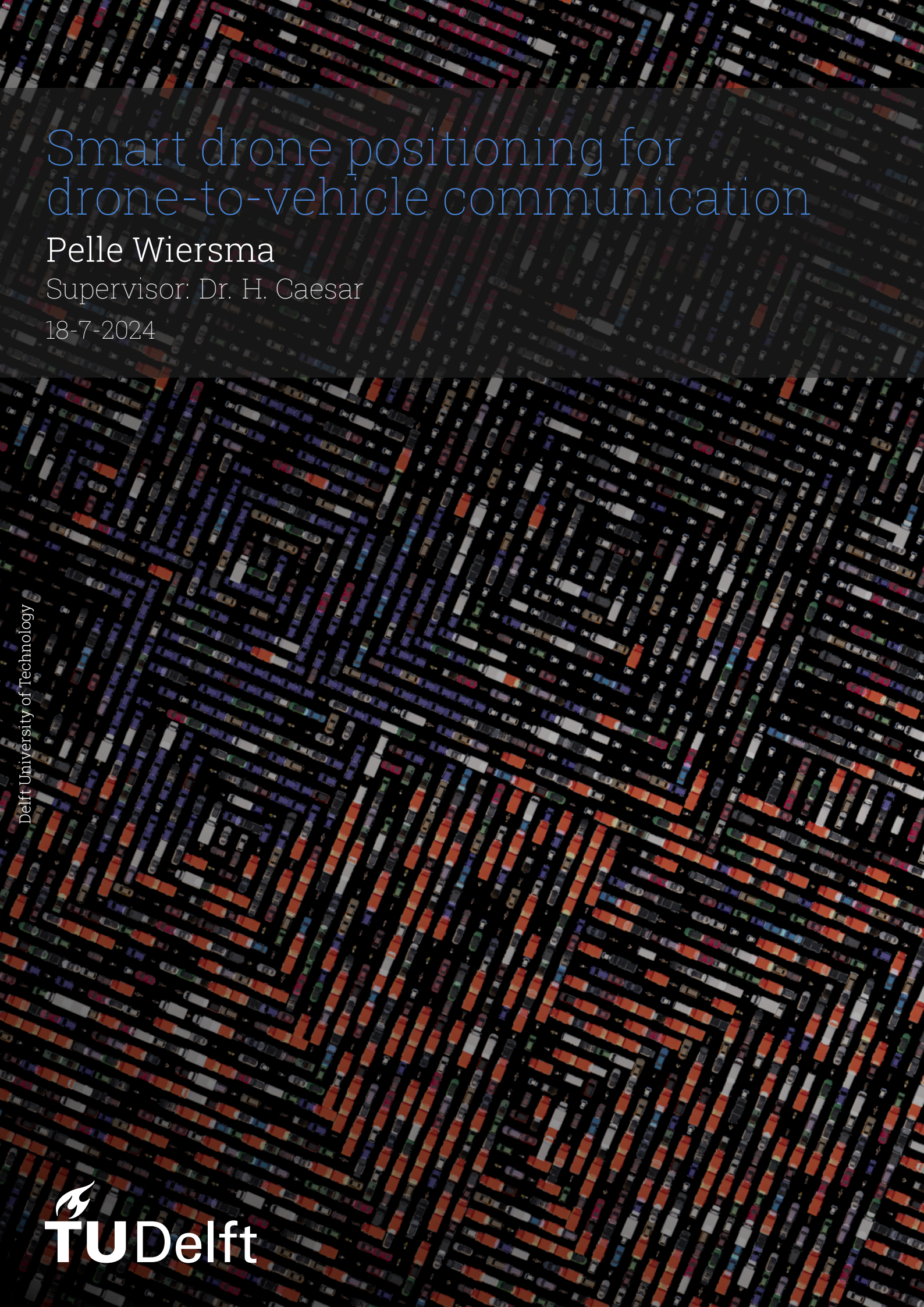
Smart drone positioning for drone-to-vehicle communication

Pelle Wiersma
Supervisor: Dr. H. Caesar

18-7-2024



Delft University of Technology

TU Delft

Smart drone positioning for drone-to-vehicle communication

by

Pelle Wiersma

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday August 1, 2024 at 12:30 AM.

Student number: 4749804
Project duration: January, 2023 – July, 2024
Thesis committee: Dr. H. Caesar, TU Delft, supervisor
Dr. C. Pek, TU Delft

Cover: Pelle Wiersma

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Smart drone positioning for drone-to-vehicle communication

P. J. Wiersma

***Abstract*—Currently, self-driving vehicles have trouble detecting partially and fully occluded objects such as pedestrians, vehicles, and static obstacles. It has been proven that a drone surveilling the area around the vehicle improves the vehicle's awareness of its surroundings. This work explores planning strategies for the drone and evaluates how much the strategies assist the vehicle. Previous work proposed a metric called PKL, which describes the awareness of a vehicle as a function of the detections of the vehicle using a planner model. Subtracting this metric calculated using detections done by the drone from the metric without these detections results in an awareness improvement metric. This metric was used to evaluate drone positions and therefore the drone planning strategies. It was found that aiming to hover directly above the autonomous vehicle improves the awareness of the vehicle measured by median PKL improvement by 1.8%. Using only the relative position of the autonomous vehicle to the drone, the velocity of the vehicle in x and y directions and the number of vehicles visible to the drone, an imitation learning-based strategy performed the best with 11.6% median PKL improvement. If the drone's planner knows the future position of the autonomous vehicle or the positions of all vehicles in the area, the median PKL improvement is 74.5% and 80.8%, respectively. Optimizing the trajectories using a genetic algorithm further improves the performance to 96.7%. From these numbers, we can conclude that aiming to stay directly above the vehicle does not benefit the autonomous vehicle the most. We show that intelligent drone trajectory planning strategies can be learned which improve the awareness of the autonomous vehicle and therefore the safety of the people in and around this vehicle.**

I. Introduction

A challenge for self-driving vehicles is to detect partially and fully occluded objects such as pedestrians, vehicles, and static obstacles. Current high-performance object detection algorithms miss approximately 1 out of every 10 partially occluded vehicles and 2 out of every 10 heavily occluded vehicles [1]. The problem is even more highlighted by the fact that Waymo claims that 2 out of the 8 incidents involving their self-driving vehicles were due to occlusion in intersection scenarios [2]. Smaller objects like pedestrians and cyclists are missed even more frequently. Fully occluded objects can be detected by tracking them while not being fully occluded, but this is not possible if objects were not detected beforehand.

A possible method to detect every relevant detection is to view the scene from a point where the objects are not occluded, such as from the air. The Horus-stationary [3] dataset was generated to test the effectiveness of a method where a drone hovers statically above the ego vehicle. It proved that more detections are done when such an additional view is used. This thesis aims to build on top of this current method by testing various strategies for controlling drones, such that the vehicles are aware of all relevant objects in their vicinity and help the vehicle look ahead. The different strategies that are evaluated vary from the baseline method (a single drone flying right above a single vehicle) to rule-based and RL-based planners.

Ideally, the drone planning strategies should not require knowledge of the environment and little information from the autonomous vehicle. This way, the planning strategy could be applied in every city and would not require a lot of bandwidth for the communication between the drone and the autonomous vehicle. These limitations challenge the models to exploit the information available to them and use it intelligently. It is a challenging task to train an RL model in a 3D real-time simulation environment due to the computational requirements of the simulation in combination with the large number of timesteps required to train models. A possible solution would be to use recordings/footage from the simulation. In this case, closed-loop training would be impossible, as the footage is fixed for fixed positions of the drone. To store every possible camera image for the positions the drone could be at would result in a massive dataset, which makes using it difficult.

Also, detecting vehicles which are not detected by the vehicle itself is not enough to improve the awareness of an autonomous vehicle, as these detections can range from: not relevant to the vehicle at all (e.g. parked vehicles, vehicles that are behind the autonomous vehicle and moving in the opposite direction) to crucial (e.g. vehicles at intersections that are occluded by other vehicles or buildings). For this reason, a metric should be chosen which is able to estimate the relevance of the detection to the autonomous vehicle. Previous work [4] proposed PKL, a metric capable of this task. Using this metric requires knowledge of the road network, which should be extracted from the simulation tool which is used. Finally, using a single drone for every autonomous vehicle does not seem economical when there are many autonomous vehicles. The visible area of drones would overlap and more complex behaviour would be possible which makes better use of the drones. This requires communication between the drones which could be achieved through a centralized unit. For this reason, a 2D simulation environment was created which is based on data created by a 3D real-time simulation environment. Using this simulation environment, models were trained and tested, resulting in the following main contributions:

- *Simulation tool:* A simulation tool was created that allows closed-loop training and evaluation of drone planning models to be done, which are used to increase awareness of autonomous vehicles resulting in safer vehicle trajectories. It uses an (almost) orthogonal view of a city. This view is cropped to the drone's field of vision allowing drone footage to be used for an infinite number of drone

All code is available in the GitHub repository: https://github.com/tudelft-iv-students/pellewiersma.git

positions while requiring little computational effort.

- *Informed model:* A novel drone trajectory planner is proposed which is optimized using a Genetic Algorithm (GA). The parameters of the GA's population are the coefficients of a polynomial which describes the trajectory. The GA's population is optimized using the mean PKL improvement as the fitness indicator. This approach requires knowledge of the positions of all vehicles in the scenario.
- *Uninformed model:* To create behaviour similar to the informed model, but without the knowledge of the positions of all vehicles, an Imitation Learning (IL) model was trained using the informed model's behaviour as demonstrations. Behaviour Cloning (BC) was applied resulting in unique behaviour using a small amount of required knowledge.

II. Related work

As mentioned in Section I, Horus-stationary [3] generated a dataset to test the effectiveness of a method where a drone hovers statically above the ego vehicle. This dataset has (on top of the sensor suite used in the nuScenes dataset [5]) a view from a drone, statically connected above a vehicle. This drone captures images viewing down towards the vehicle. This provides the autonomous vehicle with additional information about its surroundings. The Transfusion detector [6] was used to fuse the camera images recorded by the drone with the lidar point clouds recorded by the lidar sensor on the vehicle. The environment and physics were simulated using the Carla simulation tool [7]. The dataset is in the same format as nuScenes, which makes using the dataset easier, as tools that work on nuScenes (like the nuScenes devkit and Transfusion devkit) can be utilized directly on Horus-stationary. The main downside of this method is that there is no control done on the position of the drone. This is unrealistic as there are dynamics of the drone which introduce a delayed response, increased by the latency between the controller and the drone itself. Also, possibly a better position for the drone can be found which helps the vehicle more effectively. A different issue is that having a single drone for every single vehicle is expensive. It would economically be more interesting if a number of drones smaller than the number of vehicles could be used. This does however lead to a problem of how to strategically place these drones.

A different paper [8] suggests a future where a drone helps an autonomous vehicle by taking off from the vehicle, and moving towards areas that are occluded for the vehicle. The vehicle scans its surroundings, determines which areas are occluded using a 2D grid map, and calculates a trajectory for the drone to travel along. This method differs from the method presented in this work, as the calculation of the trajectory is done on the vehicle, and therefore requires much more computations from the vehicle. An upside of this method compared to the method presented in this work, is that the drone does not assume where the occluded areas are based on a limited observation space, but receives the locations known to be occluded. The presence of occluded areas is more likely in scenarios with a high density of traffic and buildings, such as cities. This means that the drone would be less active in scenarios with less traffic density and fewer buildings. Horus could provide a solution for this, as the drone(s) can change which vehicle they are helping based on whether they are in the city area, and therefore reduce the idle time and increase the effectiveness of the drone.

Unlike communication between vehicles and drones, many works research communication between vehicles (V2V) and between vehicles and infrastructure (V2I). V2Vnet [9] researches V2V by sending features of vehicles to one another in order to predict the trajectories of the vehicles in the area. Although they manage to prevent collisions using very little bandwidth, it is mentioned that their approach only works when multiple autonomous vehicles capable of communicating with each other are present in the same area. This is not required if the infrastructure communicates with autonomous vehicles. DAIR-V2X [10] presents a dataset consisting of recorded traffic scenarios at intersections. Installing sensors at intersections for V2I communication makes sense as intersections are areas where autonomous vehicles require help as mentioned in [2]. Additionally, by using statically installed sensors there are no problems regarding the safety of the drones themselves. However, modifying the infrastructure is a major investment which only makes sense if a significant number of vehicles can communicate with this infrastructure, which is unlikely in the near future. We present a method which would require no modifications to the infrastructure and could be deployed before V2I is an economical solution.

III. Methods

The varied nature of the planning strategies that will be trained and evaluated require a simulation environment capable of handling such variations. For this reason, a custom simulation environment was developed. This section covers the design of the tools required to experiment with the planning strategies. In Figure 1, a flowchart visualizes the process of simulating a traffic scenario. Section III-A explains the process of creating scenarios and storing them such that they can be used for training and evaluation of drone planning strategies. In Section III-B3, the perception module is explained, followed by a detailed description of all planning strategies which convert the drone's observation into an action (Section III-C). Next, Section III-D shows how the drone action translates into movement in the simulation. The final section, Section III-E, explains how the planning strategies are evaluated and what metric was used for evaluation.

A. Dataset

One of the main challenges for the dataset generation is that the drone should be able to generate an observation for every possible position the drone could be in to be able to control the drone in closed loop. If rendered images or video footage were used as a method to store scenarios, the large dataset size would make it difficult to use. This is why a format was created that only consists of the information required for top-down view applications.

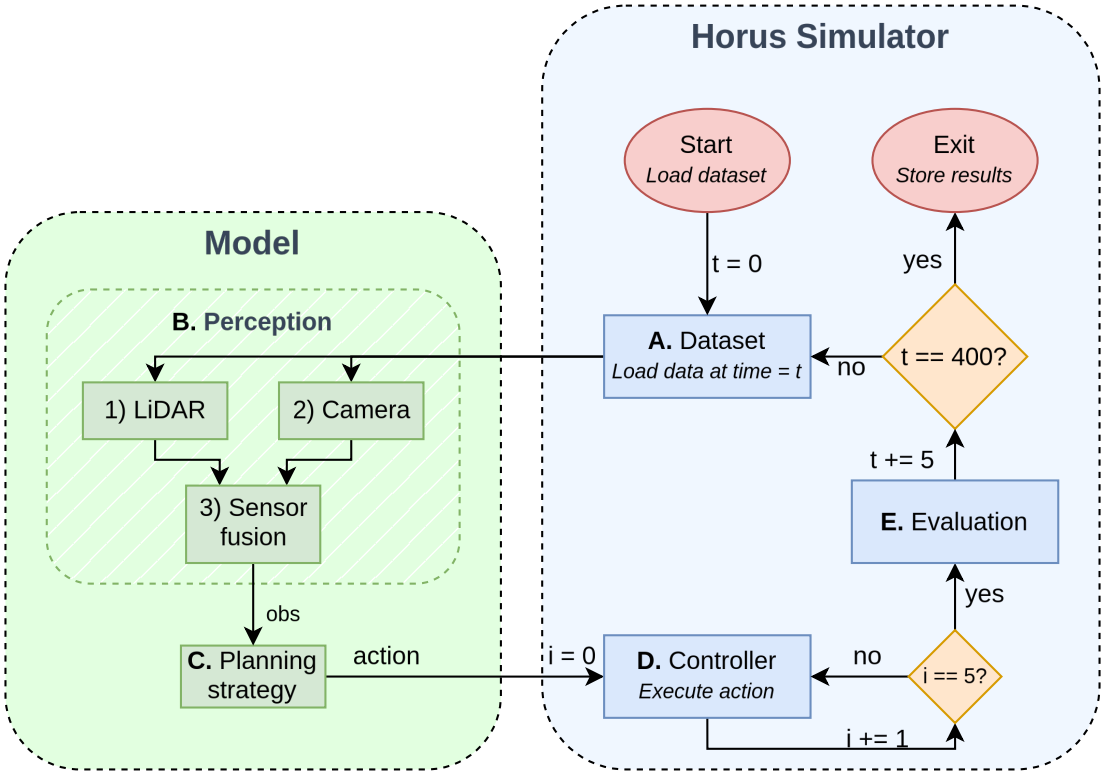


Fig. 1: This flowchart visualizes how a model is evaluated on a single scenario. The letters in the boxes represent the subsections in which they are discussed.

1) Map data: All static objects like buildings and roads look identical in every traffic scenario taking place in the same city (assuming lighting and weather conditions are ignored). It would therefore save a lot of data if the static objects could be stored only once and reused at every timestamp and every scenario. This is why a top-down HD render (39MP) of the image was made, along with an HD render of objects that could be in between the traffic and the drone such as tree foliage and bridges. The camera settings were chosen to create images which resemble an orthogonal image as closely as possible. This was done by moving the camera as high as the rendering distance of the Carla simulator allows (1.9km), and reducing the field of view to 9° to view only the city and not its surroundings. Furthermore, weather conditions were specified to avoid reflections and shadows. Also, sprites are stored for every vehicle type. This allows the vehicle sprites to be rendered on top of the city image and then covered with foliage and bridges. While our perception layer does not take foliage into account as mentioned in Section III-B3, this would be important as a realism factor if detections were done based on the images of the camera. A visualization of this top-down view is shown in Figure 2.

2) Scene data: The scenario-specific data is stored separately. It is divided into two parts: metadata and positional data:

Metadata: This information includes every vehicle's id, dimensions and vehicle model.

Positional data: This information includes the location of each vehicle, along with its orientation. Ids are used to match the scenario-specific data with the metadata. This information is stored for every timestamp.

3) Dataset size: If top-down images were used for every timestamp, the resulting size would be 16.2GB per ego-vehicle. For 6 scenes of 40s each, this would result in 97GB if detections were stored for every possible ego-vehicle. If only a single ego-vehicle would be used per scene, a dataset of 600 ego-vehicles would be 9.7TB.

Our format requires much less storage:

- Map data: 51.7MB
- Scene data: 37.7MB for 6 scenes of 40s each containing 600 possible ego-vehicles in total

If only a single ego-vehicle would be used per scene, a dataset of 600 ego-vehicles would be 3.8GB. This is only 3.9% of the dataset size if images were used.

B. Perception

Information about the scene at a certain timestamp is fed to the perception module. To prevent the performance of the perception module from interfering with the evaluation of the planning strategies, the drone's camera and lidar sensor are implemented as perfect sensors. How this was done is explained in this section.

1) LiDAR: To model a LiDAR sensor, 2D raytracing was used. 1 ray per degree is traced. Any vehicle reached by a ray within a range of 100m is classified as a detected vehicle by the ego-vehicle. To simulate buildings occluding sensor rays, the drivable area was used (How the drivable_area polygons

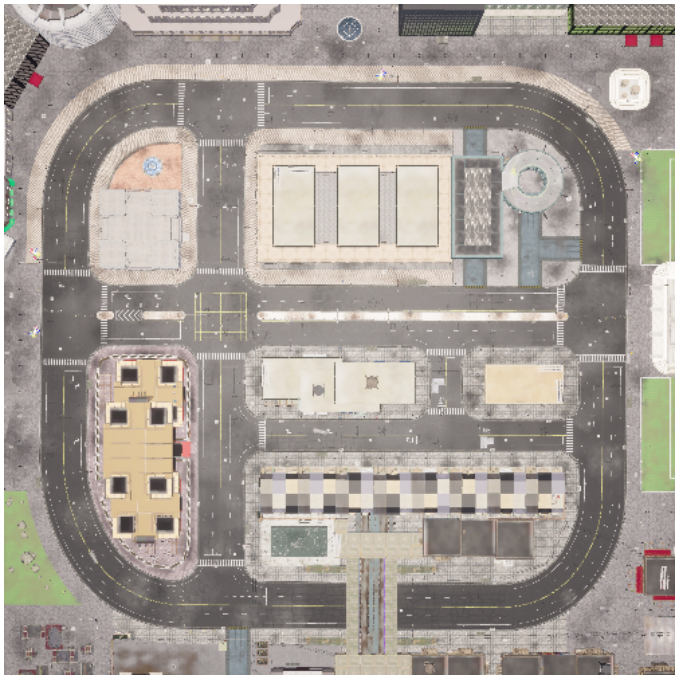


Fig. 2: The top-down view of the city. Note that some perspective is still present, as can be seen in the bottom-left corner. However, due to the sidewalk, this would not occlude any vehicles.

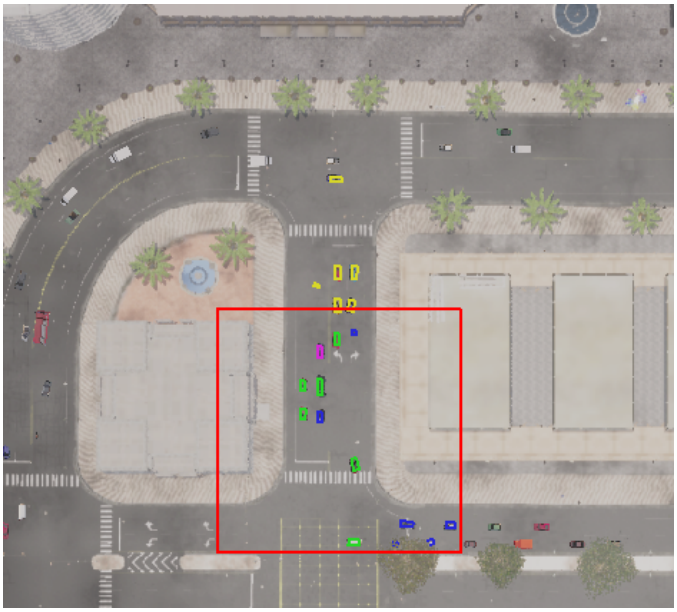


Fig. 3: A traffic scenario visualized in the environment. The red rectangle denotes the visible area of the drone's camera. The pink rectangle denotes which vehicle is the ego-vehicle, the yellow rectangles denote detections done by the ego-vehicle (LiDAR), the blue rectangles the detections done by the drone, and the green rectangles the vehicles detected by both the ego-vehicle and the drone.

were determined is explained in Appendix A). The rays will only be traced as long as they are on the drivable area.

2) Camera: Cameras usually detect vehicles using a trained model. To create the perfect detector, any vehicle inside the visible area of the drone is classified as a detected vehicle by the drone.

3) Sensor fusion: With vehicles detected by either the ego-vehicle, the drone, or both, they can now be fused. The fusion is done by taking the union of the detections done by the ego-vehicle (LiDAR) and the drone. Given that both the drone and ego-vehicle provide only detections which are part of the ground truth, no false positives or false negatives can be produced. Figure 3 shows what the detections look like.

C. Planning strategy

This section describes the strategies that were created and tested. The experiments cover a range of planning strategies both rule-based and Reinforcement Learning (RL) based. For all experiments, the drone's altitude was fixed to 50m, to reduce the number of variables in the results. The results are shown for three different traffic density levels, as this parameter has a large influence on the results.

1) Above ego (baseline) strategy: As a baseline, the drone always aims to hover directly above the ego-vehicle. This is done by passing the known location of the ego-vehicle to the low-level planner, which provides an action resulting in movement towards the ego-vehicle. This is visualized in Figure 6a.

2) Horus-stationary strategy: This approach is very similar to the baseline strategy but does not use a drone model to control the drone. Instead, the drone is fixed at a stationary position located above the ego-vehicle.

3) Flying ahead strategy: Next, the rule-based strategy "flying ahead" was tested. This strategy is tested because we think being aware of the vehicles at the future location of the ego-vehicle is valuable, especially because the ego-vehicle can not look around corners of buildings. Lead/lag time is used instead of lead/lag distance, as the distance to the future position the ego-vehicle will occupy is proportionally related to the velocity of the ego-vehicle. This means that when the ego-vehicle is on a highway, the drone will fly ahead, and when the ego-vehicle is stationary, the drone will hover above the vehicle. The best amount of time to lag or lead the ego-vehicle was determined by evaluating a set of 24 scenarios, and graphing the median % PKL improvement as a function of lag/lead time as shown in Figure 4. The best-performing lag/lead time was used for comparison with the other strategies and is visualized in Figure 6b.

4) RL-based strategy: Following this, a reinforcement-learning model was trained. It is based on the Proximal Policy Optimization (PPO [11]) method. This optimization method is applicable in a wide range of problems and is especially useful in online reinforcement learning applications where continuous action and observation spaces are required. Other algorithms like A3C [12] and DPPO [13] were considered.

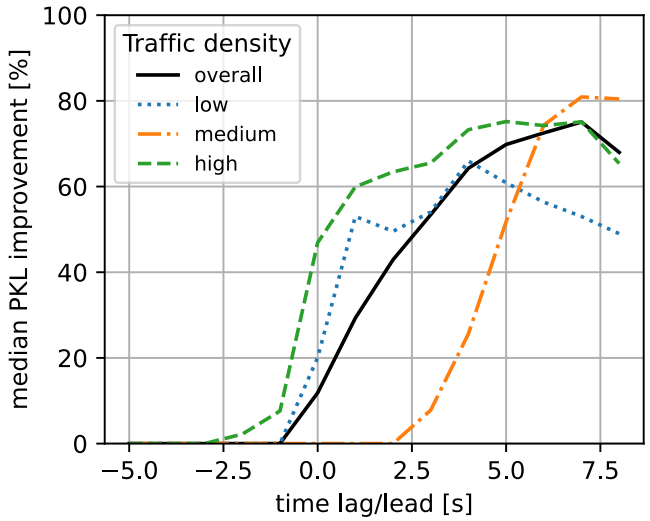


Fig. 4: The resulting PKL improvements (%) of the scenario with drone observation compared to no drone observation.

However, due to the complexity of running multiple agents at the same time and given that DPPO and A3C are not implemented in StableBaselines3 [14], the framework that was used for RL implementations, the more common PPO was chosen as a suitable algorithm. The reward function was designed to be a dense function in line with the guideline provided in [15]. Sparse rewards are a major problem in many reinforcement learning problems. This is also true for the problem present in this paper as "not detecting a vehicle" vs "detecting a vehicle" is a discrete step. This means that partial improvements (i.e. moving in the direction of the undetected vehicle) provide no partial reward if detections are part of the reward function. This would also be the case if PKL was used as a reward or part of the reward. This was tested and quickly discarded due to the slow training speed and the sparse rewards. Although methods have been developed which make a sparse function dense using guidance from an offline demonstration [16] [17], we provide partial rewards based on the distance to a good location (determined using data which is not accessible to the RL-model). The reward function is separated into two parts: A partial reward based on the distance from the ego-vehicle (promotes staying in the vicinity of the ego-vehicle), and the unseen vehicle component, which promotes seeing vehicles that are not visible to the ego-vehicle. The RL algorithm that was used has the following parameters:

Algorithm: **PPO** [11]
Action space:

- **target location rel. to drone** (x;y)

Observation space:

- **ego location rel. to drone** (x;y)
- **ego velocity** (vx;vy)
- **# vehicles in drone camera view**

Reward function:

- **Near ego component:**
  $f = 0.2 * (5 - |d_{\text{ego}}|)$
  This component is 1 when the drone is at the same location as the ego vehicle and linearly decreases with distance.
- **Unseen vehicle component:**
  $g = \sum_{i=1}^{n} e^{-0.2*|d_i|} - 0.1$
  For every unseen vehicle i, this component is 1 when the drone is at the same location as the unseen vehicle and becomes negative when the vehicle is more than 10m away. The final reward is equal to f when $g < 0$. Otherwise, the reward function is g.

5) Informed trajectory without GA: For every 5th timestep, the ideal location was determined. This was done by calculating the PKL value between the ego-vehicle's detections with and without every single unseen vehicle. The ideal location is defined as the location where the cumulative sum of PKL improvements (not %) is maximal. This means that for every location $f_{\text{PKL\_gain}} = \sum_{i=0}^{n} \Delta P_i$ is calculated, where n denotes the number of detected vehicles, ΔP_i the difference in PKL score between detecting this vehicle or not. The location where this equation is maximal was stored. This results in the location where the most relevant unseen vehicles are detected. To limit the number of helpful vehicles which improve the PKL by a negligible amount, a minimum PKL difference threshold was implemented at 3%. While these ideal locations give the expert agent information about what area to move towards, this does not consider the constraints of the drone. To optimize the trajectory using the constraints specified by the motion model, the Genetic Algorithm (GA) was used.

6) Informed trajectory with GA: Using information about the scenario outside of the relative position and the velocity of the ego-vehicle (i.e. the location of all vehicles), a theoretically perfect trajectory can be generated. As calculating the PKL value at every location or grid is expensive, some smart optimization method was used which is capable of handling functions which have many local optima. GA has proven to be a useful method of optimizing trajectories for UAVs [18] [19]. To speed up the training using GA, an informed baseline trajectory was created. This was achieved using the ideal locations mentioned in Section III-C5.
The genetic algorithm used here aims to maximize the median PKL value by evaluating and mutating trajectories. Every specimen has a genome which describes the trajectory. If the action for every 5th timestep would be part of the genome, this would result in a genome consisting of 160 variables ($400/5 = 80$ for both x and y). To reduce this number, the trajectory was parametrized by fitting a 15-degree polynomial to the trajectory. This results in 30 degrees of freedom for the GA. The mutation function modifies a number of coefficients with a probability. The details including parameters of the algorithm used are described in Appendix B.

7) IL-based strategy: The informed strategies use information about all vehicles in the scenario to construct the trajectories. We now propose a strategy which aims to imitate the behaviour executed by the drones following the optimized trajectories, without the optimization procedure and without knowledge of the entire scenario. This is done by using a method called Behaviour Cloning (BC) as implemented in [20]. It uses demonstrations known to be correct to create state-action pairs. To use BC in our machine learning problem, 100 scenarios from the optimized trajectories mentioned in Section III-C6 were fed to the PPO model using BC. The policy was then trained using these demonstrations. The specific parameters of the BC and PPO models are shown in App. C.

D. Controller

When a drone model is used, it produces an action, based on the observation. While the observation space varies from experiment to experiment, the action space is always a target waypoint. The waypoint is part of the action space rather than direct accelerations in x and y directions, as this would (in real-life scenarios) be a safer action space. Packet losses or latency could result in inaccurate and possibly dangerous movements if accelerations were used. At every timestamp the drone's x and y accelerations are calculated, along with its velocities and positions, to head towards this target waypoint (action). This is done using a Model Predictive Control (MPC) [21] implementation. For this, the following parameters are used:

- Model for movement in +x direction:

$$\begin{bmatrix}\dot{x}\\ \dot{v}\end{bmatrix} = \begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}\begin{bmatrix}x\\ v\end{bmatrix} + \begin{bmatrix}0\\ c/m\end{bmatrix} u$$

$$y = \begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}x\\ v\end{bmatrix}$$

  To reduce the complexity of the model, the term c/m was set to 1, making the input action u equal to the acceleration of the drone. An identical state-space model exists for movement in the y direction.
- Control horizon: 10 timesteps (1s)
- Acceleration bounds: [-5, +5]
- Optimization tolerance: 0.1m
- Update frequency: 2Hz
  To speed up simulation, the control action is calculated every 5 timesteps, resulting in a 2Hz update frequency. This also forces the model to commit to an action, as actions are executed for longer and will therefore be penalized more if they are not good actions.

E. Evaluation

Every 5 timesteps, the current position of the drone is evaluated. This evaluation is used to compare the different planning strategies. The evaluation metric uses the planner-centric metric PKL [4]. This metric compares a vehicle trajectory plan based on ground truth detections with a plan based on observed detections by calculating the KL divergence between the two. If we calculate this metric on the observed detections done by the lidar sensor only and also on the observed detections done by the drone and the lidar sensor together, we can assess the difference in PKL to find the PKL improvement. The PKL improvement values of all vehicles in the area will be calculated once every 5 timesteps when improvement is possible (PKL improvement not equal to 0), such that at the end of a scene lasting 40s, a list of PKL improvements is stored. By evaluating the performance on a set of 96 ego vehicles, we can take the median of all PKL improvement scores to find the final score. The median is used instead of the mean, as the median closer represents the performance when there are a significant number of outliers. [22] To calculate PKL on data which is not part of the nuScenes dataset, the Carla map was converted to the nuScenes format. How this was done is described in Appendix A.

IV. Results

This section shows the performance of the tested strategies. First, some qualitative results will be shown. These give an impression of the behaviour of the strategies using example scenarios. Then quantitative results describe the performance of the strategies across a test set of scenarios.

A. Qualitative results

To show the variety in behaviour between the strategies, Figure 5 visualizes the trajectories of the drone for a sample scenario. The flying ahead strategy does not directly follow the path of the vehicle, even though it is aware of its future position. This is due to overshooting of the low-level controller. At the end of the trajectory, the overshooting oscillations reduce and it stops at the same location as the baseline strategy. The action graphs shown in Figure 6 help us understand the differences in behaviour between the different strategies. We can only construct such graphs for strategies where the strategy uses the location relative to the ego-vehicle to determine an action. The optimized trajectories and Horus-stationary do not have such a graph as these do not have policies.

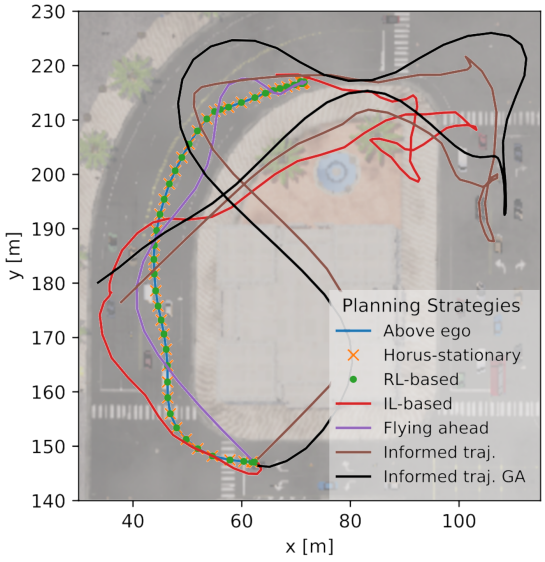
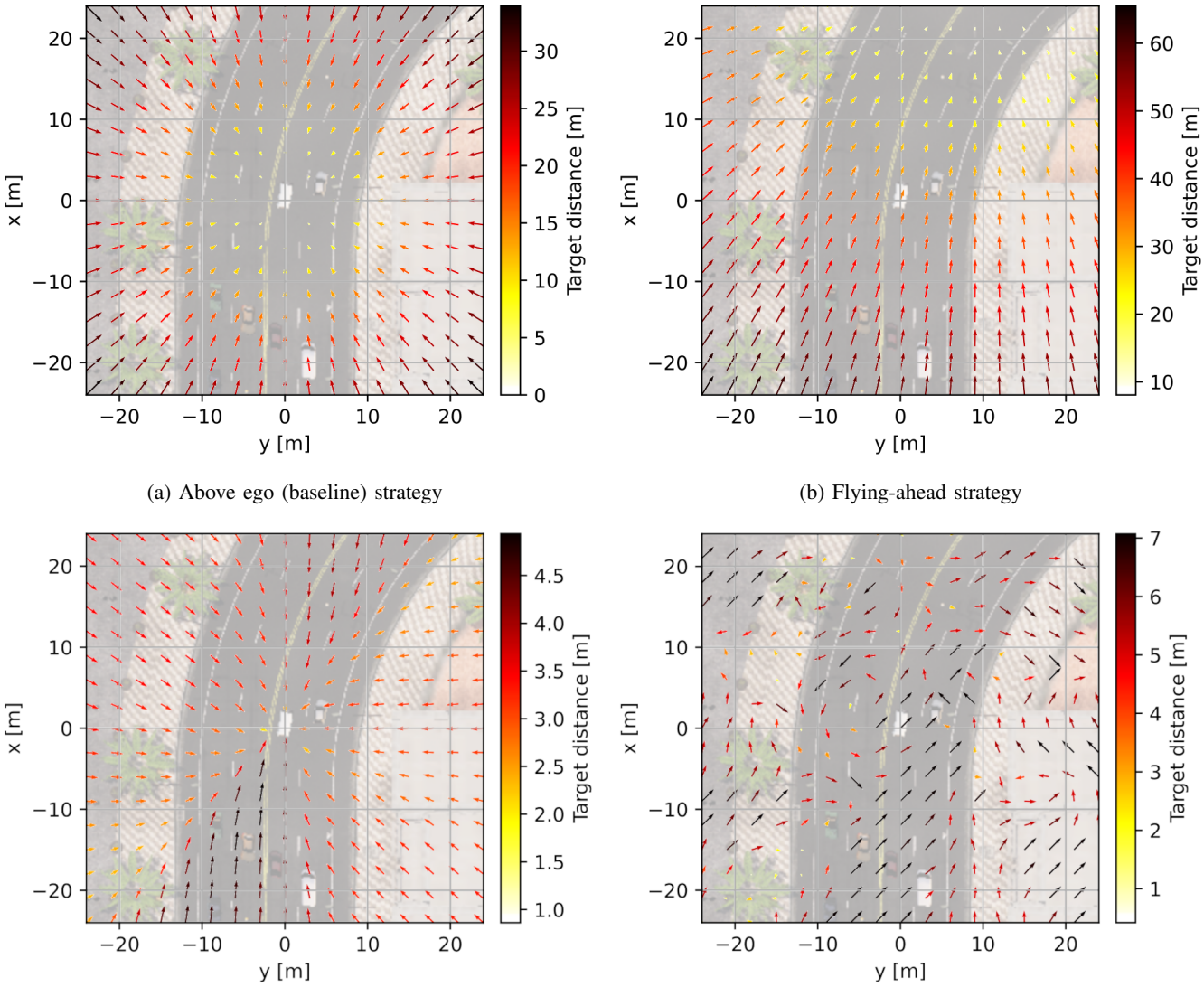


Fig. 5: Trajectories for a sample scenario.



(a) Above ego (baseline) strategy

(b) Flying-ahead strategy

(c) RL-based strategy

(d) IL-based strategy

Fig. 6: Action graphs for the drone. The x and y coordinates denote the position of the drone relative to the ego-vehicle. The magnitude of the vectors denotes the distance to the target position, which is proportional to the acceleration.

B. Quantative results

Table I shows the performance for each strategy. The ”overall” column is the most important, as this indicates the strategies' performance across a wide range of traffic densities. Additionally, results are shown for 3 different traffic densities separately, as the performance of some strategies is highly dependent on this variable. The informed column denotes whether or not the strategies use knowledge which is not the relative position of the drone to the ego-vehicle, the velocity of the ego-vehicle and the number of vehicles visible by the drone. In the case of the flying ahead strategy, this information is the future location of the ego-vehicle. In the case of the informed trajectories, all vehicles' locations in the scenario are used. The Horus-stationary row does not have the ”Informed” checkmark as it does not require additional knowledge. However, it does not use a motion model and is therefore not a realistic strategy.

			Traffic density		
	Informed	Overall	Low	Medium	High
Above ego (baseline)	✗	7.8	81.8	20.2	0.0
Horus-stationary	✗	11.1	81.8	22.6	0.0
RL-based strategy	✗	11.2	**84.5**	23.2	0.0
IL-based strategy	✗	**11.6**	0.0	**63.4**	0.0
Flying ahead strategy	✓	74.5	80.6	76.5	70.4
Informed trajectory	✓	80.8	89.5	81.6	76.4
Informed trajectory GA	✓	**96.7**	**99.5**	**88.8**	**92.4**

TABLE I: Median % PKL improvement for the strategies

V. Discussion

In this paper, various drone trajectory planning strategies were tested and trained. Their performance was evaluated and compared using the median % PKL improvement and yielded varying results.

A. Uninformed knowledge strategies

1) Above ego (baseline) strategy: The "Above ego" (baseline) method has a performance which is highly dependent on the traffic density. From Table I, it is clear that when there is a high traffic density, having the drone above the vehicle does not help the vehicle's awareness, while in a low traffic density scenario, the vehicle does benefit. This is likely due to the number of vehicles that are present outside of the ego-vehicle's visibility. When the traffic density is high, the number of undetected vehicles is larger, especially when the drone is above the vehicle, as their detections overlap for the most part.

2) Horus-stationary: The evaluation of Horus-stationary [3] shows that the drone dynamics we implemented, have little impact on the results compared to having the drone statically placed above the vehicle. The slight difference could be explained by the fact that by following the ego-vehicle, the drone will always slightly lag behind the vehicle.

3) RL-based strategy: When a RL model is trained using the properties as mentioned in Section III-C4, an interesting result is found: The performance of the RL model is very similar to the performance of the baseline strategy. It only slightly outperforms the baseline as well as Horus-stationary. The action graph Figure 6c also shows us that the behaviour is very similar to the baseline's action graph. This graph is identical (relative to the ego-vehicle) for each direction, irrespective of the number of vehicles in the visible area of the drone. This shows that the part of the reward function responsible for movement towards undetected vehicles did not influence the behaviour of the drone.
Tests show that the magnitude of the "unseen vehicle" component of the reward function is similar to that of the "near ego" component which is desirable as no component makes the other component insignificant. Another positive property of the reward component is that it provides partial rewards for desirable behaviour. Also, if all vehicles in the scene were considered, there would be no discrete steps in the reward function. However, to reduce computational effort, only vehicles within 100m of the ego-vehicle are considered. This does result in discrete steps when vehicles enter or leave this 100m zone. A different issue with this reward function is that every undetected vehicle is treated equally. As mentioned, this was done intentionally to create an "almost" continuous reward function. This does mean that for the drone, flying behind the ego-vehicle would result in a similar reward as flying ahead of the ego-vehicle given the vehicles in the scene are distributed uniformly. By not indicating a preferred position relative to the ego-vehicle based on the "unseen vehicle" component, on average, every position relative to the ego-vehicle is equally good. This is an explanation for the fact that on average the drone aims to hover directly above the ego-vehicle, just like the baseline strategy. Lastly, an issue with this reward component is that the magnitude is very dependent on the number of vehicles in the vicinity of the ego-vehicle. A scenario where there are a high number of vehicles in the 100m range lowers the reward (as many will be more than 10m from the drone). Obviously, the model should not be punished by this fact. Scaling the reward based on the number of vehicles could solve this issue. The questions that remain are then: Why is the action graph different from the baseline's action graph? And why does this difference improve the performance? The main difference in the action graphs Figure 6c and Figure 6a is the small target distances (vector lengths) resulting in small accelerations anywhere except directly behind the ego-vehicle. This could be explained by the fact that the ego-vehicle in these scenarios never moves backwards. This means that if the drone flies ahead, left or right of the ego-vehicle it is either beneficial to stay where you are (in case the ego-vehicle turns left, right or keeps moving forward), or to move towards the ego-vehicle (in case the ego-vehicle stops moving or does not turn), resulting in behaviour that moves (on average) a little bit towards the ego-vehicle. If the drone is flying behind the ego-vehicle, it is always beneficial to move towards the ego-vehicle (so forward) irrespective of what the ego-vehicle will do, as the vehicle will never reverse. As this action graph displays the sole difference between the baseline strategy and the RL-model strategy, this must also be the cause of the better performance. The model seems to have learned that the ego-vehicle will never reverse, therefore adjusting its policy to always accelerate towards the ego-vehicle, mostly when behind the vehicle. As the performance is also better than that of the Horus-stationary strategy, the RL-based strategy does not follow the vehicle better than the baseline, as that would have resulted in performance between the baseline and Horus-stationary. A possible explanation for this is that the large acceleration right behind the ego-vehicle results in slightly overshooting the ego-vehicle's position, therefore leading the ego-vehicle by a small amount. Due to the small difference in performance, this was not analysed further.

4) IL-based strategy: The IL-based strategy learned its trajectories by imitating the informed trajectories which were optimized using GA. Figure 6d seems to suggest that there is no clear direction the drone moves towards. This could be due to too little training data/overfitting but is also likely to be the cause of why this strategy outperforms the baseline as well as the RL-follower in medium traffic density scenarios. The drone never stops moving but is always in the vicinity of the ego-vehicle (as can be seen by the arrows moving inwards or equidistant far from the ego-vehicle in Figure 6d). By analysing the informed trajectories we found that there is no clear strategy for the drone to follow. Sometimes it is beneficial to move very far ahead of the ego-vehicle, while at other times it is beneficial to move very far behind the ego-vehicle. This makes it difficult to imitate behaviour and generalize this to a policy which would work in any scenario. The results of the IL-based strategy shown in Table I indicate that the strategy performs the best when applied in medium traffic density scenarios. From Figure 6d we know that this strategy does not aim to hover directly above the ego-vehicle, but instead always moves the drone in the vicinity of the ego-vehicle. This could be the reason why the IL-based strategy performs better in medium traffic scenarios compared to the low and high traffic density scenarios, as this behaviour is most beneficial in the medium traffic density scenarios. We know this as Figure 4 shows that the further ahead of the ego-vehicle the drone flies, the higher the median % PKL improvement.

Regarding the algorithm that was used for IL, a basic behavioural cloning technique was used which directly learns a policy from demonstrated state-action pairs. As mentioned in the documentation of [20], this approach often generalizes poorly and does not often recover easily if an error is made. A more advanced method such as Generative Adverserial Imitation Learning [23], could improve the performance due to its robustness.

B. Informed knowledge strategies

1) Flying ahead strategy: Figure 4 shows that it is most beneficial to fly ahead 7s. For completeness, also lagging behind the vehicle was evaluated. This is clearly less beneficial, as the vehicles around the future position of the ego-vehicle are more relevant than those that are already past. We hypothesised that the lower the traffic density, the further the drone should be ahead to detect the most relevant vehicles, as fewer vehicles around the ego-vehicle would mean that fewer vehicles can occlude others. In Figure 4 we notice that this is the case for medium and high traffic densities, but surprisingly not for the low traffic density. Analysis shows that this has to do with the distributions in median % PKL improvement. Figure 7 shows that a much broader range of median % PKL improvements is present in low traffic density scenarios compared to the median % PKL improvements in medium and high traffic density scenarios as shown in Figure 8. The broad range of the $P_{45} - P_{55}$ percentiles makes the peak at 4s less relevant. What can be concluded from this Figure 7 is that from 1s ahead and more the ego-vehicle benefits from the drone's detections.

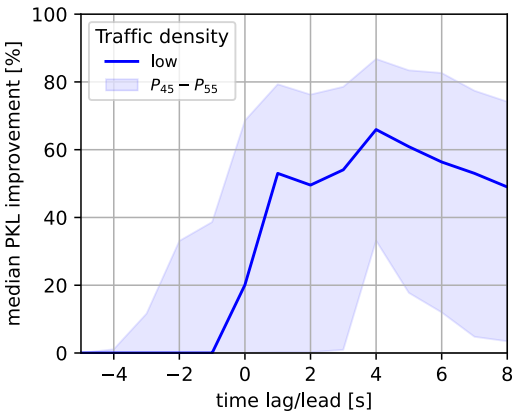


Fig. 7: The resulting PKL improvements (%) with percentiles for the low traffic density scenarios.

Looking at the performance of flying ahead strategy in Table I, we notice it exceeds the performance of staying above the vehicle for medium to high traffic densities. This can be explained by the fact that more unseen vehicles are seen when not hovering above the ego-vehicle at higher traffic densities. To fly ahead of the ego-vehicle does however require additional knowledge about the future position of the ego-vehicle. In real life this could be achieved by predicting the future location of the ego-vehicles trajectory as done in [24].

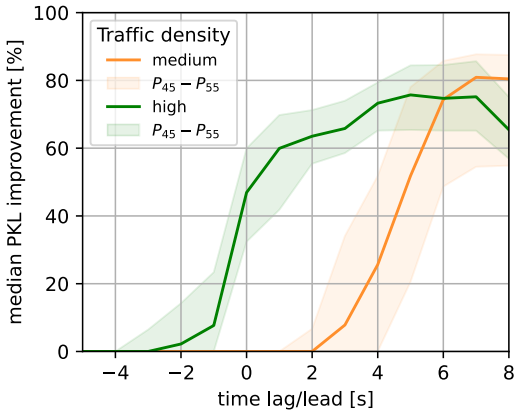


Fig. 8: The resulting PKL improvements (%) with percentiles for the medium and high traffic density scenarios.

2) Informed trajectories: The highest performance is achieved by the Informed trajectories. These require knowledge about the current location of all vehicles in the scenario. The informed trajectories without GA optimization already outperform all previous other strategies. The difference between the performance of the informed trajectories with and without GA is likely due to the aggressive movements required for optimal performance. When moving towards the ideal location at every timestamp as done in the informed trajectories without GA, the drone stays in between the optimal locations and reaches them less often. The GA then mutated the trajectories such that these optimal locations are reached more often. An example displaying this behaviour is shown in Figure 9.

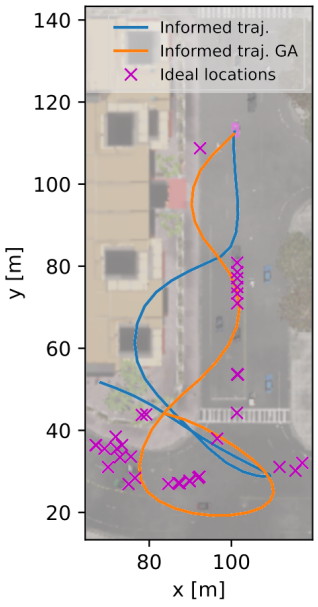


Fig. 9: A sample trajectory shown with and without GA optimization. The non-optimized trajectory tends to move in between ideal locations, while the optimized trajectory gets closer to the ideal locations.

3) Genetic Algorithm: Although the GA significantly improved the performance compared to the informed trajectories without GA, a number of improvements could be made to enhance the performance of the GA implementation:

- A total population of 10 specimens with 3 elites makes exploration and exploitation a very slow process. This was left low intentionally as evaluation of these specimens is a very time-consuming process, requiring ± 2 minutes per specimen. Speeding up the evaluation process would allow for more specimens and therefore more diversity in the population.
- Currently, the mutation and crossover probabilities are kept constant throughout all generations. To promote exploration in the early stages of evolution and promote exploitation in the later stages of evolution, the mutation and cross-over probabilities could be reduced as the generation number increases.
- The parameters in the genome are currently mutated by adding or subtracting a randomly generated number within bounds. As the order of magnitudes of the parameters has a wide range (ranging from 1 to as much as 10^4) adding or subtracting a value irrespective of the current value, does not result in an equal amount of mutation for every parameter. Multiplication with a factor $x^a + b$, where b is a randomly generated number able to change the sign of the parameter, x a set fixed number and a a randomly generated number ranging from $-\alpha$ to α, would result in a mutation proportional to the current value of the parameter with an added constant.

C. Future work

While all results in this work aim to find the optimal drone planning strategy, the performance in real-life scenarios will differ. A way of closing this gap would be to evaluate strategies in Carla itself, where perspective, shadows and other elements are taken into account which improve realism. Also, the degrees of freedom of the drone have been limited to enhance the learning and evaluation speed. Adding the heading of the drone as a degree of freedom (or camera angle) would allow for higher possible performance at the cost of simulation speed. Furthermore, in this work the assumption is made that 2D raytracing provides an accurate model for an autonomous vehicle's lidar sensor, and the camera's detections are 100% accurate. This is not the case in real life, and adding a perception module with realistic detectors and a fusion model such as TransFusion [6] would enhance the significance of the results. As the Carla simulation tool already models LiDAR sensors, which are used in Horus-stationary [3], the detector could be applied directly to the pointcloud generated by the Carla simulations tool. In addition to these improvements, there are a number of factors which further enhance the realism and challenge the strategies. These improvements include:

- Sensor noise
- Communication latency
- Weather conditions

Conditions such as wind gusts, or rain/fog reduce visibility in case a realistic perception layer is implemented. The economic feasibility of this technology can be increased when drones help more than a single vehicle at a time. To accommodate multiple drones, the algorithms used require modifications. A task allocation algorithm such as the Hungarian algorithm [25] could be implemented. This way the model could utilize the same action space and observation space. To take interaction into account in a more intelligent manner, a multidimensional action space and observation space could be used, where the action space and observation space is an array of the action and observation spaces of a single drone.

VI. Conclusion

We present a simulation environment which is capable of evaluating drone planning strategies using a metric specifically designed to evaluate autonomous vehicles' awareness. It uses traffic scenarios simulated in the Carla simulator stored in an efficient format. An almost orthogonal top-down image of the city is stored. With this information, entire traffic scenarios can be rendered. The drone's footage can be artificially generated by cropping the topdown view (with the traffic rendered on top) to the area where the drone is located. This makes closed-loop training of models possible while keeping the storage requirements and computational effort low. A range of strategies were developed and tested in this simulation environment. The performance of these strategies was compared to the baseline method (i.e. aiming to hover directly above the ego-vehicle). While the baseline method provided useful data to the ego-vehicle as proven in [3], we prove that superior strategies exist, which require little data to function. If extra data is available to the model, such as the future position of the ego-vehicle or the location of all vehicles in the area, the awareness of the autonomous vehicle can be improved even more.

References

[1] S. Gilroy, E. Jones, and M. Glavin, "Overcoming occlusion in the automotive environment. a review," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 1, pp. 23–35, 2021.

[2] M.-Q. Dao, J. S. Berrio, V. Fremont, M. Shan, E. Hery, and S. Worrall, "Practical Collaborative Perception: A Framework for Asynchronous and Multi-Agent 3D Object Detection," Sep. 2023, arXiv:2307.01462 [cs]. [Online]. Available: http://arxiv.org/abs/2307.01462

[3] M. Malak, "Horus: Drone assisted autonomous vehicles," Delft, June 2023.

[4] J. Philion, A. Kar, and S. Fidler, "Learning to evaluate perception models using planner-centric metrics," 2020.

[5] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 11 621–11 631.

[6] X. Bai, Z. Hu, X. Zhu, Q. Huang, Y. Chen, H. Fu, and C.-L. Tai, "Transfusion: Robust lidar-camera fusion for 3d object detection with transformers," 2022.

[7] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[8] A. Wallar, B. Araki, R. Chang, J. Alonso-Mora, and D. Rus, "Foresight: Remote sensing for autonomous vehicles using a small unmanned aerial vehicle," in *Field and Service Robotics: Results of the 11th International Conference*. Springer, 2018, pp. 591–604.

[9] T.-H. Wang, S. Manivasagam, M. Liang, B. Yang, W. Zeng, and R. Urtasun, "V2vnet: Vehicle-to-vehicle communication for joint perception and prediction," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*. Springer, 2020, pp. 605–621.

[10] H. Yu, Y. Luo, M. Shu, Y. Huo, Z. Yang, Y. Shi, Z. Guo, H. Li, X. Hu, J. Yuan *et al.*, "Dair-v2x: A large-scale dataset for vehicle-infrastructure cooperative 3d object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 21 361–21 370.

[11] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," Aug. 2017, arXiv:1707.06347 [cs]. [Online]. Available: http://arxiv.org/abs/1707.06347

[12] Volodymyr Mnih, V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," pp. 1928–1937, Jun. 2016.

[13] N. Heess, D. Tb, Dhruva Tb, S. Sriram, Jay Lemmon, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Ziyu Wang, Z. Wang, S. M. A. Eslami, M. Riedmiller, and D. Silver, "Emergence of Locomotion Behaviours in Rich Environments," *arXiv: Artificial Intelligence*, Jul. 2017.

[14] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[15] H. Sowerby, Z. Zhou, and M. L. Littman, "Designing rewards for fast learning," *arXiv preprint arXiv:2205.15400*, 2022.

[16] D. Rengarajan, G. Vaidya, A. Sarvesh, D. Kalathil, and S. Shakkottai, "Reinforcement learning with sparse rewards using guidance from offline demonstration," *arXiv preprint arXiv:2202.04628*, 2022.

[17] J. Hare, "Dealing with sparse rewards in reinforcement learning," *arXiv preprint arXiv:1910.09281*, 2019.

[18] P. A. Jimenez, B. Shirinzadeh, A. Nicholson, and G. Alici, "Optimal area covering using genetic algorithms," in *2007 IEEE/ASME international conference on advanced intelligent mechatronics*, 2007, pp. 1–5.

[19] H. Teng, I. Ahmad, A. Msm, and K. Chang, "3D Optimal Surveillance Trajectory Planning for Multiple UAVs by Using Particle Swarm Optimization With Surveillance Area Priority," *IEEE Access*, vol. 8, pp. 86 316–86 327, 2020, conference Name: IEEE Access. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9086499

[20] A. Gleave, M. Taufeeque, J. Rocamonde, E. Jenner, S. H. Wang, S. Toyer, M. Ernestus, N. Belrose, S. Emmons, and S. Russell, "imitation: Clean imitation learning implementations," arXiv:2211.11972v1 [cs.LG], 2022. [Online]. Available: https://arxiv.org/abs/2211.11972

[21] E. F. Camacho and C. Bordons, *Model Predictive Controllers*. Springer London, 2007, pp. 13–30. [Online]. Available: http://dx.doi.org/10.1007/978-0-85729-398-5_2

[22] Y. Guo, H. Caesar, O. Beijbom, J. Philion, and S. Fidler, "The efficacy of neural planning metrics: A meta-analysis of pkl on nuscenes," *arXiv preprint arXiv:2010.09350*, 2020.

[23] J. Ho and S. Ermon, "Generative adversarial imitation learning," *Advances in neural information processing systems*, vol. 29, 2016.

[24] N. Deo, E. Wolff, and O. Beijbom, "Multimodal trajectory prediction conditioned on lane-graph traversals," in *Proceedings of the 5th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, A. Faust, D. Hsu, and G. Neumann, Eds., vol. 164. PMLR, 08–11 Nov 2022, pp. 203–212. [Online]. Available: https://proceedings.mlr.press/v164/deo22a.html

[25] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800020109

[26] H. Blayney, "pyxodr," https://github.com/driskai/pyxodr, 2023.

APPENDIX A
CREATING A PKL-COMPATIBLE MAP FROM THE OVERDRIVE FORMAT

Carla simulator allows users to save the map data in the OverDrive format. This format is not compatible with the PKL library, which requires the map to be in the nuScenes format. The steps to convert the map format from OverDrive to nuScenes are described in this appendix.

1) The map data is loaded into Python using the pyxodr library [26].
2) **Drivable area:**
   a) All lane sections are extracted, of which the nodes are first rounded to an 80cm grid. This is done to eliminate imperfect connections between lane sections which would otherwise result in slits/cuts into the lanes which are not correct (e.g. shown in Figure 10).
   b) The outer boundaries of these lane sections are added to a list of polygons.
   c) All lane section polygons part of a junction are merged by calculating the convex hull of the polygons, to eliminate holes/imperfections which would otherwise be known as not drivable area.
   d) The union of the lane section polygons are merged with the junction polygons by calculating the union. The resulting polygon is known as the drivable area polygon.
3) **Road segments** (OverDrive term: lane section):
   a) Each road segment can be directly extracted from the map data.
   b) The road segment nodes are rounded to 10cm to eliminate imperfections such that adjacent road segments connect correctly.
   c) The resulting road segments are converted to polygons.
4) **Lane:**
   a) The lane_reference_line and boundary_line are extracted from the OverDrive data as lists of coordinates.
   b) The coordinates of the lane_reference_line are reversed to get a clockwise list of points around the lane.
   c) The points of the lane_reference_line and boundary_line are merged to form a polygon.
5) **Lane divider:**
   a) All lanes which have a lane to the left of them (heading in the same direction!) are selected.
   b) The lane_reference_line of these lanes are stored as lane dividers.
6) **Road divider:**
   a) All lanes which have a lane to the left of them heading in the *opposite* direction are selected.
   b) The lane_reference_line of these lanes are stored as road dividers.

Now all required items have been extracted or fabricated, they are stored in a dictionary according to the nuScenes format:

1) **Node:** For every coordinate of every polygon and line, a node dict is created (if this coordinate does not yet exist as a node), which consists of the x and y coordinate, as well as a randomly generated token consisting of 128 bits. These 128 bits are stored as in hex format.
   - *token:* A randomly generated id.
   - *x:* x coordinate of the node.
   - *y:* y coordinate of the node.
2) **Line:**
   - *token:* A randomly generated id.
   - *node_tokens:* A list of tokens which cover all nodes part of the line.
3) **Polygon:** For every polygon, a polygon dict is created consisting of the following items:
   - *token:* A randomly generated id.
   - *exterior_node_tokens:* All node tokens part of the exterior of the polygon
   - *holes:* The tokens of the hole polygons in the polygon. In the map we use, this part is only applicable for the drivable_area polygon.
4) **Road_segment:**
   - *token:* A randomly generated id.
   - *polygon_token:* Token of the road segments' polygon.
   - *is_intersection:* Whether or not the is_junction flag is True in the OpenDrive format for that road segment.
   - drivable_area_token: In the case of this map, there is a single drivable area polygon. Its token is entered here.
5) **Lane:**
   - *token:* A randomly generated id.
   - *polygon_token:* The token denoting the lanes' boundary polygon.
   - *lane_type:* Always "CAR" for this map.
   - *from_edge_line_token:* The first node of the lanes' lane_reference_line and boundary_line form a line object. The token of this line is stored here.
   - *to_edge_line_token:* The last node of the lanes' lane_reference_line and boundary_line form a line object. The token of this line is stored here.
   - left_lane_divider_segments: Not required for PKL and thus left empty.
   - left_lane_divider_segments: Not required for PKL and thus left empty.
6) **Lane_divider:**
   - *token:* A randomly generated id.
   - *line_token:* The token denoting the lane_dividers' line object.
   - *lane_divider_segments:* A list of items consisting of:
     - *node_token:* The node id of the node part of the lane divider.

– *segment_type:* The type of line, can be ”DOUBLE_DASHED_WHITE” or ”NIL”.

7) **Road_divider:**

- *token:* A randomly generated id.
- *line_token:* The token denoting the road_dividers' line object.
- *road_segment_token:* The token of the road segment this divider is a part of.

These items are stored in a JSON file. A visualization of the map using the nuScenes library is shown in Figure 11.

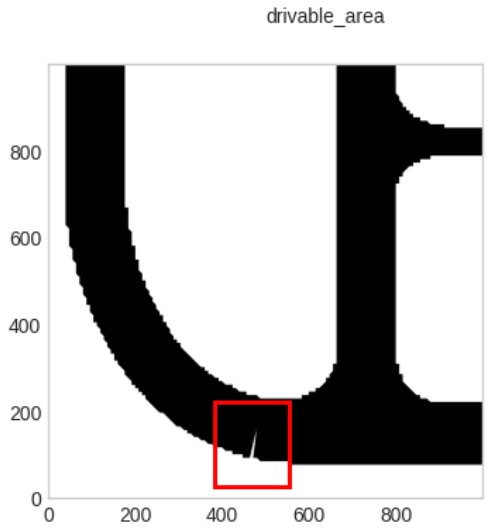


Fig. 10: Imperfections in the map data result in slits/cuts into the lanes which should be eliminated.

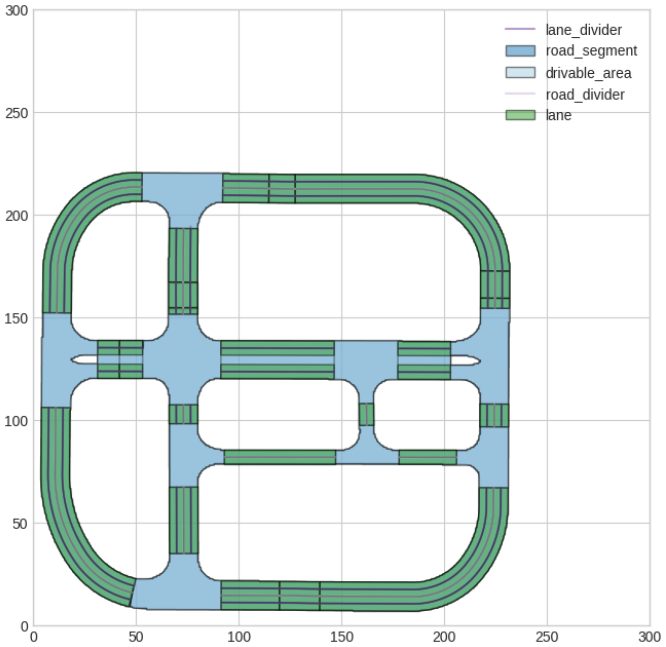


Fig. 11: The converted map is visualized using the nuScenes map expansion library.

APPENDIX B
OPTIMIZING DRONE TRAJECTORIES USING GENETIC ALGORITHMS

To optimize the drone's trajectories, a genetic algorithm was used. This is a method which is useful when there are many local optima in a function/process and especially if the evaluation of a function/process is time-consuming. As the performance of the function is quantified by using PKL as a metric (a time-consuming metric to calculate), GA is a suitable tool to optimize this function. The GA we implemented significantly improved the mean % PKL improvement. An example showing a learning curve is shown in Figure 12. Section A states the parameters and functions used during the optimization process, and Section B explains which steps were taken to optimize the trajectories.

A. Parameters

- Population size: The population consists of 10 individuals.
- Number of generations: The population is mutated and evaluated 80 times.
- Cross-over probability: Every specimen has a 50% chance that its' parameter is exchanged for that same parameter but of a different specimen.
- Mutation probability: Every specimen has a 20% chance to mutate a parameter. This is true for 20 out of 30 parameters, resulting in 4 mutated parameters on average.
- Mutation function: Every parameter that is mutated, is modified by adding a randomly generated value ranging from -5 to +5.
- Fitness function: The fitness function is the median PKL improvement of perception including the drones' view compared to perception without the drones' view.

B. Steps

1) **Initialization:** A 15-parameter polynomial is fitted to waypoints which would be the ideal position to be at. This polynomial does not take into account the motion model of the drone and the feasibility of the trajectory. This initial polynomial does serve as a good baseline for the ideal trajectory. An initial population is generated consisting of 3 elite individuals with as a genome the initial polynomials' parameters (15 parameters for x direction, 15 parameters for y direction).
2) **Reproduction:** These 3 specimens are mutated to form a population of 10 specimens.
3) **Evaluation:** The 10 specimens are evaluated by executing the polynomials' trajectory in the simulation and calculating the median PKL improvement.
4) **Selection:** The 3 specimens with the highest fitness are selected as elites.
5) **Crossover:** Crossover takes place between the 3 best individuals to create 7 new specimens.
6) **Mutation:** The 7 new specimens are mutated.
7) Steps 3 to 7 are repeated for every generation.
8) When all generations have been evaluated, the best individual is stored and used as the best expert trajectory.

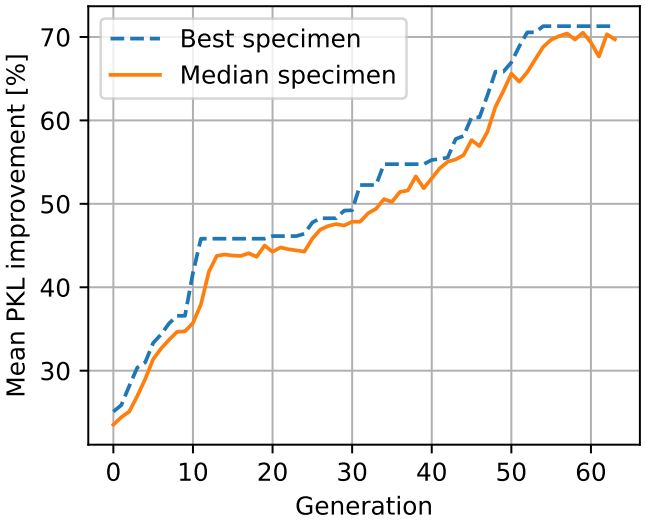


Fig. 12: Learning curve of the GA optimizing a trajectory. The mean % PKL improvement is increased from 26% to 71% in 63 generations

APPENDIX C
REINFORCEMENT LEARNING PARAMETERS

This appendix shows the parameters of the models used.

A. *Proximal Policy Optimization*

```
learning_rate=0.0003
n_steps=128
batch_size=64
n_epochs=10
gamma=0.99
gae_lambda=0.95
clip_range=0.2
normalize_advantage=True
ent_coef=0.0
vf_coef=0.5
max_grad_norm=0.5
```

B. *Behaviour Cloning*

```
policy=PPO
batch_size=32
optimizer_cls='torch.optim.adam.Adam'
ent_weight=0.001
l2_weight=0.0
n_epochs=1000
```