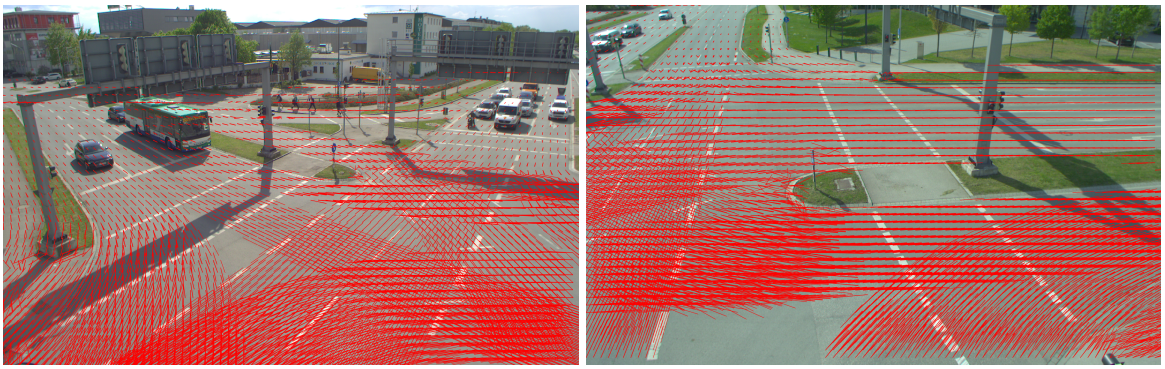


Figure 4.6: Derived heading values rendered on top of the S110-S2 camera perspective (left) and the S110-S1 camera perspective (right).

4.8 HD-Map-Augmented L-Shape Fitting

The previously outlined heading lookup grids, which are rendered from the HD map, can improve the basic LSF algorithm in two ways:

1. The grid may be used to calculate possible heading values which are to be evaluated by LSF, rather than using the basic approach of iterating over $[0, \pi]$ at fixed increments.

2. The grid can also yield a confidence value $\text{Score}_{\text{map}}(\theta_k)$ for each heading option θ_k, given how many bottom contour points support the respective option.

In general, the heading lookup grids $G_1, \ldots, G_{|R|}$ for each road $r \in R$ are queried using the bottom contour B^n_{3D} of vehicle detection n, which is a series of 3D points $\overrightarrow{\text{xyz}}^n_1, \ldots, \overrightarrow{\text{xyz}}^n_k$. We define the grid lookup operator $G_r(\overrightarrow{xy})$ as a function $\mathbb{R}^2 \to \mathbb{B} \times \mathbb{R}$ which delivers a boolean flag and a heading value given an arbitrary planar bottom contour position. The boolean flag indicates whether the road r covers the position $\overrightarrow{xy}$ at all. The two return values of the operator are referenced as $G_r(\overrightarrow{xy})_{\mathbb{B}}$ and $G_r(\overrightarrow{xy})_{\theta}$, respectively.

Now, the confidence of the map that a particular road grid G_r provides the correct heading for detection n may be expressed through the absolute count of bottom contour points $\overrightarrow{\text{xy}}^n_i \in B^n_{3D}$ for which $G_r(\overrightarrow{xy}^n_i)_{\mathbb{B}}$ is true. We call this count the *lookup histogram value* $h(G_r, B^n_{3D})$ for the road grid G_r and the bottom contour B^n_{3D}. The heading option θ_r provided by G_r will then be the $\texttt{average}(\{G_r(\overrightarrow{xy}^n_i)_\theta | G_r(\overrightarrow{xy}^n_i)_{\mathbb{B}}\}^{i \le k}_{i=1})$. Finally, the lookup histogram value must be

converted to a range of $[0,1]$ to become a score which can be used as a factor. To this end, we normalize $h(G_r, B^n_{3D})$ as $\text{Score}_{\text{map}}(\theta_r, B^n_{3D})$:

$$\text{Score}_{\text{map}}(\theta_r, B^n_{3D}) = \frac{h(G_r, B^n_{3D})}{\max_{s \in R} h(G_s, B^n_{3D})}$$

This process is illustrated in Figure 4.7. There are three grids G_1, G_2 G_3 for three overlapping roads. The bottom contour of vehicle V as observed by camera C is shown with 11 points. Now, the lookup histogram values for each grid are shown in the bar-chart on the right side of the figure. For grid G_1, all 11 points match to a nonempty cell. For grid G_2, six points match. For grid G_3, only five points match. These absolute histogram values are then converted to relative confidence score values, by dividing each over the maximum histogram value, which is 11.

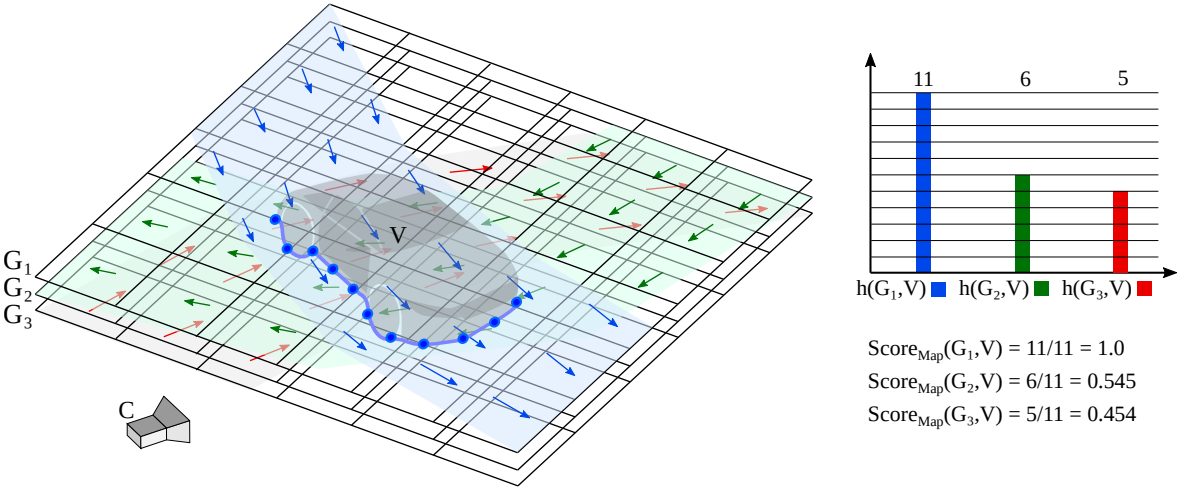


Figure 4.7: Illustration of a map heading histogram lookup for the bottom contour of vehicle V. The map lookup grid-set provides grids for three roads G_1, G_2 and G_3. The histogram values $h(G,V)$ per road, as well as the respective normalized scores which are calculated from the histogram, are shown on the right.

4.9 Tracking-Augmented L-Shape Fitting

In the example from Figure 4.7, the vehicle is conveniently positioned such that the correct heading option clearly wins the highest confidence score. However, it is easily possible to imagine V in a position slightly further upwards, where the map confidence scores would not be so unambiguous. For such cases, the raw LSF variance criterion score may help. However, we hypothesize that the vehicle's positional history, as described all the way back in the Introduction Section 1.6.1 may provide an additional historical plausibility score $\text{Score}_{\text{track}}(\theta_k, V)$ for vehicle V and heading candidate θ_k.

This is calculated as follows: Using a screen-space SORT tracker [Bew+16], we match a detected object's bounding box to detections from previous frames. For a successfully matched detection, historical 3D position values $L = \{\boldsymbol{l}_{t-1}, \ldots, \boldsymbol{l}_{t-T}\}$ are retrieved. Given the historical positions L and a position hypothesis $\boldsymbol{l}_t(\theta_t)$, the historical plausibility score $\text{Score}_{\text{track}}(\theta_k, V)$ for a yaw hypothesis θ_k is calculated as in the following equation:

$$\text{Score}_{\text{track}}(\theta_k, V) = \prod_{\delta_t=1}^{\delta_t \leq T} \pi/2 - \Delta_{\angle}(|\theta_t - \text{atan2}(\boldsymbol{l}_t(\theta_t) - \boldsymbol{l}_{t-\delta_t})| \bmod \pi)$$

The Delta-Angle function $\Delta_{\angle} : [0, \pi) \rightarrow [0, \pi/2)$ converts the passed raw angular difference, which is already less than π, into a value less than $\pi/2$ by returning angular deltas $\delta_{>\pi/2}$ larger than $\pi/2$ as $\pi - \delta_{>\pi/2}$. This ensures that a yaw hypothesis, which is parallel, yet opposed to a historical orientation, is not erroneously punished. In practice, we have implemented a threshold of six historical positions that are evaluated to determine the plausibility of a yaw hypothesis.

4.10 Joint Height and Position Estimation

As an estimate for a detected vehicle's width and length is made using our augmented L-Shape-Fitting algorithm, two critical values remain to be established for a fully defined 3D bounding box: The vehicle position and the vehicle height. Also, our tests showed that the LSF-based BEV dimension hypothesis must often be corrected due to errors in the bottom contour (see Sections 5.7 and 5.7). Because of this, we need to clamp the BEV width/length dimensions to category-based minimum and maximum values. In this case, the vehicle's bounding box needs to grow or shrink in some direction, and the position hypothesis of the LSF algorithm becomes uncertain.

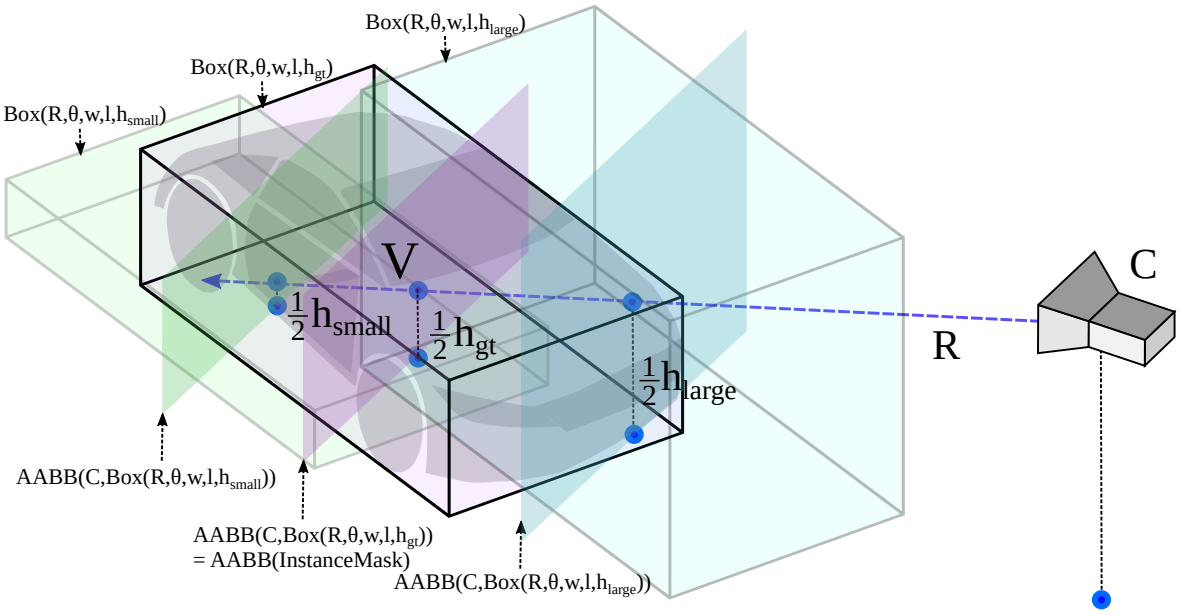


Figure 4.8: Illustration of a joint 3D height and location for vehicle V. Initially, camera C observes only the screen-space bounding box $\texttt{AABB}(\texttt{InstanceMask}_V)$. We now search for the vehicle height h based on a value along the ray R, which is cast from C through the 2D AABB center. Only for the correct h_{gt} will the 2D AABB of the predicted 3D box $\texttt{AABB}(C, \texttt{Box}(R, w, l, \theta, h_{\text{gt}}))$ be equal to $\texttt{AABB}(\texttt{InstanceMask}_V)$.

There is however a strong criterion towards the position of the 3D bounding box, which we can use to estimate it: It is desirable to position a vehicle's 3D bounding box such that it covers the 2D instance mask of the same vehicle when projected to screen-space. This also means, that the screen-space-projected center of the 3D bounding box should coincide with the center of the 2D bounding box. So we know, that the center of the 3D bounding box must be somewhere along a 3D ray which is cast from the camera through the center of the instance mask. Because a camera mounted from the infrastructure perspective also generally looks down, the position along the ray also dictates the height (more specifically half of the height) of the vehicle. Now, as the bounding box height changes, so does the height of its

2D screen-space projection. Therefore, we can simultaneously optimize both the height of the 3D box and its position, with the goal to match the screen-space bounding box of the estimated 3D bounding box to the instance mask bounding box.

Algorithm 1 RegressHeightAndLocation

Require: `width, length, theta, bbox_2d, cam, min_height, mean_height, max_height`
Ensure: `height, location`

```
x0, y0, x1, y1 ← bbox_2d
detection_img_height ← y1 − y0
height ← mean_height                    ▷ Initially estimate height by mean for category
jump_size ← max_height − mean_height
mask_center ← [x0 + (x1 − x0)//2  y0 + (y1 − y0)//2]
location ← [0 0 0]
for i ← 0 to 9 do                       ▷ Binary search for correct height/position, 10 iterations max.
    location ← cam.project_to_ground(mask_center, height × 0.5)
    location[2] ← 0
    box ← get_box(theta, width, length, location)
    box ← box ⊕ (box + [0  0  height])
    all_corners_projected ← cam.project_to_image(box^⊤)
    min_img_y ← min(all_corners_projected[1])
    max_img_y ← max(all_corners_projected[1])
    estimated_img_height ← max_img_y − min_img_y
    img_height_delta ← detection_img_height − estimated_img_height
    if |img_height_delta| < 1.0 then    ▷ Less than one px difference. We are done.
        break
    end if
    if i > 0 and sign(img_height_delta) ≠ sign(jump_size) then
        jump_size ← jump_size × 0.5     ▷ Halve the step size on direction change.
    end if
    jump_size ← sign(img_height_delta) × |jump_size|
    prev_height ← height
    height ← height + jump_size
    height ← max(min(height, max_height), min_height)   ▷ Respect limits.
    if height = prev_height then        ▷ We've hit a limit, no need to continue iterating.
        break
    end if
end for
return height, location
```

In practice, we have implemented this joint height and position regression using binary search over the 3D height value, as described in Algorithm 1. As the optimization goal, it is sufficient to match the image space box height. We limit the number of optimization steps to 10, but observe that the algorithm converges on average after five steps. We also add a sanity check to abort iteration, if the optimization cannot progress because the estimated height is growing below or beyond sanity limits. This can occur when the pixel mask for the detected instance is highly erroneous or obscured. The approach is also illustrated in Figure 4.8. Initially, camera C observes only the screen-space bounding box $\texttt{AABB}(\texttt{InstanceMask}_V)$. We now search for the vehicle height h based on a value along the ray R, which is cast from C through the 2D AABB center. Only for the correct h_{gt} will the 2D AABB of the predicted 3D box $\texttt{AABB}(C, \texttt{Box}(R, w, l, \theta, h_{\text{gt}}))$ be equal to $\texttt{AABB}(\texttt{InstanceMask}_V)$.

4.11 Late HD Map Lookup

In the rest of this chapter, two optional design elements are explained which do not necessarily improve our system, but answer some questions in our evaluation study. Feel free to skip ahead to the next chapter if you are only interested in the best version of the system.

First, as an alternative to the rather complex "early" HD map lookup for all bottom contour points (with a derived confidence histogram which feeds into L-Shape-Fitting), we initially also implemented much a simpler "late lookup" approach.

Here, we only query the HD map for heading options at the current vehicle position. Then we pick the heading option which is most in line with the vehicle's historic positions, or merely the option closest to the LSF-based proposal, if tracking is turned off.

The drawbacks of this approach are manifold: L-Shape-Fitting has already settled on a heading value prior to the map lookup, so the width and length will not be corrected. Furthermore, the vehicle position at this point is already very disconnected from the ground-truth sensor input. So, a lot of positional information is discarded before the HD map lookup is made. Finally, the joint position-height regression algorithm relies on established values for width, length and θ, so correcting θ afterwards is logically not optimal.

4.12 3D SORT Tracking

Another "questionable idea" (in hindsight) which we tested was to use *SORT* in 3D bird's-eye-view (BEV). In conjunction with 2D screen-space tracking, which is required to calculate the historical plausibility score for L-Shape-Fitting, we have also applied the *SORT* algorithm to track detections in 3D BEV space. This is supposed to stabilize 3D position estimates through the *Kalman Filters* which are used by *SORT*. Because *SORT* does not support a heading value in the bounding box state, we feed BEV detections as squares of size $((l + w)/2)^2$ into the algorithm. This is immediately obvious as a drawback of using *SORT* for 3D tracking, and we do not expect this application of *SORT* to improve our results. Instead, a more capable Kalman Filter detection is needed which also takes vehicle heading into account.

Chapter 5

Evaluation

5.1 Qualitative Results

To acquire a first impression of the strengths and weaknesses of our fully implemented HD map-assisted monocular 3D object detector, some sample detection frames are presented here which showcase particular situations.

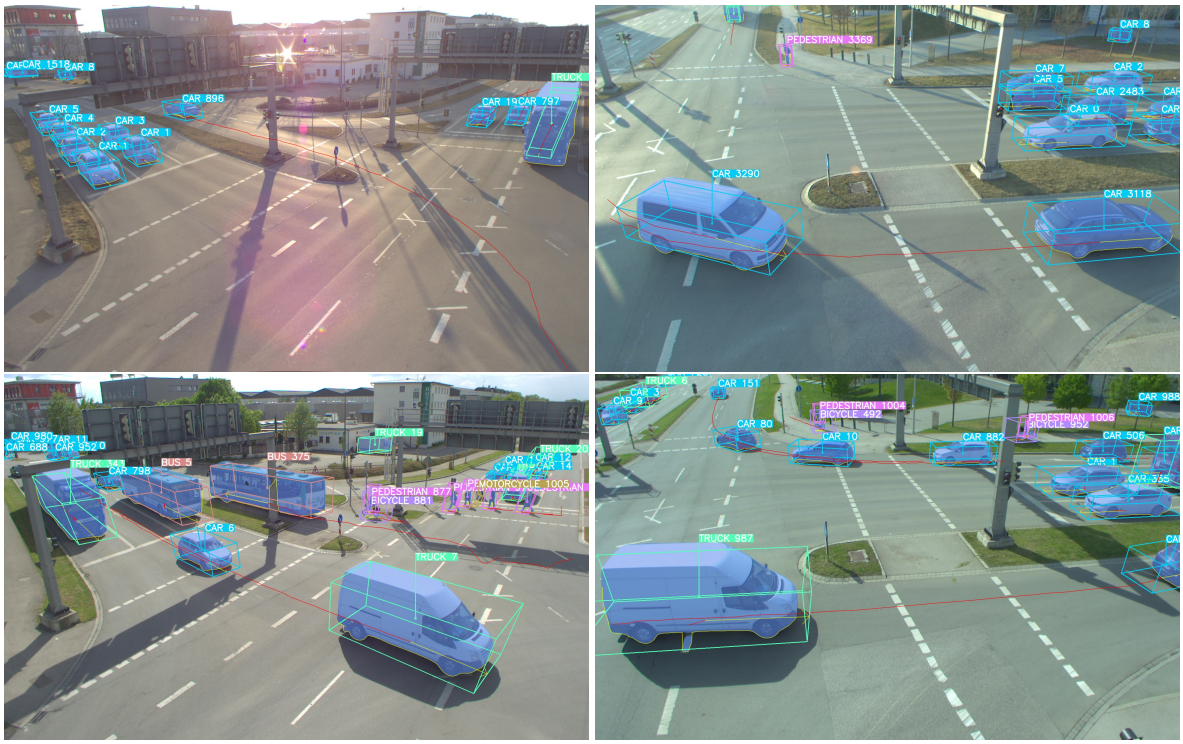


Figure 5.1: Hand-picked frames to showcase the qualitative performance of our implemented monocular 3D object detector during daytime.

First off, Figure 5.1 shows representative frames with predicted vehicle bounding boxes, where the implemented system performs quite well. The frames in the left column are from the South-2 camera, which has a greater detection range, as it is angled more towards the horizon. The two frames for this camera show that the implemented system can correctly detect the 3D shapes of many overlapping vehicles, which are oriented opposed or perpendicular to each other. The top-left frame showcases the good performance of the screen-space *SORT* tracker, which tracks `CAR 896` throughout its whole left turn. The bottom-left image shows the usefulness of the *DBSCAN*-based bottom contour filter in the case of `BUS 375`. The

detection of this bus is split into two parts by the pole of the overhead gantry bridge. This introduces a lot of noise into this detection's bottom contour, as highlighted in yellow. However, the detected 3D bounding box is unaffected by this noise. This frame also showcases the performance of our Vulnerable Road User (VRU) detection. Both present pedestrians and bicycles in the image are correctly localized.

The frames in the right column are from the South-1 camera, which has a shorter detection range, as it is angled more towards the ground. Both images in the right column again show generally good vehicle tracking and orientation estimation capabilities.

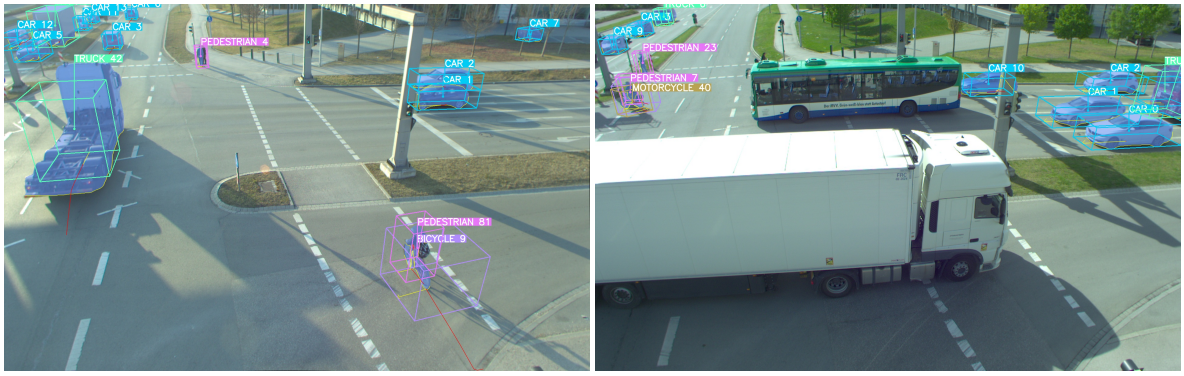


Figure 5.2: Selected frames to showcase problems of our implemented monocular 3D object detector.

Figure 5.2 highlights particular weaknesses of the implemented monocular detector. The image on the left highlights three problems. First, the length estimation of `TRUCK 42` is bad, most likely due to over-eager bottom-contour filtering[1]. Second, only one of the two pedestrians is detected —`PEDESTRIAN 4` is standing next to another one. Third, the system detects the rider of a bicycle as a separate entity, as in the case of `BICYCLE 9` and `PEDESTRIAN 81`. This points to a need for more fine-tuned non-maximum suppression (NMS) of 3D detections. The image on the right shows a single problem: Our detector (more specifically the used *Yolov7* instance segmentation model) frequently has problems with detecting the instance masks of large objects near the camera. In this case, both the big white truck in front of the camera, and the bus, which is a bit further away, are not detected.

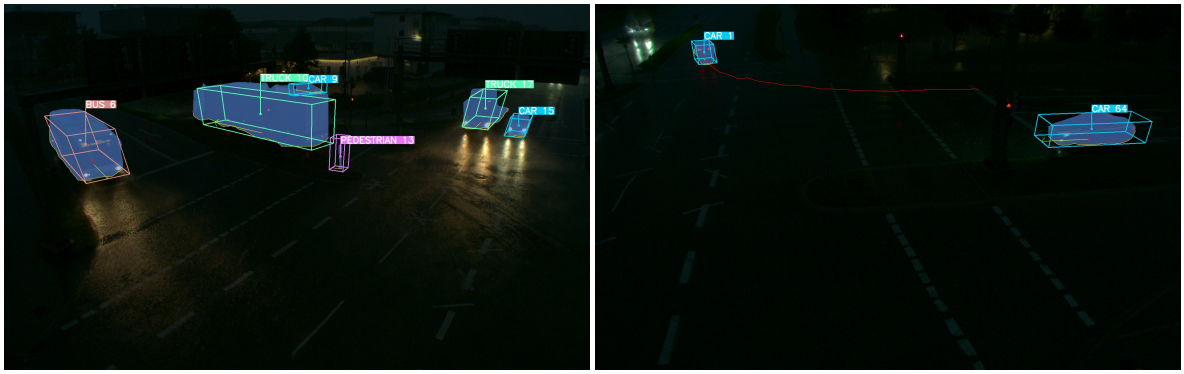


Figure 5.3: A selection of frames to showcase the qualitative performance of our implemented monocular 3D object detector at night.

Finally, Figure 5.3 displays how the system performs at night. Naturally, the detection performance of a system which is based on an RGB camera which operates in the human visible light spectrum will be worse at night, due to the reduced visibility. However, the

[1]This could be fixed by tweaking the ϵ (maximum point distance) and `min_samples` (minimum cluster size) hyper-parameters for the *DBSCAN* algorithm. We have set $\epsilon = 0.5m$ and `min_samples` $= 5$

used *Yolov7* instance segmentation model still works to some degree at night. The predicted instance masks are generally noisier, and there are more false positives (such as PEDESTRIAN 13) and false negatives (such as the truck on the top-left in the right image).

5.2 Runtime Performance

The runtime performance of the monocular detection architecture is limited by the computational capabilities of the GPU. In our evaluation, we used an *RTX-3090* GPU to assess the frame rates achieved by various models. Table 5.1 presents the frame-rates of the different models:

Table 5.1: Model performance on *RTX-3090* GPU

Model	Frame Rate (FPS)
Yolact Edge (550^2 *px)*	60
Yolov7 (640^2 *px)*	52
Yolov7 (1280^2 *px)*	22
Yolov7 (1920^2 *px)*	12

As shown in Table 5.1, both *Yolact Edge* and *Yolov7-640* models achieve frame-rates between 55 and 60 FPS on the *RTX-3090* GPU, indicating that they are suitable for real-time applications. However, as the input resolution increases, the runtime performance of the models decreases. For instance, *Yolov7-1280* achieves 22 FPS, and *Yolov7-1920* reaches only 12 FPS. This analysis highlights the importance of considering the GPU constraints when designing and implementing two-stage monocular 3D object detection models, particularly for real-time applications.

5.3 Quantitative Evaluation Strategy

For an objective understanding of the performance of our *Mono3D* solution, we performed a thorough ablative quantitative evaluation. For this purpose, we use the *Providentia Intersection Scenario* dataset, as previously mentioned in the Introduction (see Chapter 1.3). As stated, the dataset provides 3D lidar labels for two camera perspectives onto the urban *S110* road intersections. For each camera, four scenes of varying length (between 300 and 1200 frames) are available. The combination of two camera perspectives and four scenes yields eight frame sequences on which we evaluate our detector. For each frame sequence, the detector is run in different configurations to evaluate the effect of various design choices on the final performance. The detector is configured along four major components:

1. **Instance Segmentation**: In the first detector stage, we can employ the *Yolact* [Liu+21] instance segmentation model (running on 550x550 input frames), or the *YoloV7* [WBL22] detector in $640x640$, $1280x180$, or $1920x1920$ px resolution mode. We respectively designate these modes as I^{550}_{YOL}, I^{640}_{Yv7}, I^{1280}_{Yv7}, and I^{1920}_{Yv7}.

2. **Tracking**: This component concerns the usage of *SORT tracking* [Bew+16] in our detector. We can apply tracking in $2D$ screen-space to assist the LSF algorithm, or in $3D$ space to stabilize vehicle position estimates, or both in $2D$ and $3D$, or not at all. We respectively designate these modes as T_{2D}, T_{3D}, T^{2D}_{3D}, and T_0.

3. **HD Map Usage**: In this aspect, we consider whether to use heading information from the HD map as an early input to the LSF algorithm, as a late single-position lookup correction, or not at all. These modes are designated as M_{LSF}, M_1 and M_0.

4. **Filtering**: Finally, we can switch two crucial filters on or off: The *DBSCAN* [Sch+17] bottom contour point filter, and the output vehicle size filter. These modes will be symbolized as F_{Cont}, F_{Size}, $F_{\text{Cont}}^{\text{Size}}$, or F_0 (if no filter is used).

In summary, each detector configuration is some combination of I, T, M and F. For example, $\left[I_{\text{Yv7}}^{640} T_{3D}^{2D} M_{\text{LSF}} F_{\text{Size}}\right]$ would be the detector running with 640x640 px *Yolov7* instance segmentation, both 2D and 3D tracking, L-Shape-Fitting Map Input, and vehicle size filtering (but no bottom-contour filtering). The combinatorial expansion yields 192 possible detector configurations, which are executed on each of the 8 LiDAR-labeled camera frame sequences. This results in 1536 prediction sequences.

Furthermore, we evaluate the impact which emerges from the sensor delay between the camera and the LiDAR sensor frames. This delay is roughly 18.54 ms on average. This synchronization error is important, because we evaluate the 3D camera detections on labeled LiDAR sensor measurements. These LiDAR labels will contain an inherent offset towards the camera detections due to the synchronization error. We check whether the error can be reduced by estimating the spatial velocity of the LiDAR detections using *SORT*, and then correcting the label position based on the velocity and the known synchronization error time delta.

We evaluate both the time-shifted and the original LiDAR labels with each prediction sequence. We call these two label modes L_0 (original) and $L_\uparrow$ (shifted). The label mode is appended to the detector configuration combination. For each of the 3072 sequence-detector-label combinations and each road user class, we calculate the following metrics:

1. The average orientation error **(AOE)** ($= \text{TP}_0$).
2. The average translation error **(ATE)** ($= \text{TP}_1$.
3. The average length error **(ALE)** ($= \text{TP}_2$).
4. The average width error **(AWE)** ($= \text{TP}_3$).
5. The average height error **(AHE)** ($= \text{TP}_4$).
6. The average precision at 10% minimum $\textbf{IoU}_{3D}$ **(AP@10)**.
7. The average 3D intersection-over-union ($\textbf{IoU}_{3D}$)

In the following, the values for these metrics are presented in relation to particular detector- and labeling-configurations. To judge the overall performance of a configuration, we aggregate these metrics into a single "Providentia Detection Score" (*PDS*), similar to the *nuScenes Detection Score* (see Section 3.8):

$$\text{PDS} = \frac{1}{10}\left[5 * \text{mAP} + \sum_{k=1}^{5}(1 - \min(1, \text{mTP}_k))\right]$$

The score is calculated for each configuration, by averaging the per-class metrics of the configuration across all perspectives of all dataset scenes. A chart of all calculated Scores is shown in Figure 5.4. We consider the following object classes: `CAR`, `BUS`, `TRUCK`, `MOTORCYCLE`, `BICYCLE`, `PEDESTRIAN`. Additionally, the A9 dataset contains annotations for `VAN`, `EMERGENCY_VEHICLE`, `TRAILER` and `OTHER`. However, these categories are not supported

by our instance segmentation models, which will usually label these special vehicles as trucks or cars. To gain a better understanding of our model's raw shape-estimation performance, we also evaluate using the `VEHICLE` super-class. With the super-class, a `VAN` labeled as a `TRUCK` will still be considered as a True Positive detection. Note that we skip the AOE metric for the pedestrians and bicycles, as we do not attempt to estimate their heading.

5.4 Overall Quantitative Results

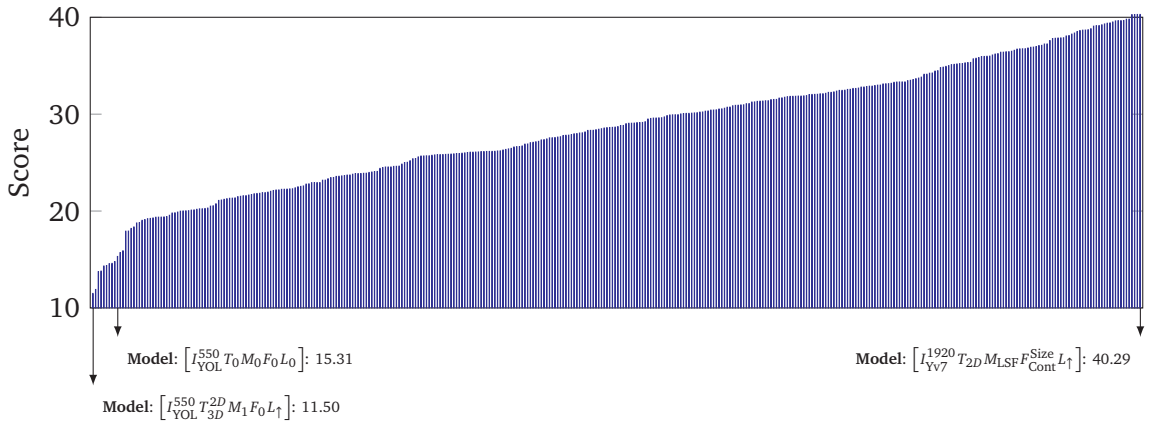


Figure 5.4: Overview for the *Providentia Detection Score (PDS)* values of all possible 384 detector configuration-label combinations, with arrows pointing out the best, baseline, and worst combination.

To establish the veracity of our score metric, we plotted the score values for all possible 384 detector configuration-label combinations into a single chart. This plot is shown in Figure 5.4. In this plot, the Scores of the worst-performing- ($\left[I_{\text{YOL}}^{550} T_{3D}^{2D} M_1 F_0 L_\uparrow\right]$), baseline- ($\left[I_{\text{YOL}}^{550} T_0 M_0 F_0 L_0\right]$), and best-performing ($\left[I_{\text{Yv7}}^{1920} T_{2D} M_{\text{LSF}} F_{\text{Cont}}^{\text{Size}} L_\uparrow\right]$) model-configurations are also highlighted. The baseline model works with pure L-Shape-Fitting that does not use any HD map, tracking, or bottom-contour/size filtering augmentations. Table 5.2 shows the performance of this model in greater detail. Of particular interest is the large AOE of 52.19°. This means, that the predicted orientation of bounding boxes in this model is completely disconnected from the ground truth. Picking a random orientation would yield the same result.

	Model: $\left[I_{\text{YOL}}^{550} T_0 M_0 F_0 L_0\right]$						**PDS**: 15.31%		
Class	**AOE**	**ATE**	**AWE**	**ALE**	**AHE**	**IoU$_{3D}$**	**Precision**	**Recall**	**AP@10**
Car	57.39°	1.10 m	0.49 m	1.03 m	0.69 m	21.80%	47.60%	40.57%	46.85%
Truck	34.61°	2.17 m	0.77 m	4.15 m	1.31 m	20.31%	3.55%	1.67%	2.86%
Motorcycle	71.98°	0.63 m	0.76 m	1.31 m	0.13 m	14.72%	31.14%	22.98%	30.68%
Bus	44.76°	1.85 m	1.47 m	3.07 m	1.55 m	22.47%	17.47%	10.51%	16.19%
Pedestrian	—	0.38 m	0.25 m	0.18 m	0.07 m	21.62%	0.88%	0.10%	0.65%
Bicycle	—	0.34 m	0.65 m	0.46 m	0.10 m	20.46%	0.92%	0.17%	0.67%
Mean	**52.19°**	**1.08 m**	**0.73 m**	**1.70 m**	**0.64 m**	**20.23%**	**16.93%**	**12.67%**	**16.31%**

Table 5.2: Baseline model results.

This stands in contrast to the performance metrics for the best-performing model, which are shown in Table 5.3. Additionally, Table 5.4 shows the results for the best vehicle-only

model. In the following sections, we will discuss in detail how specific model configurations improve (or worsen) specific metrics for different road user classes.

	Model: $\left[I^{1920}_{\text{Yv7}} T_{2D} M_{\text{LSF}} F^{\text{Size}}_{\text{Cont}} L_{\uparrow}\right]$						**PDS:** 40.29% (+24.98%)		
Class	**AOE**	**ATE**	**AWE**	**ALE**	**AHE**	**IoU$_{3D}$**	**Precision**	**Recall**	**AP@10**
Car	3.13°	0.85 *m*	0.25 *m*	0.89 *m*	0.35 *m*	39.19%	72.76%	73.19%	72.26%
Truck	4.16°	1.69 *m*	0.62 *m*	2.52 *m*	0.58 *m*	17.67%	23.47%	28.83%	22.84%
Motorcycle	3.85°	0.78 *m*	0.31 *m*	0.63 *m*	0.14 *m*	26.91%	34.22%	30.43%	33.88%
Bus	10.35°	1.15 *m*	0.55 *m*	2.93 *m*	1.08 *m*	33.55%	51.71%	35.50%	50.74%
Pedestrian	—	0.38 *m*	0.28 *m*	0.20 *m*	0.07 *m*	32.26%	20.84%	13.55%	20.45%
Bicycle	—	0.56 *m*	1.22 *m*	0.69 *m*	0.08 *m*	21.99%	33.75%	35.33%	33.49%
Mean	**5.37°**	**0.90 m**	**0.54 m**	**1.31 m**	**0.38 m**	**28.59%**	**39.46%**	**36.14%**	**38.94%**
Δ Baseline	−46.81°	−0.18 *m*	−0.19 *m*	−0.39 *m*	−0.26 *m*	+8.36%	+22.53%	+23.47%	+22.63%

Table 5.3: Best model results, with improvements towards the baseline highlighted in green for each metric.

	Model: $\left[I^{1920}_{\text{Yv7}} T_{2D} M_{\text{LSF}} F^{\text{Size}}_{\text{Cont}} L_{\uparrow}\right]$						**PDS:** 50.07% (+26.40%)		
Class	**AOE**	**ATE**	**AWE**	**ALE**	**AHE**	**IoU$_{3D}$**	**Precision**	**Recall**	**AP@10**
Vehicle	3.47°	0.96 *m*	0.33 *m*	1.30 *m*	0.44 *m*	36.79%	56.72%	48.18%	55.90%
Δ Baseline	−52.79°	−0.25 *m*	−0.35 *m*	−0.27 *m*	−0.36 *m*	+15.41%	+19.22%	+21.32%	+19.36%

Table 5.4: Best model results focusing on the `Vehicle` super-category, with improvements towards the vehicle-only baseline highlighted in green for each metric.

5.5 Influence of Late HD Map Lookup

In this section, we present the evaluation of the "Late HD Map Lookup" technique, a simpler alternative to the "early" HD map lookup. As mentioned in Section 4.11, this approach has several drawbacks which are expected to negatively impact the system's performance. The purpose of this evaluation is to provide a clear understanding of the limitations and consequences of using the late lookup approach. This was the first approach we tried to improve upon the baseline detector. The metrics of a Yolact-based detector which simply adds late HD map-lookup-based heading correction are presented in Table 5.5.

	Model: $\left[I^{550}_{\text{YOL}} T_0 M_1 F_0 L_0\right]$						**PDS:** 19.56% (+4.25%)		
Class	**AOE**	**ATE**	**AWE**	**ALE**	**AHE**	**IoU$_{3D}$**	**Precision**	**Recall**	**AP@10**
Car	45.15°	1.13 *m*	0.48 *m*	1.10 *m*	0.68 *m*	27.26%	46.89%	39.65%	46.12%
Truck	50.81°	2.47 *m*	0.88 *m*	4.90 *m*	1.18 *m*	18.93%	4.27%	2.59%	3.55%
Motorcycle	13.04°	0.40 *m*	0.49 *m*	0.95 *m*	0.05 *m*	11.14%	36.05%	27.21%	35.77%
Bus	36.76°	1.81 *m*	1.32 *m*	3.63 *m*	1.62 *m*	25.44%	21.03%	14.67%	19.80%
Pedestrian	—	0.38 *m*	0.25 *m*	0.18 *m*	0.07 *m*	21.52%	0.88%	0.10%	0.65%
Bicycle	—	0.34 *m*	0.65 *m*	0.46 *m*	0.10 *m*	20.43%	0.92%	0.17%	0.67%
Mean	**36.44°**	**1.09 m**	**0.68 m**	**1.87 m**	**0.62 m**	**20.78%**	**18.34%**	**14.07%**	**17.76%**
Δ Baseline	−15.75°	+0.01 *m*	−0.05 *m*	+0.17 *m*	−0.02 *m*	+0.55%	+1.41%	+1.40%	+1.44%

Table 5.5: Results for the simplest possible late map lookup detector, with differences towards the baseline highlighted in red and green.

One of the main issues with the late HD map lookup is that it takes place after the L-Shape-Fitting has already determined a heading value. This means that the heading information from the HD map cannot be used to guide or correct the LSF-based size , leading to inaccuracies in the width and length estimation. This is also shown in the results, as none of the size or translation metrics are significantly improved.

Another significant drawback of the late HD map lookup is that it discards a lot of the positional information from the ground truth sensor input. By querying the HD map only for heading options at the predicted vehicle position, the system loses the ability to take advantage of the more accurate and granular bottom contour data from the sensor inputs. This loss of information is reflected in the barely improved orientation error.

5.6 Effects of YOLOv7 and Image Resolution

	Average Results Using Yolact-Edge						**PDS**: 32.51%		
Class	**AOE**	**ATE**	**AWE**	**ALE**	**AHE**	**IoU$_{3D}$**	**Precision**	**Recall**	**AP@10**
Vehicle	32.92°	1.05 *m*	0.48 *m*	1.52 *m*	0.62 *m*	30.66%	49.16%	39.74%	48.29%
Pedestrian	—	0.38 *m*	0.25 *m*	0.17 *m*	0.07 *m*	20.91%	0.85%	0.10%	0.63%
Bicycle	—	0.26 *m*	0.65 *m*	0.46 *m*	0.10 *m*	22.55%	0.92%	0.17%	0.67%
Mean	**32.92°**	**0.56 m**	**0.46 m**	**0.72 m**	**0.26 m**	**24.71%**	**16.97%**	**13.33%**	**16.53%**
	Average Results Using Yolov7 (640x640)						**PDS**: 31.65% (−0.86%)		
Class	**AOE**	**ATE**	**AWE**	**ALE**	**AHE**	**IoU$_{3D}$**	**Precision**	**Recall**	**AP@10**
Vehicle	32.89°	1.07 *m*	0.57 *m*	1.82 *m*	0.59 *m*	29.09%	48.00%	39.40%	47.11%
Pedestrian	—	0.33 *m*	0.22 *m*	0.17 *m*	0.07 *m*	24.41%	6.81%	1.23%	6.05%
Bicycle	—	0.52 *m*	1.09 *m*	0.62 *m*	0.10 *m*	27.09%	15.10%	6.40%	14.36%
Mean	**32.89°**	**0.64 m**	**0.63 m**	**0.87 m**	**0.25 m**	**26.86%**	**23.30%**	**15.68%**	**22.51%**
Δ Previous	−0.03°	+0.07 *m*	+0.17 *m*	+0.15 *m*	−0.01 *m*	+2.16%	+6.33%	+2.34%	+5.98%
	Average Results Using Yolov7 (1280x1280)						**PDS**: 35.38% (+3.73%)		
Class	**AOE**	**ATE**	**AWE**	**ALE**	**AHE**	**IoU$_{3D}$**	**Precision**	**Recall**	**AP@10**
Vehicle	32.91°	1.09 *m*	0.52 *m*	1.65 *m*	0.59 *m*	30.08%	48.22%	40.13%	47.27%
Pedestrian	—	0.26 *m*	0.22 *m*	0.17 *m*	0.06 *m*	28.98%	13.53%	7.38%	13.01%
Bicycle	—	0.66 *m*	1.20 *m*	0.64 *m*	0.08 *m*	21.87%	29.59%	32.15%	29.25%
Mean	**32.91°**	**0.67 m**	**0.64 m**	**0.82 m**	**0.24 m**	**26.98%**	**30.45%**	**26.55%**	**29.84%**
Δ Previous	+0.02°	+0.03 *m*	+0.02 *m*	−0.05 *m*	−0.01 *m*	+0.12%	+7.14%	+10.88%	+7.34%
	Average Results Using Yolov7 (1920x1920)						**PDS**: 36.72% (+1.33%)		
Class	**AOE**	**ATE**	**AWE**	**ALE**	**AHE**	**IoU$_{3D}$**	**Precision**	**Recall**	**AP@10**
Vehicle	33.15°	1.08 *m*	0.49 *m*	1.52 *m*	0.59 *m*	30.67%	46.79%	39.09%	45.90%
Pedestrian	—	0.37 *m*	0.28 *m*	0.21 *m*	0.07 *m*	32.28%	20.77%	13.58%	20.38%
Bicycle	—	0.58 *m*	1.20 *m*	0.65 *m*	0.08 *m*	22.65%	31.45%	31.95%	31.13%
Mean	**33.15°**	**0.68 m**	**0.66 m**	**0.79 m**	**0.25 m**	**28.53%**	**33.00%**	**28.21%**	**32.47%**
Δ Previous	+0.25°	+0.01 *m*	+0.01 *m*	−0.03 *m*	±0.00 *m*	+1.55%	+2.55%	+1.65%	+2.63%

Table 5.6: Average results for different instance segmentation models, with differences in the resulting metrics shown towards the previous model's average values.

The comparison of the average performance of detectors using different instance segmentation models highlights some interesting points, as shown in Table 5.6. Comparing the performance of the *Yolact-Edge* model with the Yolov7 models at different resolutions, we can see

that the Yolov7 models generally outperform the *Yolact-Edge* model. The *Yolov7* (640x640) model shows an mAP improvement of 5.98% compared to the *Yolact-Edge* model, albeit there is a small decrease in the score 0.86% due to slightly worse TP metric performance. Similarly, *Yolov7* (1280x1280) has a score improvement of 3.73% compared to *Yolov7* (640x640), and a higher mean AP of +7.34%.

The *Yolov7* (1920x1920) again outperforms the *Yolov7* (1280x1280) model on average, as it has a slightly higher overall score by 1.33%. Most notably, the pedestrian detection performance is greatly improved by the *Yolov7* (1920x1920) model, as its AP for pedestrian detection is at 20.38%, compared to 13.01% in the *Yolov7* (640x640) model and only 0.63% in the *Yolact-Edge* model.

The results indicate that higher-resolution input may not always result in better overall performance but can significantly improve the detection of specific object classes like pedestrians. This suggests that for certain applications or scenarios, focusing on higher resolutions could be beneficial, but not true for every use-case.

In summary, *Yolov7* (1920x1920) shows the overall best performance, but its improvement over *Yolov7* (1280x1280) mainly comes from improved pedestrian detection. The *Yolov7* models also generally outperform the *Yolact-Edge* model. For reference, Table 5.7 towards the end of this chapter also shows the results for the best model in each instance segmentation configuration.

	Best w/ Yolact-Edge: $\left[I_{\text{YOL}}^{550} T_{2D} M_{\text{LSF}} F_{\text{Cont}}^{\text{Size}} L_{\uparrow}\right]$						**PDS**: 41.97% (−4.04%)		
Class	**AOE**	**ATE**	**AWE**	**ALE**	**AHE**	**IoU$_{3D}$**	**Precision**	**Recall**	**AP@10**
Vehicle	3.46°	0.87 m	0.33 m	1.31 m	0.46 m	37.73%	58.32%	47.92%	57.56%
Pedestrian	—	0.38 m	0.25 m	0.14 m	0.06 m	20.16%	0.81%	0.09%	0.61%
Bicycle	—	0.17 m	0.65 m	0.46 m	0.10 m	24.25%	0.92%	0.17%	0.67%
Mean	**3.46°**	**0.48 m**	**0.41 m**	**0.63 m**	**0.20 m**	**27.38%**	**20.02%**	**16.06%**	**19.61%**
Δ Best	−0.01°	−0.16 m	−0.20 m	−0.10 m	+0.01 m	−2.97%	−17.09%	−16.29%	−17.00%
	Best w/ Yolov7 (640^2): $\left[I_{\text{Yv7}}^{640} T_0 M_{\text{LSF}} F_{\text{Size}} L_0\right]$						**PDS**: 41.17% (−4.83%)		
Class	**AOE**	**ATE**	**AWE**	**ALE**	**AHE**	**IoU$_{3D}$**	**Precision**	**Recall**	**AP@10**
Vehicle	3.80°	1.09 m	0.32 m	1.42 m	0.58 m	36.32%	59.87%	49.65%	59.11%
Pedestrian	—	0.34 m	0.22 m	0.19 m	0.07 m	23.67%	7.04%	1.27%	6.26%
Bicycle	—	0.51 m	1.07 m	0.57 m	0.10 m	28.97%	15.88%	7.07%	15.17%
Mean	**3.80°**	**0.65 m**	**0.54 m**	**0.73 m**	**0.25 m**	**29.65%**	**27.60%**	**19.33%**	**26.85%**
Δ Best	+0.33°	+0.02 m	−0.07 m	−0.01 m	+0.05 m	−0.70%	−9.50%	−13.02%	−9.76%
	Best w/ Yolov7 (1280^2): $\left[I_{\text{Yv7}}^{1280} T_{2D} M_{\text{LSF}} F_{\text{Size}} L_0\right]$						**PDS**: 44.29% (−1.72%)		
Class	**AOE**	**ATE**	**AWE**	**ALE**	**AHE**	**IoU$_{3D}$**	**Precision**	**Recall**	**AP@10**
Vehicle	3.51°	1.15 m	0.26 m	1.46 m	0.55 m	36.84%	58.89%	50.57%	58.09%
Pedestrian	—	0.26 m	0.22 m	0.18 m	0.06 m	28.96%	13.95%	7.45%	13.42%
Bicycle	—	0.67 m	1.18 m	0.59 m	0.08 m	22.68%	31.57%	36.26%	31.27%
Mean	**3.51°**	**0.70 m**	**0.55 m**	**0.74 m**	**0.23 m**	**29.49%**	**34.80%**	**31.43%**	**34.26%**
Δ Best	+0.04°	+0.06 m	−0.06 m	+0.01 m	+0.04 m	−0.86%	−2.30%	−0.92%	−2.35%

Table 5.7: Best results for different instance segmentation models, with differences in the resulting metrics shown towards the overall best model's values, which are those of I_{Yv7}^{1920}.

5.7 Impact of Filters

Furthermore, we evaluate the impact of *DBSCAN*-based bottom-contour filtering and size filtering on the detector performance. The results presented in Table 5.8 show the impact of different filter types on the 3D object detection task. Three different filter types are considered for comparison: no filters, contour filters, and size filters. For each filter type, the performance of the best-performing model is compared with a model that is equivalent but lacking one or both filters. The following conclusions can be drawn for these configurations:

The best model without any filters yields an overall score of 40.95%, which is the lowest among the three variations. All the metrics exhibit worse performance compared to the best model. This indicates that filters are essential for improving the model's performance.

When only contour filters are applied, the overall score increases to 44.26%. This is because vehicle sizes are harder to estimate from a filtered bottom contour. Nonetheless, the contour filters help to enhance the overall performance of the model.

Applying size filters results in an even higher overall score of 45.61%, which is only 0.40% lower than the score of the best model. The size filter leads to improvements in AWE, IoU, Precision, Recall and AP, while causing slight regressions in AOE, ATE, ALE, and AHE. The size filter demonstrates the most significant impact on the model's performance.

In summary, using filters in the 3D object detection task can substantially improve the model's performance. Among the different filter types, size filters show the greatest positive impact on the overall score, while bottom contour filters also provide a noticeable improvement.

	Best w/o Filters ($\left[I_{\text{Yv7}}^{1920} T_{2D} M_{\text{LSF}} F_0 L_0\right]$)						**PDS**: 40.95% (−5.06%)		
Class	**AOE**	**ATE**	**AWE**	**ALE**	**AHE**	**IoU$_{3D}$**	**Precision**	**Recall**	**AP@10**
Vehicle	3.50°	1.29 *m*	0.62 *m*	1.44 *m*	0.78 *m*	28.67%	45.86%	36.45%	44.93%
Pedestrian	—	0.37 *m*	0.29 *m*	0.22 *m*	0.07 *m*	32.68%	20.98%	13.55%	20.59%
Bicycle	—	0.61 *m*	1.18 *m*	0.60 *m*	0.08 *m*	22.89%	34.75%	36.78%	34.49%
Mean	**3.50°**	**0.76 m**	**0.69 m**	**0.75 m**	**0.31 m**	**28.08%**	**33.86%**	**28.93%**	**33.34%**
Δ Best	+0.03°	+0.12 *m*	+0.08 *m*	+0.02 *m*	+0.11 *m*	−2.27%	−3.24%	−3.43%	−3.28%
	Best w/ Contour Filter ($\left[I_{\text{Yv7}}^{1920} T_0 M_{\text{LSF}} F_{\text{Cont}} L_0\right]$)						**PDS**: 44.26% (−1.74%)		
Class	**AOE**	**ATE**	**AWE**	**ALE**	**AHE**	**IoU$_{3D}$**	**Precision**	**Recall**	**AP@10**
Vehicle	3.47°	0.95 *m*	0.46 *m*	1.39 *m*	0.54 *m*	32.65%	50.24%	41.41%	49.36%
Pedestrian	—	0.37 *m*	0.29 *m*	0.21 *m*	0.07 *m*	32.43%	20.76%	13.39%	20.38%
Bicycle	—	0.61 *m*	1.18 *m*	0.60 *m*	0.08 *m*	22.89%	34.72%	36.78%	34.46%
Mean	**3.47°**	**0.64 m**	**0.64 m**	**0.73 m**	**0.23 m**	**29.32%**	**35.24%**	**30.53%**	**34.74%**
Δ Best	±0.00°	+0.01 *m*	+0.03 *m*	±0.00 *m*	+0.04 *m*	−1.02%	−1.86%	−1.83%	−1.88%
	Best w/ Size Filter ($\left[I_{\text{Yv7}}^{1920} T_{2D} M_{\text{LSF}} F_{\text{Size}} L_0\right]$)						**PDS**: 45.61% (−0.40%)		
Class	**AOE**	**ATE**	**AWE**	**ALE**	**AHE**	**IoU$_{3D}$**	**Precision**	**Recall**	**AP@10**
Vehicle	3.51°	1.09 *m*	0.25 *m*	1.46 *m*	0.51 *m*	38.24%	57.50%	49.23%	56.71%
Pedestrian	—	0.37 *m*	0.29 *m*	0.22 *m*	0.07 *m*	32.68%	20.98%	13.55%	20.59%
Bicycle	—	0.61 *m*	1.18 *m*	0.60 *m*	0.08 *m*	22.89%	34.75%	36.78%	34.49%
Mean	**3.51°**	**0.69 m**	**0.57 m**	**0.76 m**	**0.22 m**	**31.27%**	**37.74%**	**33.18%**	**37.26%**
Δ Best	+0.04°	+0.06 *m*	−0.04 *m*	+0.02 *m*	+0.03 *m*	+0.92%	+0.64%	+0.83%	+0.65%

Table 5.8: Ablated results for models which use at most one of the *DBSCAN*-based bottom contour filter or size filter, compared with the best model which uses both filters. It is apparent, that the filtering mostly has a positive effect across all metrics, and the filters also synergize with each other, as none of the ablated model beats the model which uses all filters.

5.8 Contribution of L-Shape-Fitting Augmentations

In this section, we will discuss the impact of different LSF augmentations on the detector's performance. Two augmentations are tested: HD Map Input and Screen-Space Tracking (Historical Plausibility). We will analyze the results of these augmentations and compare them with the best performance achieved without LSF Augments. As an initial overview, Figure 5.6 provides a qualitative impression of these different configurations. Furthermore, Figure 5.5 provides a rough indication for the score distribution of different augmentation configurations.

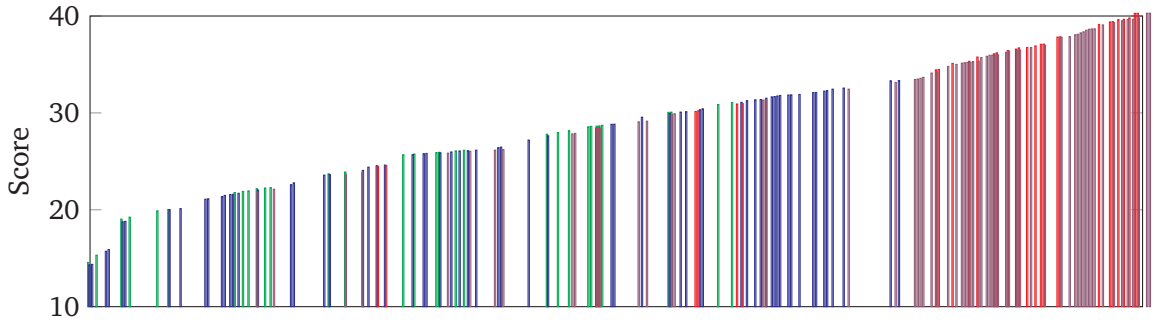


Figure 5.5: Overview for model performances across different LSF augmentation configurations. The bars for a model is color-coded by their usage of Raw LSF without Augmentation (green), LSF with screen-space tracking (blue), LSF with HD map augmentation (red), or LSF with both augmentations (violet).

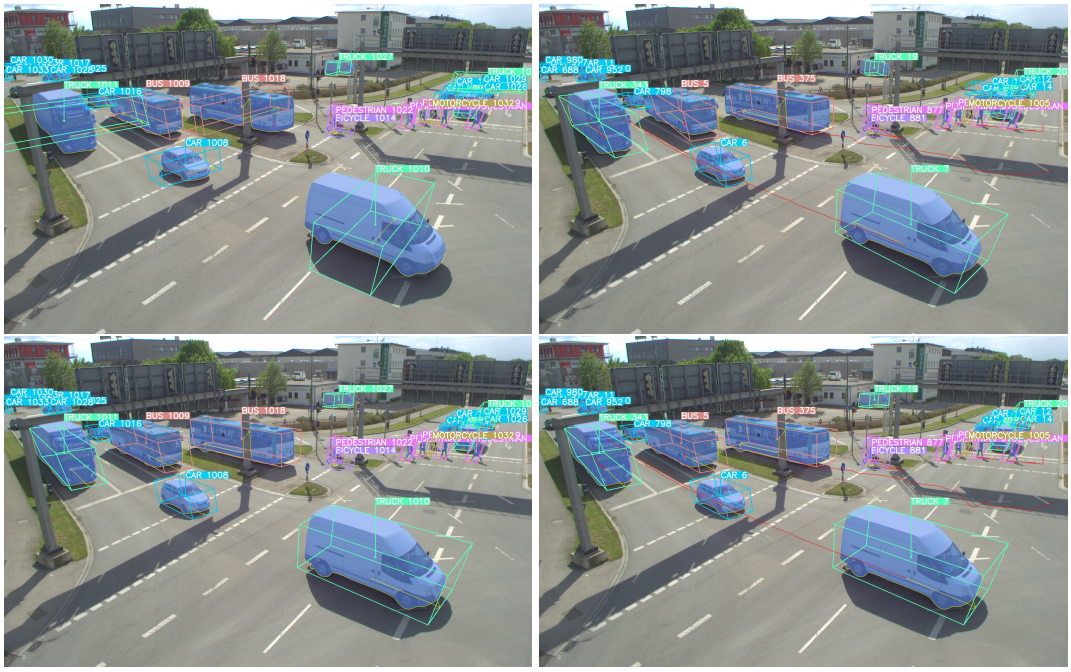


Figure 5.6: Comparison of the same frame from our best model ($\left[I^{1920}_{\text{Yv7}} T_{2D} M_{\text{LSF}} F^{\text{Size}}_{\text{Cont}}\right]$) (bottom-right), vs. the same model except with disabled LSF augmentations (top-left), disabled map augmentation (top-right), or disabled tracking augmentation (bottom-left).

	Best w/o LSF Augments ($\left[I_{\text{Yv7}}^{1920} T_0 M_0 F_{\text{Cont}}^{\text{Size}} L_\uparrow\right]$)						**PDS**: 36.34% (−9.67%)		
Class	**AOE**	**ATE**	**AWE**	**ALE**	**AHE**	**IoU$_{3D}$**	**Precision**	**Recall**	**AP@10**
Vehicle	59.54°	0.94 *m*	0.40 *m*	1.26 *m*	0.43 *m*	28.60%	55.02%	46.65%	54.24%
Pedestrian	—	0.38 *m*	0.28 *m*	0.20 *m*	0.07 *m*	32.26%	20.82%	13.55%	20.44%
Bicycle	—	0.56 *m*	1.22 *m*	0.69 *m*	0.08 *m*	21.99%	33.73%	35.33%	33.46%
Mean	**59.54°**	**0.63 m**	**0.63 m**	**0.72 m**	**0.19 m**	**27.62%**	**36.52%**	**31.84%**	**36.05%**
Δ Best	+56.08°	−0.01 *m*	+0.02 *m*	−0.01 *m*	±0.00 *m*	−2.73%	−0.58%	−0.51%	−0.56%
	Best w/ LSF Tracking-Aug. ($\left[I_{\text{Yv7}}^{1920} T_{2D} M_0 F_{\text{Cont}}^{\text{Size}} L_\uparrow\right]$)						**PDS**: 38.02% (−7.99%)		
Class	**AOE**	**ATE**	**AWE**	**ALE**	**AHE**	**IoU$_{3D}$**	**Precision**	**Recall**	**AP@10**
Vehicle	48.41°	0.95 *m*	0.37 *m*	1.28 *m*	0.45 *m*	33.83%	56.27%	47.76%	55.44%
Pedestrian	—	0.38 *m*	0.28 *m*	0.20 *m*	0.07 *m*	32.26%	20.84%	13.55%	20.45%
Bicycle	—	0.56 *m*	1.22 *m*	0.69 *m*	0.08 *m*	21.99%	33.75%	35.33%	33.49%
Mean	**48.41°**	**0.63 m**	**0.62 m**	**0.73 m**	**0.20 m**	**29.36%**	**36.95%**	**32.21%**	**36.46%**
Δ Best	+44.94°	±0.00 *m*	+0.01 *m*	−0.01 *m*	±0.00 *m*	−0.99%	−0.15%	−0.14%	−0.15%
	Best w/ LSF Map-Aug. ($\left[I_{\text{Yv7}}^{1920} T_0 M_{\text{LSF}} F_{\text{Cont}}^{\text{Size}} L_\uparrow\right]$)						**PDS**: 45.98% (−0.03%)		
Class	**AOE**	**ATE**	**AWE**	**ALE**	**AHE**	**IoU$_{3D}$**	**Precision**	**Recall**	**AP@10**
Vehicle	3.63°	0.96 *m*	0.33 *m*	1.30 *m*	0.44 *m*	36.77%	56.72%	48.21%	55.90%
Pedestrian	—	0.38 *m*	0.28 *m*	0.20 *m*	0.07 *m*	32.26%	20.82%	13.55%	20.44%
Bicycle	—	0.56 *m*	1.22 *m*	0.69 *m*	0.08 *m*	21.99%	33.73%	35.33%	33.46%
Mean	**3.63°**	**0.63 m**	**0.61 m**	**0.73 m**	**0.19 m**	**30.34%**	**37.09%**	**32.36%**	**36.60%**
Δ Best	+0.16°	±0.00 *m*	±0.00 *m*	±0.00 *m*	±0.00 *m*	−0.01%	−0.01%	+0.01%	−0.01%

Table 5.9: Ablation study for the best model $\left[I_{\text{Yv7}}^{1920} T_{2D} M_{\text{LSF}} F_{\text{Cont}}^{\text{Size}}\right]$, when none or just one of the Screen-Space-Tracking or HD-Map-Lookup Augmentations for the L-Shape-Fitting (LSF) algorithm is used.

5.8.1 Impact of HD Map Augmentation

Table 5.9 provides ablated performance metrics for the best-performing model. First off, when no augmentations are used, the same model's performance drops by 9.67%. When only the HD map augmentation is used, the model reaches a Score of 40.11%, which is only 0.06% lower than that of the best model, which uses both LSF Augments. The mean Average Orientation Error (AOE) is significantly reduced to 3.63 °, which is still below the best's performance by 0.16 °. Overall, the HD-map-input LSF augmentation demonstrates a very positive impact on the detector's performance, but it alone does not beat the top model.

5.8.2 Impact of Screen-Space Tracking Augmentation

When the best-performing uses only the Screen-Space Tracking LSF augmentation it reaches a Score of 38.02%, which is 7.99% lower than the best model. Compared to the best un-augmented model, the mean AOE is slightly improved to 48.41°. Overall, the sole screen-space-tracking LSF augmentation shows some improvements towards the un-augmented model but does not reach the best's performance. We can conclude that Tracking alone is insufficient as a bias for vehicle orientation estimation. However, the final best model still uses this augmentation in conjunction with the HD map, and thereby manages to reach the highest score.

5.9 Significance of 3D Tracking

In this section, we discuss the evaluation results for the 3D SORT tracking technique, as described in Section 4.12 and presented in Table 5.10.

The results demonstrate that the 3D SORT tracking approach led to a decrease in overall performance compared to the best configuration without 3D tracking. Although there were some minor improvements in certain aspects, such as a decrease in Absolute Translation Error (ATE) by 0.04 *m*, the overall score dropped by 1.07%, with reductions in Precision, Recall, and mAP. In conclusion, the 3D SORT tracking technique, as implemented in this study, did not improve the overall performance. As mentioned in Section 4.12, a more capable Kalman Filter that takes vehicle heading into account is needed for better performance in 3D tracking.

	Best w/ 3D Tracking ($[I^{1920}_{\text{Yv7}} T_{3D} M_{\text{LSF}} F^{\text{Size}}_{\text{Cont}} L_{\uparrow}]$)						**PDS**: 39.22% (−1.07%)		
Class	**AOE**	**ATE**	**AWE**	**ALE**	**AHE**	**IoU$_{3D}$**	**Precision**	**Recall**	**AP@10**
Car	3.46°	0.85 *m*	0.29 *m*	1.19 *m*	0.35 *m*	36.12%	71.35%	72.39%	70.85%
Truck	4.14°	1.47 *m*	0.74 *m*	2.03 *m*	0.68 *m*	18.44%	17.87%	22.40%	17.30%
Motorcycle	4.04°	0.78 *m*	0.35 *m*	0.62 *m*	0.14 *m*	24.26%	33.61%	30.23%	33.28%
Bus	7.29°	1.14 *m*	0.51 *m*	1.99 *m*	1.23 *m*	36.89%	48.12%	31.80%	47.09%
Pedestrian	—	0.38 *m*	0.28 *m*	0.20 *m*	0.06 *m*	32.06%	20.93%	13.70%	20.54%
Bicycle	—	0.55 *m*	1.22 *m*	0.69 *m*	0.08 *m*	22.22%	33.77%	35.44%	33.52%
Mean	**4.73°**	**0.86 m**	**0.56 m**	**1.12 m**	**0.42 m**	**28.33%**	**37.61%**	**34.33%**	**37.10%**
Δ Best	−0.64°	−0.04 *m*	+0.02 *m*	−0.19 *m*	+0.04 *m*	−0.26%	−1.85%	−1.81%	−1.85%

Table 5.10: Best result for model which uses 3D Tracking, compared with the best model.

	Average Results Using Original Labels						**PDS**: 28.13%		
Class	**AOE**	**ATE**	**AWE**	**ALE**	**AHE**	**IoU$_{3D}$**	**Precision**	**Recall**	**AP@10**
Car	32.47°	0.92 *m*	0.39 *m*	1.09 *m*	0.45 *m*	31.27%	60.24%	58.87%	59.61%
Truck	29.86°	2.08 *m*	0.78 *m*	4.34 *m*	0.82 *m*	18.59%	12.91%	13.27%	12.26%
Motorcycle	34.00°	0.63 *m*	0.35 *m*	0.63 *m*	0.12 *m*	23.75%	34.79%	27.87%	34.35%
Bus	29.55°	1.48 *m*	0.93 *m*	2.99 *m*	1.21 *m*	27.65%	38.20%	25.15%	37.15%
Pedestrian	—	0.34 *m*	0.24 *m*	0.20 *m*	0.07 *m*	26.39%	10.52%	5.55%	10.05%
Bicycle	—	0.53 *m*	1.02 *m*	0.55 *m*	0.09 *m*	23.87%	19.50%	17.97%	19.09%
Mean	**31.47°**	**1.00 m**	**0.62 m**	**1.63 m**	**0.46 m**	**25.25%**	**29.36%**	**24.78%**	**28.75%**
	Average Results Using Time-Shifted Labels						**PDS**: 28.16% (+0.02%)		
Class	**AOE**	**ATE**	**AWE**	**ALE**	**AHE**	**IoU$_{3D}$**	**Precision**	**Recall**	**AP@10**
Car	32.44°	0.90 *m*	0.39 *m*	1.08 *m*	0.45 *m*	31.50%	61.02%	59.71%	60.38%
Truck	30.03°	2.10 *m*	0.78 *m*	4.35 *m*	0.83 *m*	17.98%	12.64%	13.29%	12.09%
Motorcycle	33.93°	0.62 *m*	0.35 *m*	0.65 *m*	0.15 *m*	23.23%	35.59%	28.06%	35.15%
Bus	29.60°	1.41 *m*	0.98 *m*	3.01 *m*	1.25 *m*	27.62%	38.14%	25.14%	37.09%
Pedestrian	—	0.33 *m*	0.24 *m*	0.17 *m*	0.06 *m*	26.91%	10.45%	5.58%	9.99%
Bicycle	—	0.47 *m*	1.05 *m*	0.63 *m*	0.09 *m*	23.21%	19.03%	17.37%	18.62%
Mean	**31.50°**	**0.97 m**	**0.63 m**	**1.65 m**	**0.47 m**	**25.08%**	**29.48%**	**24.86%**	**28.89%**
Δ Previous	+0.03°	−0.02 *m*	+0.01 *m*	+0.01 *m*	+0.01 *m*	−0.18%	+0.12%	+0.08%	+0.14%

Table 5.11: Average results for evaluations with time-shifted labels compared to the original unsynchronized LiDAR labels.

5.10 Implications of LiDAR Label Shifting

In this section, we discuss the evaluation results of the LiDAR label shifting technique, as described in Section 5.3 and presented in Table 5.11.

The results indicate that using time-shifted LiDAR labels ($L_{\uparrow}$) led to a minor improvement in the overall performance compared to using original LiDAR labels (L_0). The overall score increased by 0.02%, with small improvements in Precision, Recall, and mAP.

Overall, the LiDAR label shifting technique using SORT for estimating spatial velocity and correcting label positions based on the known synchronization error time delta led to a slight improvement in overall performance. Although the improvements are minor, they demonstrate that accounting for sensor delay between camera and LiDAR sensor frames can help reduce the inherent offset in the LiDAR labels and improve evaluation accuracy.

5.11 Challenges of Night-time Conditions

As discussed in the qualitative analysis, the detection performance of a system that relies on an RGB camera operating within the human visible light spectrum is expected to be worse at night due to the reduced visibility. The results in Table 5.12 quantitatively confirm this expectation, showing that both *Yolact-Edge* and *Yolov7* models exhibit degraded performance under night-time conditions.

	Average Night-Time Results Using Yolact-Edge						**PDS**: 21.13%		
Class	**AOE**	**ATE**	**AWE**	**ALE**	**AHE**	**IoU$_{3D}$**	**Precision**	**Recall**	**AP@10**
Car	40.07°	0.94 *m*	0.32 *m*	1.29 *m*	0.54 *m*	31.57%	39.74%	26.44%	38.55%
Truck	22.71°	2.33 *m*	0.38 *m*	3.93 *m*	1.31 *m*	18.70%	0.65%	0.19%	0.44%
Motorcycle	—	—	—	—	—	—	—	—	—
Bus	30.01°	1.22 *m*	0.83 *m*	2.16 *m*	1.48 *m*	29.71%	31.99%	18.15%	30.76%
Pedestrian	—	—	—	—	—	—	—	—	—
Bicycle	—	—	—	—	—	—	—	—	—
Mean	**30.93°**	**1.50 m**	**0.51 m**	**2.46 m**	**1.11 m**	**26.66%**	**24.12%**	**14.93%**	**23.25%**
	Average Night-Time Results Using Yolov7						**PDS**: 24.99% (+3.86%)		
Class	**AOE**	**ATE**	**AWE**	**ALE**	**AHE**	**IoU$_{3D}$**	**Precision**	**Recall**	**AP@10**
Car	34.38°	0.96 *m*	0.38 *m*	1.20 *m*	0.48 *m*	29.62%	45.17%	37.03%	44.20%
Truck	24.64°	1.46 *m*	0.41 *m*	2.69 *m*	0.58 *m*	11.80%	3.82%	2.25%	3.34%
Motorcycle	—	—	—	—	—	—	—	—	—
Bus	26.71°	1.28 *m*	0.92 *m*	2.54 *m*	1.07 *m*	28.34%	30.36%	15.65%	29.16%
Pedestrian	—	—	—	—	—	—	—	—	—
Bicycle	—	—	—	—	—	—	—	—	—
Mean	**28.58°**	**1.23 m**	**0.57 m**	**2.14 m**	**0.71 m**	**23.25%**	**26.45%**	**18.31%**	**25.57%**
Δ Previous	−2.35°	−0.27 *m*	+0.06 *m*	−0.31 *m*	−0.40 *m*	−3.41%	+2.33%	+3.38%	+2.32%

Table 5.12: Average results for evaluations on the night-time scene of the A9 dataset with *Yolact-Edge* and *Yolov7* models.

When comparing the night-time results, it is evident that the *Yolov7* model outperforms the Yolact-Edge model in most of the considered evaluation metrics. The overall score for the Yolov7 model is 24.99%, which is an improvement of 3.86% compared to the Yolact-Edge model. This improvement can be primarily attributed to the higher precision and recall rates, as well as a reduced average translation error (ATE) and average length error (ALE).

Despite the degraded performance at night, the *Yolov7* instance segmentation model still demonstrates a reasonable level of detection accuracy for certain object classes, such as cars and buses. However, for some classes like trucks, the average precision drops significantly, indicating a higher rate of false positives and false negatives.

It is also important to note that the performance for certain object classes like motorcycles, pedestrians, and bicycles is not reported in the night-time evaluation. This is due to a lack of evaluation samples for these classes in the night-time dataset scene.

In summary, the quantitative evaluation confirms the expected reduction in detection performance under night-time conditions. While the *Yolov7* model demonstrates better performance compared to the *Yolact-Edge* model, further research and development are necessary to improve the overall performance of instance segmentation models in low-light and night-time environments.

5.12 Perspective-Dependent Detector Performance

The detection performance can be significantly influenced by the camera perspective. In this work, we analyze the impact of two different camera perspectives from the A9 test field: `S110-S1` and `S110-S2`. The `S110-S1` camera is focused downwards onto the intersection, whereas the `S110-S2` camera has a longer detection range and is angled towards the horizon. In this section, we provide a quantitative comparison of the best-performing models for each perspective, focusing on the vehicle super-category, which emphasizes the shape detection performance rather than the object classification aspect.

Table 5.13 shows the quantitative results for the best-performing vehicle-only models for each camera perspective.

	Best for S110-S1 Perspective: $\left[I_{\text{YOL}}^{550} T_0 M_{\text{LSF}} F_{\text{Cont}}^{\text{Size}} L_0\right]$						**PDS**: 55.60% (+5.53%)		
Class	**AOE**	**ATE**	**AWE**	**ALE**	**AHE**	**IoU$_{3D}$**	**Precision**	**Recall**	**AP@10**
Vehicle	3.90°	1.04 *m*	0.31 *m*	1.34 *m*	0.34 *m*	39.90%	66.05%	55.82%	65.54%
Δ Best	+0.43°	+0.08 *m*	−0.02 *m*	+0.03 *m*	−0.10 *m*	+3.11%	+9.33%	+7.64%	+9.64%
	Best for S110-S2 Perspective: $\left[I_{\text{Yv7}}^{640} T_{3D}^{2D} M_{\text{LSF}} F_{\text{Size}} L_0\right]$						**PDS**: 50.90% (+0.83%)		
Class	**AOE**	**ATE**	**AWE**	**ALE**	**AHE**	**IoU$_{3D}$**	**Precision**	**Recall**	**AP@10**
Vehicle	3.43°	0.93 *m*	0.29 *m*	1.49 *m*	0.66 *m*	35.52%	61.39%	51.79%	60.69%
Δ Best	−0.04°	−0.02 *m*	−0.04 *m*	+0.19 *m*	+0.22 *m*	−1.27%	+4.67%	+3.62%	+4.79%

Table 5.13: Quantitative results for the best-performing vehicle-only models for each camera perspective.

For the `S110-S1` perspective, the best vehicle-only model achieves a score of 55.60%, which is an improvement of 5.53% over the best model for both perspectives. Most interestingly, this best model uses *Yolact-Edge*. But as Table 5.14 shows, this model does not transfer its performance to `S110-S2`.

For the `S110-S2` perspective, the best vehicle-only model obtains a score of 50.90%, with a performance improvement of 0.83% over the generally best model. Although the improvements in the performance metrics are not as significant as for the `S110-S1` perspective, the model still outperforms the best general model for this camera perspective.

In conclusion, the choice of camera perspective plays a crucial role in the detection performance of a model. In our case, the best-performing models for each perspective individually beat the best-performing general model. Therefore, it is essential to consider the camera perspective when designing and evaluating object detection models, as it can significantly impact the overall performance.

	Performance of $\left[I^{550}_{\mathbf{YOL}}\, T_0 M_{\mathbf{LSF}} F^{\mathbf{Size}}_{\mathbf{Cont}} L_0\right]$ on S110-S2						**PDS**: 48.63% (−1.44%)		
Class	**AOE**	**ATE**	**AWE**	**ALE**	**AHE**	**IoU$_{3D}$**	**Precision**	**Recall**	**AP@10**
Vehicle	3.48°	0.79 m	0.33 m	1.32 m	0.51 m	35.28%	52.28%	41.90%	51.32%
Δ Best	+0.02°	−0.17 m	±0.00 m	+0.01 m	+0.08 m	−1.51%	−4.44%	−6.28%	−4.57%
	Performance of $\left[I^{640}_{\mathbf{Yv7}}\, T^{2D}_{3D} M_{\mathbf{LSF}} F_{\mathbf{Size}} L_0\right]$ on S110-S1						**PDS**: 49.77% (−0.30%)		
Class	**AOE**	**ATE**	**AWE**	**ALE**	**AHE**	**IoU$_{3D}$**	**Precision**	**Recall**	**AP@10**
Vehicle	4.31°	1.23 m	0.35 m	1.37 m	0.50 m	37.04%	58.70%	47.35%	57.89%
Δ Best	+0.84°	+0.27 m	+0.02 m	+0.07 m	+0.06 m	+0.25%	+1.99%	−0.83%	+2.00%
	Performance of $\left[I^{1920}_{\mathbf{Yv7}}\, T_{2D} M_{\mathbf{LSF}} F^{\mathbf{Size}}_{\mathbf{Cont}} L_{\uparrow}\right]$ on S110-S1						**PDS**: 50.86% (+0.79%)		
Class	**AOE**	**ATE**	**AWE**	**ALE**	**AHE**	**IoU$_{3D}$**	**Precision**	**Recall**	**AP@10**
Vehicle	3.71°	1.10 m	0.30 m	1.24 m	0.37 m	38.39%	57.09%	48.27%	56.32%
Δ Best	+0.24°	+0.14 m	−0.03 m	−0.06 m	−0.07 m	+1.60%	+0.38%	+0.10%	+0.43%
	Performance of $\left[I^{1920}_{\mathbf{Yv7}}\, T_{2D} M_{\mathbf{LSF}} F^{\mathbf{Size}}_{\mathbf{Cont}} L_{\uparrow}\right]$ on S110-S2						**PDS**: 50.30% (+0.23%)		
Class	**AOE**	**ATE**	**AWE**	**ALE**	**AHE**	**IoU$_{3D}$**	**Precision**	**Recall**	**AP@10**
Vehicle	3.23°	0.82 m	0.36 m	1.36 m	0.51 m	35.19%	56.34%	48.08%	55.47%
Δ Best	−0.24°	−0.14 m	+0.03 m	+0.06 m	+0.07 m	−1.60%	−0.38%	−0.10%	−0.43%

Table 5.14: Results for perspective-specific best models on the respective other perspective.

Chapter 6

Conclusion

In the previous chapter, we evaluated various aspects of our proposed two-stage monocular infrastructure traffic object detection architecture, *MonoDet3d*. We analyzed the performance of different models on the *Providentia Intersection Scenario* dataset, using both detailed object categories and the vehicle super-category. Focusing on vehicles, the best model for the `S110-S1` perspective achieves a score of 55.60% (65.54% mAP), while for the `S110-S2` perspective, the best model scores 50.90% (60.69% mAP). The best model across all object categories and scenes exhibits a score of 40.29% (38.94% mAP).

Moreover, we provided a thorough ablation study of the architecture across all possible configuration regimes. This covered the impact of different L-Shape-Fitting augmentations, bottom-contour/size filters, different instance segmentation models, and various filter types. We also accounted for the synchronization lag between the camera frames and the LiDAR-sensor-based labels.

Despite the comprehensive evaluation, there are some shortcomings that can be addressed. Firstly, the early version of the *A9R1* dataset used in the evaluation has some issues, as some LiDAR labels are inaccurate, potentially affecting the model performance assessment. This might also explain the ceiling of 40% IoU, which we seemingly cannot overcome. Secondly, ablation studies for the VRU detection algorithm and the position-height regression algorithm are missing, which could provide more valuable insights into the system's performance and help identify areas for improvement.

In summary, our evaluation offers a thorough understanding of the system's performance, its limitations, and the challenges that arise from different camera perspectives and computational constraints. It may now be concluded without a doubt, that the HD map as an auxiliary bias for the L-Shape-Fitting algorithm offers a very significant improvement, guiding the proposed architecture into the realm of production-readiness. Without the HD map, the detector does not reach an acceptable orientation error. Therefore, the research hypothesis is confirmed insofar as concluding, that tracking alone does not substitute the HD map as a bias for deciding vehicle orientations. However, additional analysis and improvements can be made by addressing the mentioned shortcomings, to ultimately enhance the system's performance and reliability even further.

"Begin at the beginning," the King said gravely, "and go on till you come to the end: then stop."

— Lewis Carroll, *Alice in Wonderland*

Chapter 7

Future Work

In this chapter, we outline potential avenues for future work to further enhance the performance, capabilities, and robustness of our monocular 3d object detection system.

7.1 Improved Non-Maximum Suppression (NMS)

A common issue encountered in our system is the overlapping of bounding boxes, particularly for pedestrians inside bicycles, pedestrians inside vehicles, or vans inside trucks. To address this problem, we can explore more advanced non-maximum suppression (NMS) techniques that better handle these cases and suppress overlapping boxes, thus improving the system's overall accuracy and robustness.

7.2 Substituting HD Maps

Instead of relying solely on HD maps for obtaining heading information, we can explore deriving this information directly from the optical flow. This approach could offer a more adaptive and flexible solution to changing environments and potentially improve the system's performance in situations where the HD maps are unavailable or inaccurate.

7.3 Extended use of Kalman Filters

The current implementation of Kalman filters in our system does not include heading information in the state. This omission renders the current approach non-viable for accurate tracking and prediction de-noising. Extending the Kalman filter to include heading information would significantly improve its effectiveness in estimating the state of the tracked objects.

7.4 Improving Pedestrian/Cyclist Detection

The current solution for pedestrian and cyclist (VRU) detection is highly unoptimized, as the focus of this work was mostly directed towards road vehicle pose estimation. Future work should focus on optimizing and refining the detection algorithm, increasing its accuracy and robustness, and attempting to predict VRU orientations.

7.5 Neural Keypoint Estimation

The long tail robustness of our system can be improved by incorporating neural keypoint estimation techniques. This approach would help to better handle obstructions, occlusions, and unusual vehicle orientations, ultimately enhancing the system's performance in complex and dynamic traffic scenarios. It could also unify the VRU/Vehicle detection pipelines, and further improve the system's runtime performance. This presents itself as a very promising research direction.

7.6 Amodal Instance Segmentation

Amodal instance segmentation is a way if predicting instance masks while also completing masks where the detection is cut off or obstructed. This may be a viable alternative to key-points for object detection and segmentation, as it would be a less intrusive architectural change. While it may not be as robust as key-points in some cases, it might still present a valid approach to improve our system's overall performance. By investigating amodal instance segmentation techniques, we can potentially enhance the system's ability to detect and segment objects in challenging traffic scenarios.

"Optimism is a strategy for making a better future. Because unless you believe that the future can be better, you are unlikely to step up and take responsibility for making it so."

— Noam Chomsky

Bibliography

[AUK18] Althoff, M., Urban, S., and Koschi, M. “Automatic conversion of road networks from opendrive to lanelets”. In: *2018 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)*. IEEE. 2018, pp. 157–162.

[Bai+22] Bai, Z., Wu, G., Qi, X., Liu, Y., Oguchi, K., and Barth, M. J. “Infrastructure-based object detection and tracking for cooperative driving automation: A survey”. In: *2022 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2022, pp. 1366–1373.

[Bew+16] Bewley, A., Ge, Z., Ott, L., Ramos, F., and Upcroft, B. “Simple online and realtime tracking”. In: *2016 IEEE international conference on image processing (ICIP)*. IEEE. 2016, pp. 3464–3468.

[Blu22] Blumenthal, L. “Real-Time Monocular 3D Object Detection to Support Autonomous Driving”. Bachelors Thesis. Technische Universität München, 2022.

[Bol+19] Bolya, D., Zhou, C., Xiao, F., and Lee, Y. J. “Yolact: Real-time instance segmentation”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 9157–9166.

[Bra00] Bradski, G. “The OpenCV Library”. In: *Dr. Dobb’s Journal of Software Tools* (2000).

[BBB59] Brooks, F. P., Blaauw, G. A., and Buchholz, W. “Processing Data in Bits and Pieces”. In: *IRE Transactions on Electronic Computers* EC-8.2 (1959), pp. 118–124. DOI: 10.1109/TEC.1959.5219512.

[Cae+20] Caesar, H., Bankiti, V., Lang, A. H., Vora, S., Liong, V. E., Xu, Q., Krishnan, A., Pan, Y., Baldan, G., and Beijbom, O. “nuscenes: A multimodal dataset for autonomous driving”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 11621–11631.

[CW21] Carrillo, J. and Waslander, S. “Urbannet: Leveraging urban maps for long range 3d object detection”. In: *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. IEEE. 2021, pp. 3799–3806.

[Che+20] Chen, H., Sun, K., Tian, Z., Shen, C., Huang, Y., and Yan, Y. “Blendmask: Top-down meets bottom-up for instance segmentation”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 8573–8581.

[CK21] Creß, C. and Knoll, A. C. “Intelligent Transportation Systems With The Use of External Infrastructure: A Literature Survey”. In: *CoRR* abs/2112.05615 (2021). arXiv: 2112.05615. URL: https://arxiv.org/abs/2112.05615.

[DSG10] Dupuis, M., Strobl, M., and Grezlikowski, H. “Opendrive 2010 and beyond–status and future of the de facto standard for the description of road networks”. In: *Proc. of the Driving Simulation Conference Europe*. 2010, pp. 231–242.

[Guo+21] Guo, E., Chen, Z., Rahardja, S., and Yang, J. “3D Detection and Pose Estimation of Vehicle in Cooperative Vehicle Infrastructure System”. In: *IEEE Sensors Journal* 21.19 (2021), pp. 21759–21771. DOI: 10.1109/JSEN.2021.3101497.

[Har+20] Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[He+17] He, K., Gkioxari, G., Dollár, P., and Girshick, R. “Mask r-cnn”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2961–2969.

[How+17] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. “Mobilenets: Efficient convolutional neural networks for mobile vision applications”. In: *arXiv preprint arXiv:1704.04861* (2017).

[Joc+20] Jocher, G., Stoken, A., Borovec, J., NanoCode012, ChristopherSTAN, Changyu, L., Laughing, tkianai, Hogan, A., lorenzomammana, yxNONG, AlexWang1900, Diaconu, L., Marc, wanghaoyang0106, ml5ah, Doug, Ingham, F., Frederik, Guilhen, Hatovix, Poznanski, J., Fang, J., Yu, L., changyu98, Wang, M., Gupta, N., Akhtar, O., PetrDvoracek, and Rai, P. *ultralytics/yolov5*. Version v3.1. Oct. 2020. DOI: 10.5281/zenodo.4154370. URL: https://doi.org/10.5281/zenodo.4154370.

[Krä+22] Krämmer, A., Schöller, C., Gulati, D., Lakshminarasimhan, V., Kurz, F., Rosenbaum, D., Lenz, C., and Knoll, A. “Providentia-A Large-Scale Sensor System for the Assistance of Autonomous Vehicles and Its Evaluation”. In: *Journal of Field Robotics* (2022), pp. 1156–1176.

[Kuh55] Kuhn, H. W. “The Hungarian method for the assignment problem”. In: *Naval research logistics quarterly* 2.1-2 (1955), pp. 83–97.

[Lin+14] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. “Microsoft coco: Common objects in context”. In: *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer. 2014, pp. 740–755.

[Liu+21] Liu, H., Soto, R. A. R., Xiao, F., and Lee, Y. J. “Yolactedge: Real-time instance segmentation on the edge”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 9579–9585.

[Ma+22] Ma, X., Ouyang, W., Simonelli, A., and Ricci, E. “3d object detection from images for autonomous driving: a survey”. In: *arXiv preprint arXiv:2202.02980* (2022).

[Mas+21] Masi, S., Xu, P., Bonnifait, P., and Ieng, S.-S. “Augmented perception with cooperative roadside vision systems for autonomous driving in complex scenarios”. In: *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. IEEE. 2021, pp. 1140–1146.

[Pog+18] Poggenhans, F., Pauls, J.-H., Janosovits, J., Orf, S., Naumann, M., Kuhnt, F., and Mayr, M. “Lanelet2: A high-definition map framework for the future of automated driving”. In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2018, pp. 1672–1679.

[Qui+09] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A. Y., et al. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, p. 5.

[RAM21] Rezaei, M., Azarmi, M., and Mir, F. M. P. "Traffic-Net: 3D traffic monitoring using a single camera". In: *arXiv preprint arXiv:2109.09165* (2021).

[Sch+17] Schubert, E., Sander, J., Ester, M., Kriegel, H. P., and Xu, X. "DBSCAN revisited, revisited: why and how you should (still) use DBSCAN". In: *ACM Transactions on Database Systems (TODS)* 42.3 (2017), pp. 1–21.

[TAL16] Targ, S., Almeida, D., and Lyman, K. "Resnet in resnet: Generalizing residual architectures". In: *arXiv preprint arXiv:1603.08029* (2016).

[Van16] Vanholder, H. "Efficient inference with tensorrt". In: *GPU Technology Conference*. Vol. 1. 2016, p. 2.

[WBL22] Wang, C.-Y., Bochkovskiy, A., and Liao, H.-Y. M. "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors". In: *arXiv preprint arXiv:2207.02696* (2022).

[WB+95] Welch, G., Bishop, G., et al. "An introduction to the Kalman filter". In: (1995).

[Yu+22] Yu, G., Li, H., Wang, Y., Chen, P., and Zhou, B. "A Review on Cooperative Perception and Control Supported Infrastructure-Vehicle System". In: *Green Energy and Intelligent Transportation* (2022), p. 100023.

[Zha+17] Zhang, X., Xu, W., Dong, C., and Dolan, J. M. "Efficient L-shape fitting for vehicle detection using laser scanners". In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. 2017, pp. 54–59. DOI: 10.1109/IVS.2017.7995698.

List of Figures

List of Tables