Contents

Chapter 1

Mapping, Environment Modeling, and Decision Making

Acknowledgement: The work in this chapter was done primarily by Rowan Dempster, with help from those acknowledged in Section 1.9.5

1.1 Introduction

Decision making in Autonomous Vehicle (AV)s relies completely on the ability of the agent to aggregate disparate information sources into a useful model of the agent's environment. Examples of disparate information sources include: hand-crafted lane geometries (see top left of Fig. 1.1), tracks produced by a perception scheme (see middle right of Fig. 1.1), or even a prior environment model from a previous drive. In this work, we tackle modeling these disparate sources using relationships, and study how to embed relationships in a unified graphical model we refer to as the Dynamic Relation Graph (Dynamic Relation Graph (DRG)).

Motivating our DRG approach anthropologically, humans do not make decisions based on isolated objects floating around in a disconnected world, but rather using relationships between objects to elicit higher order properties. For example, we infer that the keyboard in front of us has a relationship with the desk it is sitting on. We understand the properties of this relationship: the keyboard is on top, the desk is solid. Using this understanding, we decide that we can carry on typing without the keyboard falling away from our fingers.

Widely accepted prior maps of static environment features are relationship-rich [7, 11, 4]. Such models encode lane neighborhood/successor relations, relationships between traffic lights and the lane(s) they control, and have the potential for much more. These relationships are handcrafted offline by humans during the laborious map creation process. Therefore, the relationships can be highly complex and expressive if we are willing to incur high creation costs (time and effort). However, prior maps are static representations. In order to make dynamic decisions, relationships between online detections and the prior map must be built at system runtime.

Standardized methods and toolboxes for creating and embedding prior-online feature relationships are lacking in the literature, making the logic of AV decision systems opaque. Many different decisions could be made using the same prior and online information, depending on how the relationships are modeled. Some systems must be safer than others, but without standardization there is little hope of being able to compare implementations. The ability to compare implementations goes hand-in-hand with the ability to verify a model, since verification can be done by comparing to some benchmark. Verifying an AV's model of the world is essential to explaining why a decision was made, an area where the AV industry has struggled to meet the public's expectations[1].

In light of these holes in the literature, we propose our work on the DRG which offers the following contributions: (1) The DRG serves as a standard tool for extending prior maps with online observations to create a unified environment model (see Fig. 1.1), allowing for greater transparency and collaboration on decision-making algorithms. (2) A novel method for analyzing the combinatorial complexity of decision making in urban environments, and an analysis of how the DRG reduces these complexities. (3) Design details on various DRG augmentation routines for common environment features, including pseudo-code and a C++ prototype implementation which are released at github.com/WATonomous/DRG.

The rest of this chapter is organized as follows: Section 1.2 presents related work on offline mapping toolsets and attempts at prior-online modeling, followed by our novel DRG methodology of extending a prior relationship graph with online entities in Section 1.3. Section 1.4 then briefly covers our implementation of specific feature designs to illustrate the practical use of the DRG, and Section 1.5 presents the novel method of analyzing the reduction in decision-making complexity afforded by the DRG. Simulation and on-road results are discussed in Section 1.7, conclusions are drawn in Section 1.8, and Section 1.9 serves as an appendix for implementation details. To limit the scope of our proposed work, we are not concerned with ego state estimation errors nor measurement uncertainties,

[1]The infamous Uber ATG fatality in 2018 notably lacked an explanation of the autonomous system's model of the world before the crash, and led to Uber ceasing testing [1].

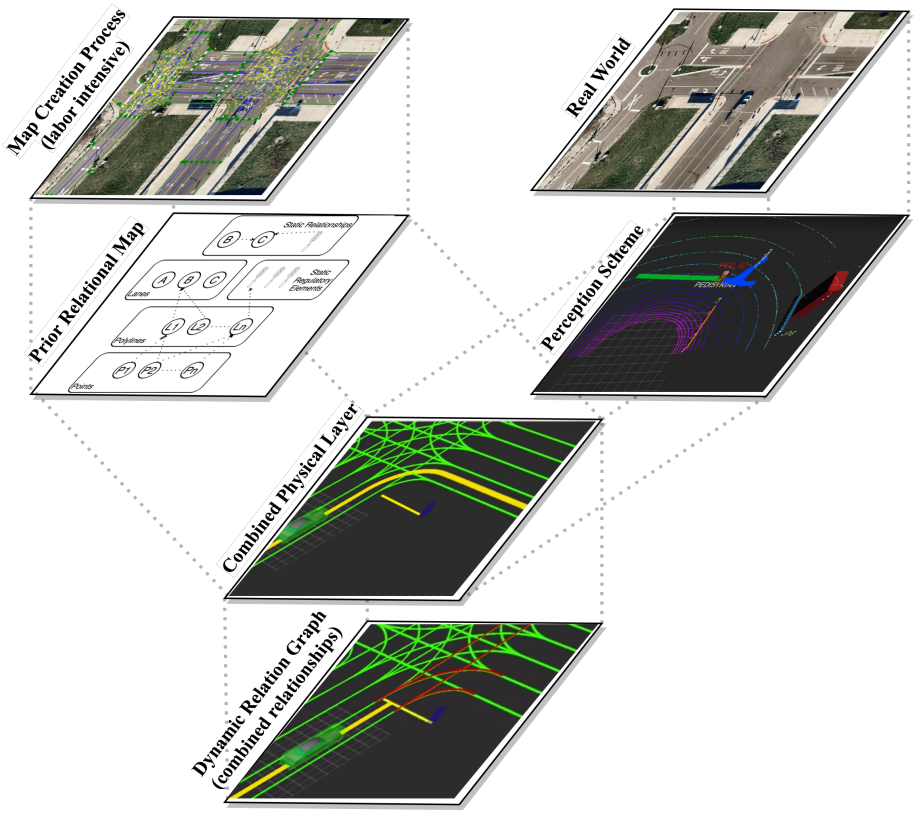

Figure 1.1: The information layers that are fused to create the DRG. Top right is a section of the MCity Course (https://mcity.umich.edu/), top left is the prior map of the same section. The middle three layers show how physical information from the map and tracker are combined, and the bottom layer shows the conflict relationship created.

which are assumed to be handled by upstream modules [2, 3].

1.2 Related Work

The mapping toolset and graph implementation which DRG uses is Lanelet2 [11] [10], an offline tool for handcrafted static maps. A Lanelet2 map consists of three layers: a physical layer, a relational layer, and a topological/routing layer (see Fig. 1.2). The physical layer contains the observable elements of the world represented using points and linestrings. Points are the basic elements of the map described by their three-dimensional position in metric coordinates and linestrings are an ordered array of points used to represent the polygonal elements in the map.

The relational layer forms edges between elements of the physical layer, thus introducing relational entities such as lanelets, areas, and regulatory elements (short: regElems). Lanelets are used to identify sections of the map with directed motion, like vehicle lanes, pedestrian crossings, rails, etc. A lanelet owns one linestring as the left border, one as the right, and optionally owns regElems describing the traffic rules applicable to the lanelet. RegElems are used to define traffic rules such as speed limits or traffic signals[2]. The neighborhood relationships between lanelets generate a topological layer, also known as the routing graph. The routing graph arises from a network of passable regions, where the exact topology depends on the road user at hand (emergency vehicles are allowed to take different routes than passenger vehicles).

Lanelet2 is purely a tool for offline map creation, whereas the DRG is capable of augmenting prior maps autonomously during system runtime, which is far outside of the Lanelet2 design scope. The RoadGraph model [8] [6] and other graph based environment modeling techniques [13] are closer in functionality to the DRG. As with our work, these papers deal with aggregating and fusing information obtained at runtime of the system with a prior map. However, they fall short of our work in key places: (1) They do not give design details or examples of how object tracks from on-board sensors are incorporated into the model (see our design details in Section 1.4). (2) They focus on describing relationships in the prior map, which are now part of the Lanelet2 library. Our work uses the established Lanelet2 library as our prior map and focuses on pushing the design paradigms forwards, towards a unified online model. (3) They do not cover how their RoadGraph model is mutated in cases where certain lanes are no longer traversable due to signage or blockage. Our work examines in detail the problem of modeling features that affect routing decisions.

In [9], Koschi and Althoff describe a reachability set approach to analyzing the interactions between online tracked features and a prior map. The scope of their design is large, considering phantom tracks and abstractions of vehicle dynamic models in their reachability

[2] In our work, the DRG primarily exploits regElems as our graph entity for modeling online observations.

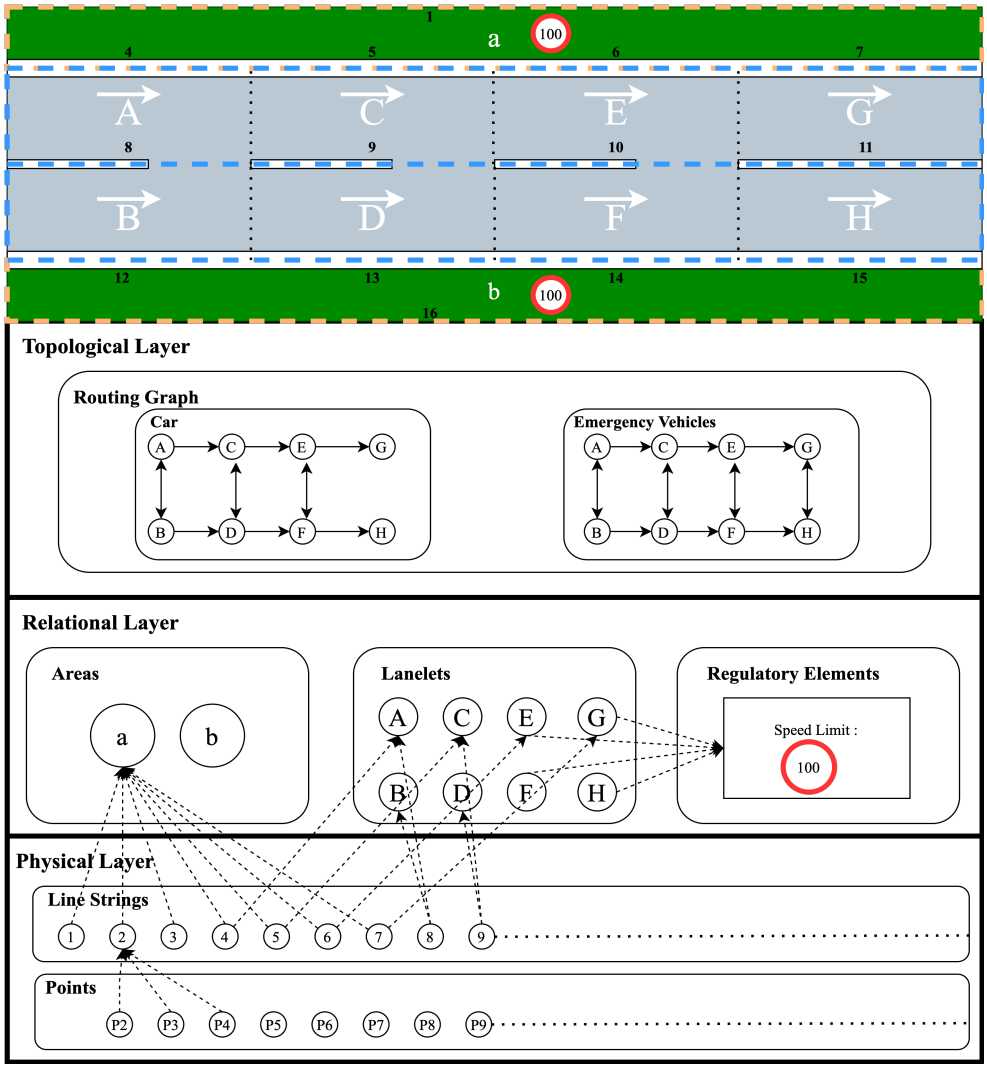

Figure 1.2: A one way street illustrating the different layers of a Lanelet2 map. The lanelets are represented using uppercase letters, the areas using lowercase letters, and the linestrings using numbers.

analysis. However, the paper does not consider extracting the information of surrounding traffic participants from sensor measurements, and the uncertainty of these measurements. In contrast, our approach is integrated into a full scale AV system; we explicitly state and deal with perception, tracking, and occupancy estimation schemes. Additionally, our system was validated in a closed-loop fashion whereas Koschi's and Althoff's approach was

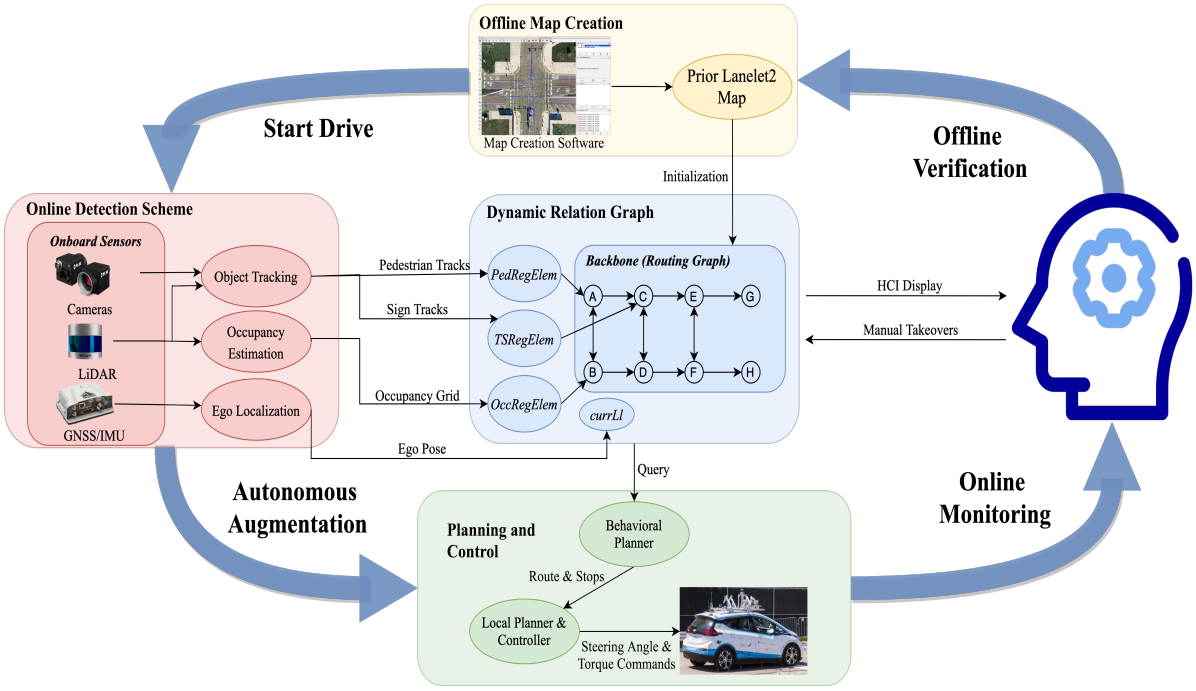

Figure 1.3: Data flow diagram illustrating the DRG's lifecycle during a drive, as described in Section 1.3

.

only evaluated in an open-loop fashion.

1.3 Methodology

The DRG sits between the measurement (perception) and decision (planning) layers of the AV stack (see Fig. 1.3). It represents a map-centric view of the environment, as opposed to the common ego-centric approach [14] [5]. The DRG is implemented as a graph of relationships which is being continuously augmented by the fusion of new online measurements.

Initialization: On AV system start-up, the DRG is initialized to the prior map. The features and relationships in the prior map may have been handcrafted, or they may have been autonomously created and embedded during a previous drive. This flexibility allows the DRG to continuously and autonomously refine itself, with the possibility of humans in the loop for verification and correction.

Augmentation: During a drive, the graph structure of the DRG is augmented over time as new measurements become available. Details of how various specific environment features mutate or create new entities and relationships in the DRG is addressed in Section 1.4. However, the DRG is flexible to any sort of augmentation that can be cast as an update to an existing entity, or as a new entity which has a relationship to an existing entity. All physical information used to create or update an entity or relationship must be stored in the DRG for use in verification as described below.

Querying: The primary function of the DRG is to be queried by the decision making module, providing all necessary physical and relational information. The DRG serves as a single source of truth for the decision making module, and therefore explains every behavioral decision the AV system makes. Section 1.5 details how the decision making complexity of a generic behavioral planner is reduced using the DRG.

Verification: In the event of an accident, the state of the DRG at any point in time can be inspected offline and run through the behavioral planner to reproduce the sequence of decisions that lead to the accident. Other designs that do not have a centralized environment model must store (in the worst case) all sensor data to reproduce behavioral decisions. Furthermore, the state of the relationship graph can be monitored and interpreted online by a safety driver. If at any point in time the relationship graph does not match what the safety driver observes, they can stop the vehicle before a flawed behavioral decision is made. Because of the DRG's map-centric design, online verification and data sharing about the environment between multiple autonomous agents is possible.

1.4 Implementation

The methodology behind the DRG is implementation agnostic, any mapping toolset and graph library implementation can be used. For our prototype we used the Lanelet2 C++ library[3] due to existing implementations of convenient entities like points, linestrings, lanelets, and regElems, as well as methods to describe properties (via the *attributes* API) and relationships (via the *addRegElem* API). Entities that did not already exist, e.g. pedestrians, were implemented as new regElems. The handcrafted lane geometry and topology (i.e. the routing graph) is the "backbone" of the DRG implementation, which can be traversed using the *besides*, *next*, and similar Lanelet2 graph APIs. Thus, graph search

[3]Descriptions and algorithms rely on Lanelet2 Application Programming Interface (API) concepts covered in Section 1.2 and in the Implementation Details Section 1.9, which also contains more detailed descriptions of the augmentations routines mentioned in this section.

routines that allow for expressive queries can be implemented. During runtime the backbone is augmented by mutating entity properties or instantiating new regElem instances and forming relationships between those regElems and the backbone.

The remainder of this section covers four case studies of DRG augmentation routines which run as the AV perceives the environment. It is important to note that the DRG methodology can be applied on a wide range of features using different augmentation routines. The routines presented here are intentionally simple as the focus is on the DRG methodology; more complex routines are discussed in Section 1.8.

A. Stop Signage: In our implementation, we assume that signage is not part of the prior map, and that the DRG ingests tracks of stop signs at runtime with their 3D position and a unique ID. A regElem is instantiated using the *TSRegElem* API to hold the sign's physical information and the *addRegElem* API is used to assign the stop sign to the lanelet(s) it regulates. The regulated lanelet(s) are chosen using a search routine over the DRG backbone. The behavioral planner then queries each lanelet the ego traverses for ownership of the regElem, stopping when necessary.

B. Intersection Signage: Signage such as No Right/Left Turn augments the DRG in a similar manner to stop signs. For each sign, regElems are initialized using the *TSRegElem* API and regulated lanelets are assigned ownership via the *addRegElem* API. The regulated lanelet search routine over the DRG backbone is presented in Algorithm 1. After the regElem is owned by the regulated lanelets, the behavioral planner queries for a new route that obeys the augmented routing graph because it is possible the previous route is now disallowed by the new regulation.

C. Closed Roads: The DRG also ingests occupancy grid information about the environment, from which untraversable lanelets can be identified. A regElem is initialized using the *OccRegElem* API to store the blocking occupancy grid, which is then added to each untraversable lanelet. The behavioral planner again uses the *findRoute* API to search for a new traversable route.

D. Pedestrian Movement: In our implementation, the DRG ingests tracks of pedestrians with their predicted and historical states. A regElem is instantiated using the *PedRegElem* API which holds the predicted and historical states. A lanelet conflict set is generated for the pedestrian based on its predicted and historical states as described in Algorithm 2. The regElem is then augmented with the stopping point of each conflict and added to the conflicting lanelets. Similar to stop signs, the behavioral planner queries each lanelet the ego traverses for ownership of *PedRegElems* and stops at the designated stopping point.

Algorithm 1 Intersection signage augmentation

Input: *sign*, *DRG*
Output: Augmented *DRG*

Initialize *regElem* ← *TSRegElem(sign.type, sign.pos)*
Initialize *interLls* ← {} {Empty dictionary of relevant intersection lanelets}
adjacentLls ← *currLl.besides()* {Set of all lanelets adjacent to the ego vehicle}
for *ll* ∈ *adjacentLls* **do**
    interLls.insert(DRG.findInter(ll)) {*DRG.findInter* API searches for a successor intersection lanelet}
end for
for *ll* ∈ *interLls* **do**
    if *DRG.signControls(regElem, ll)* {If the sign controls the turning direction of the lanelet} **then**
        ll.addRegElem(regElem)
        DRG.findRoute() {*DRG.findRoute()* API searches for a new route given the new regulation}
    end if
end for

Algorithm 2 Pedestrian movement augmentation

Input: *ped*, *DRG*, *waitDist* {The conflict radius around a waiting pedestrian}
Output: Augmented *DRG*

regElem ← *PedRegElem(track)*
confLls ← *DRG.within(regElem.predictedStates)*
if *regElem.isWaiting()* **then**
    confLls.insert(DRG.within(regElem.currState, waitDist))
end if
for *confLl* ∈ *confLls* **do**
    stopPoint ← *intersect(confLl, track.predictedStates)*
    regElem.setStop(stopPoint)
    confLl.addRegElem(regElem)
end for

1.5 Decision Complexity Analysis

In this section, we take a general view of behavioral planners, analyzing how any given planner deals with the combinatorial complexity of the perceived environment features.

In the literature, there is a notable lack of techniques for expressing the combinatorial complexity of making a decision, and thus we present our own novel approach: Let $F = \{C_1, \ldots, C_N\}$ be the feature class sets for the N distinct classes of features, (e.g. C_{ped} represents the set of pedestrians and $C_{ped}^{(1)}$ is a specific pedestrian instance). Let $E = \{e_1, \ldots, e_P\}$ be the set of atomic ego behaviors (a simple binary planner, with $P = 2$, may have $e_1 = driving, e_2 = stopping$). The task of any given behavioral planner is to implement a mapping between the *feature space* F, of size $N * \sum_{C_i \in F} |C_i|$, to the planner's range E.

There are two sources of combinatorial complexity in this mapping, intra- and inter-class interactions. For example, in the pedestrian feature set C_{ped}, intra-class interactions arise from considering pairs of instances ($C_{ped}^{(1)}$, $C_{ped}^{(2)}$) and extracting aggregated information such as minimum distance to the ego vehicle. Inter-class interactions must also be addressed. Consider the case where C_{cw} is the set of pedestrian crosswalk lanelets specified in the prior map and how the planner reacts depends on the pedestrian's proximity to the C_{cw} features. Here, the interaction between the pair of classes (C_{ped}, C_{cw}) is pertinent.

Let $I_{inter}(\cdot, \cdot)$ and $I_{intra}(\cdot, \cdot)$ be the interaction functions, which express the inter- and intra- sources of combinatorial complexity (see Fig. 1.4 for an illustration of these functions and their composition). Note that the output spaces of I_{inter} and I_{intra} scale quadratically over the number of feature classes, and the number of instances in each feature class, respectively:

$$\Theta(I_{inter}(F, F)) = |F \times F| = \Theta(N^2) \tag{1.1}$$

$$\Theta(I_{intra}(C_i, C_i))) = |C_i \times C_i| = \Theta(|C_i|^2) \tag{1.2}$$

Applying the $I_{intra} \circ I_{inter}(F)$ (or equivalently, $I_{inter} \circ I_{intra}(F)$) composition to the original feature space F produces a quadratic *decision space* D of size $N^2 * \sum_{C_i \in F} |C_i|^2$ (see bottom left of Fig. 1.4).

Our claim is that the relational graph structure of the DRG benefits any planner using it by mitigating the quadratic complexity of the decision space introduced by $I_{intra}(\cdot, \cdot)$ and $I_{inter}(\cdot, \cdot)$, resulting in an algorithm that scales linearly with the number of feature classes and instances.

First, regarding inter-class interactions, the DRG introduces a set of homogeneity operators $H_{ij}(C_{ll}) : C_i \mapsto R_j(C_{ll})$ which map heterogeneous feature classes C_i to homogeneous representations R_j. An operator is defined over all i, and for $j \in \{1...K\}$ where importantly

K is a small constant. The implementation of the operators depends on the prior map features, and thus all H_{ij} are parameterized by C_{ll}. Together, the set of operators transforms all heterogeneous feature classes into a homogeneous *representation space* $R = \{R_1 ... R_K\}$.

Concretely, $H_{ij}(C_{ll})$ is defined for $i = \{ped, car, cyclist\}$ and $j = lanelet\ conf$ by abstracting each heterogeneous feature class into a homogeneous lanelet conflict representation as described in Section 1.4-D for pedestrians. Furthermore, $H_{ij}(C_{ll})$ is also defined for $i = \{intersection\ sign, blocked\ road\}$ and $j = untraversable\ lanelet$ by applying the techniques presented in Sections 1.4-B and 1.4-C.

After applying the set of homogeneity operators to the feature space F, the subsequent application of $I_{inter}(\cdot, \cdot)$ is performed on the constant-sized representation space R (see Fig. 1.4 (right)). The effect is a reduction in inter-class combinatorial complexity:

$$\Theta(I_{inter}(R, R)) = |R \times R| = \Theta(K^2) = \Theta(1) \tag{1.3}$$

Note that $\Theta(N)$ homogeneity operators need to be defined, one for each of the N feature classes.

Next we discuss interactions of intra-class instances. For this discussion either the original feature space F or the representation space R can be used; we will use the feature space notation. Note that an instance $C_i^{(j)}$ becomes independent of another instance $C_i^{(k)}$ when conditioned upon their inter-class interaction with the prior map. For example, consider the case of $i = ped$. Here, the relative positions or trajectories of two pedestrians are irrelevant and can be ignored, given their relationship (a potential lanelet conflict) to the prior map (C_{ll}). Let $I_m = I_{inter}(C_{ll}, C_i^{(m)})$ be instance m's inter-class interaction with the prior map. The effect on intra-class combinatorial complexity is then:

$$\Theta(I_{intra}(C_i, C_i)) = \Theta(|C_i|) \quad \text{given } I_m, \forall m \in C_i \tag{1.4}$$

Eq. (4) effectively states that intra-class interactions between instances can be ignored under knowledge of their interactions with the DRG backbone, allowing the decision algorithm to process each feature class in $\Theta(|C_i|)$ time. Moreover, as mentioned previously, applying the homogeneity operations and calculating the inter-class interactions on the representation space instead of the feature space yields a constant number of inter-class interactions. Overall, the decision space size under the DRG is reduced to $\Theta(\sum_{C_i \in F} |C_i|)$ (see Fig. 1.4 (right)); linear in the number of observed features. This theoretical analysis is tested in Section 1.7.1, where a simulation with a crowd of pedestrians is carried out.

In summary, by solving the generic behavioral planner task $F \mapsto E$ in the context of the DRG structure, we are able to limit or completely eliminate inter- and intra- class

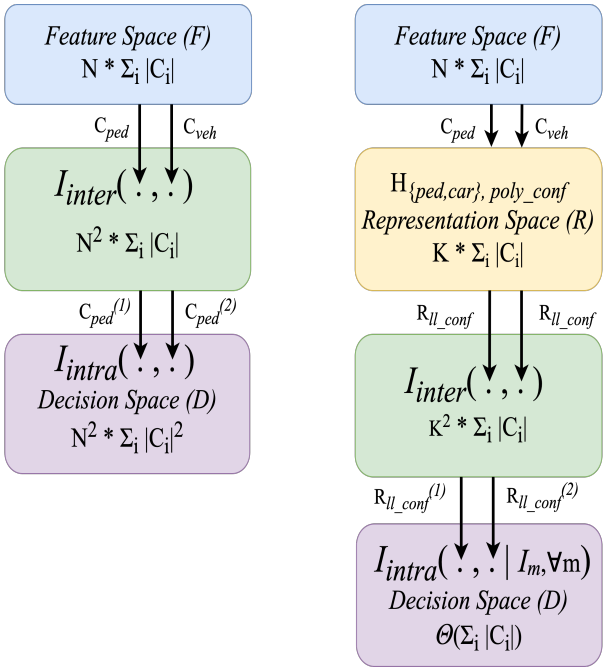

Figure 1.4: Diagrams illustrating the combinatorial complexity of decision making in different contexts. See Section 1.5 for symbol definitions.
Left: In the context of a generic planner. Right: In the context of a planner which employs the DRG. Note that under the DRG, representation space R is captured using $O(N)$ homogeneity operators, resulting in a $\Theta(N) + \Theta(\sum_{C_i \in F} |C_i|)$ design complexity.

interaction checks, resulting in an $\Theta(N) + \Theta(\sum_{C_i \in F} |C_i|)$ algorithm (where $\Theta(N)$ represents the number of homogeneity operators needed and $\Theta(\sum_{C_i \in F} |C_i|)$ is the size of the decision space). We expect that this is a lower bound on the complexity of understanding and reacting to each detected feature instance.

1.6 Verification and Interpretability

There are three properties that make our view of the world easy to verify and interpret. The first property is the relationship graph itself, which is an explicit encoding of the behavioral planner's understanding of the world. Therefore, any behavioral decision must be derived using information in the relationship graph. The second property is inherited from the Lanelet2 design philosophy: Our feature designs (implemented as regElems) always

contain the complete physical information that was used to infer the relationships. Therefore, a behavioral decision can always be traced back, through the relationships graph, to physical information that was observed. The last property is a subtle but distinguishing factor: The DRG is not ego-centric, but rather map-centric.

These properties benefit verification and interpretability in the following ways:

1. In the event of an accident, the state of the DRG at any point in time can be inspected offline and run through the behavioral planner to reproduce the sequence of decisions that lead to the accident. Other designs that do have a centralized environment model cannot do this.
2. The state of the DRG can be monitored and interpreted online by a safety driver. If at any point in time the relationship graph does not match what the safety driver observes, they can stop the vehicle before a flawed behavioral decision is made.
3. Because of the DRG's map-centric design, online verification and data sharing about the environment between multiple autonomous agents is possible. This is not possible in most behavioral planning schemes, which have an ego-centric view of the world [14] [5].

1.7 Evaluation

The efficacy of the DRG was evaluated using a stress test in simulation and real world deployment onto the WATonomous research vehicle (Bolty).

1.7.1 Simulated Pedestrian Crowd

In simulation we were able to assess and prove the reduction in behavioral planner complexity analyzed in Section 1.5. Using the CARLA simulator [12] we spawned increasingly large crowds of pedestrians (see Fig. 1.5) and measured the runtime of the DRG augmentation and behavioral planner query. As seen in Fig. 1.6, the runtime scales linearly as the number of pedestrian instances increases. Even with 25 pedestrian instances the planner runtime stays real-time, under 500 ms. This result is expected due to the elimination of intra-class interactions afforded by the DRG as analyzed in Section 1.5.

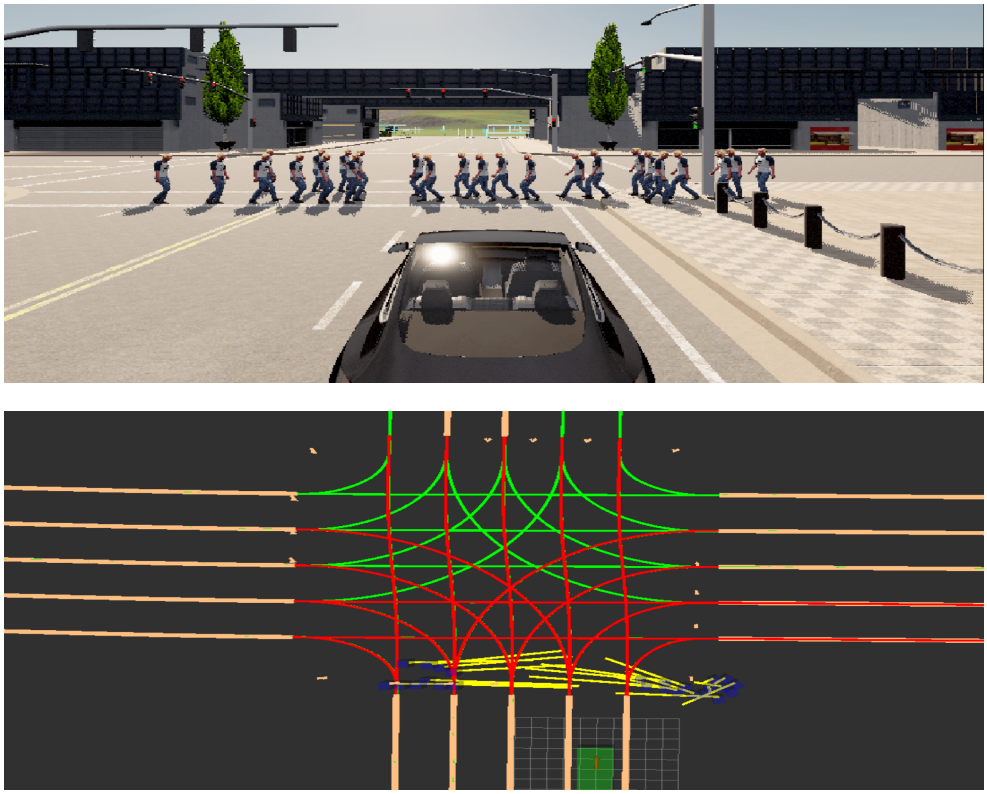

Figure 1.5: The simulated pedestrian crowd used to stress test the DRG and measure the accuracy of the claims made in Section 1.5. The yellow lines are linear state predictions of the pedestrians obtained from the simulator's ground truth, from which lanelet conflicts (indicated in red) are extracted as described in Section 1.4-D

1.7.2 Vehicle Deployment

The DRG was primarily evaluated by deploying the prototype implementation to a research vehicle (Bolty) performing AV taxi routes in a closed course. The lane geometry and relational topology was handcrafted offline using the Lanelet2 toolset. At runtime the AV system encountered stop and intersection signage, blocked roads, pedestrians, and traffic signals. All of these features were observed online (not part of the handcrafted map) and the DRG was augmented in real time, enabling the behavioral planner to obey the traffic rules.

Overall, results of the experiments were promising; the AV system was able to navigate the course and obey all traffic rules, including stops and re-reroutes, by using the DRG.

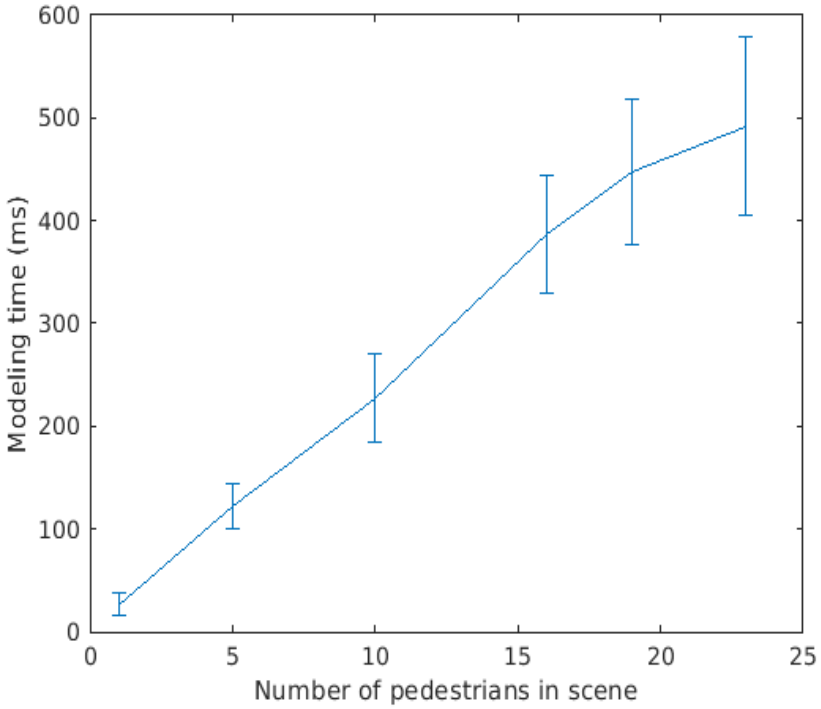

Figure 1.6: Plot showing the linear relationship between the number of observed pedestrian features and the complexity of decision making under the DRG. Execution time in milliseconds is used as the complexity metric.

The full taxi route can be viewed at https://youtu.be/DNZgheT4Y2s?t=153. Additionally, the safety driver was able to continuously monitor the state of the DRG, verifying its correctness or bringing the vehicle to a stop when inconsistencies appeared.

1.8 Conclusion

In this work, we have shown how prior relational maps can be extended to include online observations, and the advantages this approach has for reducing behavioral planning complexity and verification. The proposed Dynamic Relation Graph is a natural and effective step forward towards a unified model of encoding both prior and online information sources. We believe that the proposed scheme will standardize how AV systems encode the richness of relationships in the world, and will allow for greater collaboration in the

development of AV decision-making algorithms. Our main contribution in this regard is a detailed insight into how such a relationship graph can be implemented in a real world, closed loop system in the context of a standard perception and mapping schemes.

One of the main advantages of the DRG framework is its flexibility to accommodate many different feature model implementations, which will be the goal of our future work. We plan to enhance our pedestrian model using the set reachability methods presented in [9], and employing learning based approaches where procedural logic falls short. We also aim to perform online verification and consistency checking experiments on a single DRG instance shared between multiple autonomous agents observing the environment.

1.9 Implementation Details

1.9.1 DRG and Lanelet2 API

All Lanelet2 entities (points, linestrings, etc...)	
attributes	Dictionary member variable that stores properties.
Lanelets	
addRegElem	Function that assigns ownership of a regElem to the lanelet.
besides	Function that returns the lanelet(s) to the left and right.
next	Function that returns the successive lanelet(s).
RegElems	
TSRegElem	RegElem subtype that stores traffic sign positions and types.
OccRegElem	RegElem subtype that stores an occupancy grid.
PedRegElem	RegElem subtype that stores an a pedestrian track with historical and predicted states.
.predictedStates	PedRegElem member variable storing a linestring representation of the predicted states.
.isWaiting	PedRegElem boolean function indicating whether a pedestrian is not moving based on its historical states.
.setStop	Mutates a PedRegElem by adding a point on the owning lanelet where the ego vehicle must stop for the pedestrian.
.currState	Member variable which stores the current position of the pedestrian.
DRG	
findInter	Performs Dijkstra's search to find the closest lanelet tagged with a *turn_dir*, where the edge weights are the lengths of lanelets. Then performs a max cost depth first search to find all other intersection lanelets within a distance threshold from the closest lanelet.
signControls	Checks if an intersection sign type applies to a lanelet based on the lanelets's *turn_dir* attribute.
findRoute	Performs Dijkstra's search from *currLl* to the destination over the routing graph.

within	Returns the lanelet(s) which contain the point or linestring operand, optionally accepts a distance operand which expands the search by that radius.
Global variables and functions	
currLl	The lanelet that the ego vehicle is driving in.
intersect	Returns the geometric intersection(s) of two linestrings.
RoutingGraph	
canPass	Boolean function which consumes a lanelet and a traffic participant, returning whether that lanelet is traversable by the given traffic participant.

1.9.2 Stop Signs

The DRG aims to associate a tracked stop sign to the lanelet(s) that it regulates. The objective is to first obtain a set of candidate lanelets based on a search radius[4] around the sign's position, and then to determine which of the candidate lanelets contain the ego vehicle (the controlled candidate). The controlled candidate as well as its adjacent lanelets form the controlled set, to which the sign is associated.

A schematic diagram of the process is shown in Fig. 1.7A. The green rectangle represents the ego vehicle approaching the intersection and the three parallel red lines identify the ego vehicle's lanelet. When the stop sign is observed, all lanelets in a search radius of the stop sign are considered candidate lanelets (highlighted with orange and blue lines). All lanelets that are adjacent to the initial candidate lanelets are also added to the set of candidate lanelets. If the ego vehicle is inside the bounds of a candidate lanelet, that lanelet is the controlled candidate (the red lanelet). Additionally, all lanelets adjacent to the controlled candidate are regulated (the blue lanelet). Therefore, if the ego vehicle changes lanes after the initial construction of the controlled set, it will still be regulated by the stop sign.

In our implementation we use the *TSRegElem* class, a custom subclass of Lanelet2's *RegElem* class, to model the physical stop sign. The *TSRegElem* is constructed using the stop sign's tracked location, so that offline verification of associations is possible. After being constructed, the *TSRegElem* is added to each lanelet in the controlled set via the *addRegElem* API. As a result, the behavioral planner can determine if the lanelet the ego

[4]The search radius is a parameter in this approach. In the proposed implementation, a search radius of 9.2 m is used, based on the maximum lane width of 4.6 m.

is traversing is regulated by a stop sign by looking at the *TSRegElem*(s) that the lanelet owns.

1.9.3 Intersection Signage

This subsection covers associating tracked intersection signs to the virtual lanelets inside of the intersection they control. Signs include: No Right/Left Turn, Right/Left Turn Only and Do Not Enter. In addition, we cover the implications on routing after association.

Our objective is to construct relationships between traffic signs and upcoming virtual intersection lanelets in order to block potential routes that disobey traffic rules. First, all detections of the five mentioned traffic sign types are collected from the tracker. For each tracked sign, we instantiate a *TSRegElem* tagged with the sign's type. Each *TSRegElem* stores the location and type of sign and will be used to model relationships between the sign and the lanelet(s) it regulates.

In order to determine which lanelets are regulated by a *TSRegElem*, we must query the Lanelet2 map. We begin by querying all lanelets adjacent to the ego's current lanelet. Next, we execute a search procedure, starting at each adjacent lanelet, through successor lanelets with the goal of finding all intersection lanelets tagged with a *turn_direction* attribute. This attribute is a custom *a priori* attribute on the virtual intersection lanelets which stores the turn direction (right, straight, left) of the lanelet. The search for intersection lanelets is done in two steps:

1. Perform Dijkstra's search to find the closest lanelet tagged with a *turn_direction* (i.e. the closest intersection lanelet) where the edge weights are the lengths of lanelets. Dijkstra's algorithm was selected for its simplicity and because we know that the search space is a small, constant, number of levels deep.

2. Perform a Max Cost Depth First Search to find all other intersection lanelets within a distance threshold from the closest lanelet.

We now have a set of intersection lanelets in our direction of travel that are candidates to be associated with a sign. In Fig. 1.7B the candidates are drawn in red. By comparing and matching the *turn_direction* attribute with the type of the *TSRegElem*, we add the appropriate *TSRegElem* to the lanelet(s) that are regulated by the sign. For example, a "right" *turn_direction* lanelet would have the "no right turn" *TSRegElem* applied to it (as shown in Fig. 1.7B).

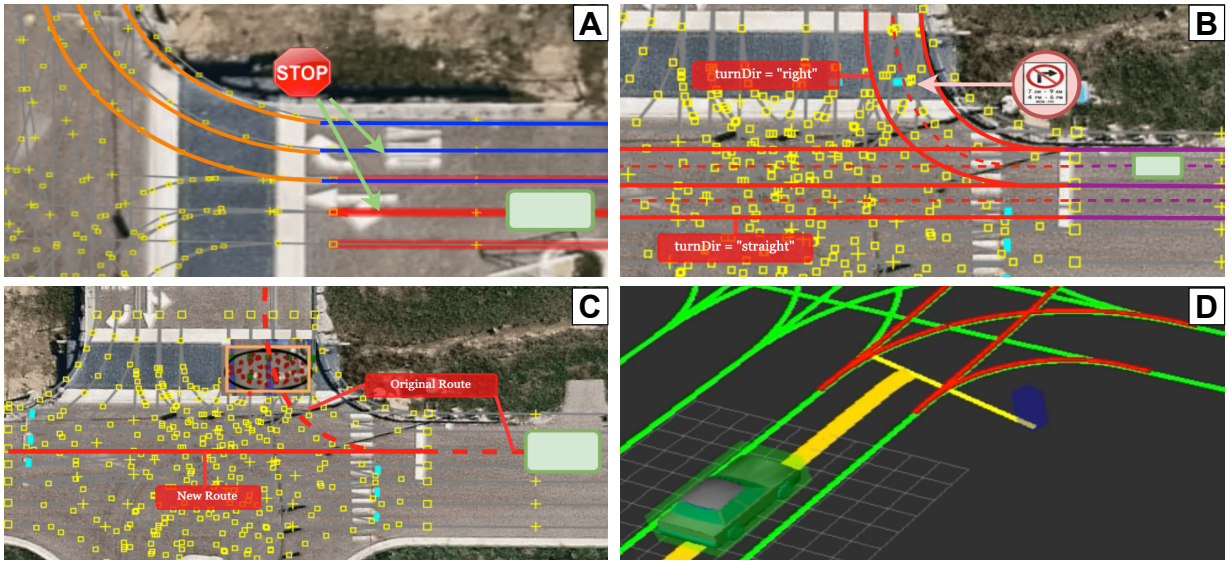

Figure 1.7: Illustrative schematics for (A) stop signage, (B) intersection signage, (C) occupancy, and (D) pedestrian modeling. The schematics are drawn on top of a 4-way intersection at the MCity Course.

By adding each *TSRegElem* to the lanelet(s) it regulates, we have a model of how the traffic signs relate to the prior map. However, we have yet to take this information into account in the context of routing decisions, that is, the Lanelet2 *RoutingGraph* still holds the old lanelet connectivity information. In order for our *TSRegElem*s to affect the *RoutingGraph* connectivity, we overrode the *canPass* traversability evaluation of each lanelet to take into account *TSRegElem* owned by that lanelet. However, the current *RoutingGraph* implementation is immutable after being created, since it was designed assuming complete *a priori* knowledge of information relevant to routing. Thus, in order to update connectivity, we must destroy and recreate the entire *RoutingGraph* whenever a lanelet has its *canPass* evaluation changed.

1.9.4 Closed Roads

This subsection focuses on modeling occupancy of the road surface in a fashion that stores both the physical specification of the occupancy and the relationship it has to specific lanelets. Our goal is to determine if there are any obstacles on the road, which lanelet(s) are affected, and what impact this has on routing.

Occupancy grid estimation is done with a VLP-32 LiDAR installed on the ego vehicle. To establish a relationship between the occupancy grid and lanelets, we first align the occupancy grid coordinate frame with the map frame. Next we check if a lanelet is blocked by looping through all grid cells that overlap with the lanelet and check if any cells are occupied. We do this using functions that project from a map coordinate to a grid cell and vice versa.

At this point, the algorithm attempts to find a maneuver around the obstacle by switching lanes. The lateral shift distance needed to avoid the obstacle is based on the leftmost and rightmost bounds of the occupied area. Two outcomes are possible depending on the lateral shift distance and the existence/occupancy of adjacent lanelet(s): (1) There exists an unblocked adjacent lanelet that the vehicle can switch to and avoid the obstacle while staying on the current route, (2) all adjacent lanelets are blocked, and a reroute is necessary (Fig. 1.7A illustrates this case).

In the second case we must mutate the Lanelet2 map and *RoutingGraph* to add information about the blockage. In the map, we create a relationship between the observed physical occupancy and the blocked lanelet(s) using a custom Lanelet2 *RegElem* subclass, *OccRegElem*. An *OccRegElem* object is instantiated with the relevant occupancy cells and added to each of the blocked lanelets. In the *RoutingGraph*, the new *OccRegElem* is interpreted in the same manner as in Section 1.9.3, by overriding the *canPass* method in a custom *RoutingGraph* subclass, returning false if the lanelet owns an *OccRegElem*. Finally, the *RoutingGraph* needs to be destroyed and re-created with the new occupancy information so that a new route can be computed.

1.9.5 Pedestrian Movement

In this subsection we aim to model pedestrian movement and interactions with the prior map. Modeling is done by calculating a lanelet conflict set for each tracked pedestrian, and constructing a relationship between the pedestrian's physical attributes and each lanelet in that conflict set. First, we examine the spatial interactions between the pedestrian's predicted movement (from the tracker) and the lanelets in the map coordinate frame. Lanelets whose polygonal boundary intersects with the predicted movement of the pedestrian are added to the conflict set (see Fig. 1.7D). However, even if a pedestrian is not moving, it could still be in behavioral conflict with a lanelet, e.g. if waiting to cross the street. Therefore, if a pedestrian is not moving and is within a distance threshold from a lanelet, that lanelet is also added to the conflict set. This behavioral conflict is an example of how our environment model explicitly encodes behavioral decisions, simplifying behavioral planning and verification as discussed in Section 1.5.

Once the conflict set has been determined, we calculate and store properties about the relationship between the pedestrian and each conflicting lanelet. These properties are used in the future tasks of behavioral planning and verification. We briefly review these properties and the motivation for including them:

1. Stop Point: the spatial point in the map frame at which the pedestrian's predicted movement intersects with the lanelet's center line; or the point closest to the pedestrian if the pedestrian is waiting. Used in behavioral planning to determine where the ego vehicle should stop for the pedestrian.

2. Predicted/Past Movement: a copy of the information provided by the tracker that was used to calculate this conflict. Necessary for verification.

The C++ instantiation of the relationship is done via a new subclass, *PedRegElem*, of *RegElem* class, which stores the physical properties of the relationship. The *PedRegElem* object is added to each lanelet in the conflict set. This implementation allows for easy access to all relevant *PedRegElem* information in the behavioral planning phase. To determine if a lanelet along the ego's current lanelet path conflicts with a pedestrian, the behavioral planner simply queries each lanelet on that path, asking it if it owns a *PedRegElem*. If so, the behavioral planner can then retrieve the *PedRegElem* object and the *stopPoint* property.

Acknowledgment

We would like to thank Maahir Gupta, Wanda Song, Christopher Mannes, and Jacob Armstrong for their help implementing and evaluating the prototype.

References

[1] Self-driving uber car that hit and killed woman did not recognize that pedestrians jaywalk. http://www.nbcnews.com/tech/tech-news/self-driving-uber-car-hit-killed-woman-did-not-recognize-n1079281. Accessed: 2021-02-17.

[2] Mohammad Al-Sharman, David Murdoch, Dongpu Cao, Chen Lv, Yahya Zweiri, Derek Rayside, and William Melek. A sensorless state estimation for a safety-oriented cyber-physical system in urban driving: deep learning approach. *IEEE/CAA Journal of Automatica Sinica*, 8(1):169–178, 2020.

[3] Mohamed A. Daoud, Mohamed W. Mehrez, Derek Rayside, and William W. Melek. Simultaneous feasible local planning and path-following control for autonomous driving. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–13, 2022.

[4] Marius Dupuis, Martin Strobl, and Hans Grezlikowski. 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*, pages 231–242, 2010.

[5] M. Fu, W. Song, Y. Yi, and M. Wang. Path planning and decision making for autonomous vehicle in urban environment. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pages 686–692, 2015.

[6] Kai Homeier and Lars Wolf. Roadgraph: High level sensor data fusion between objects and street network. In *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1380–1385. IEEE, 2011.

[7] Yunfan Kang and Amr Magdy. Hidam: A unified data model for high-definition (hd) map data. In *2020 IEEE 36th International Conference on Data Engineering Workshops (ICDEW)*, pages 26–32. IEEE, 2020.

[8] Jörn Knaup and Kai Homeier. Roadgraph-graph based environmental modelling and function independent situation analysis for driver assistance systems. In *13th International IEEE Conference on Intelligent Transportation Systems*, pages 428–432. IEEE, 2010.

[9] Markus Koschi and Matthias Althoff. Set-based prediction of traffic participants considering occlusions and traffic rules. *IEEE Transactions on Intelligent Vehicles*, 2020.

[10] Fabian Poggenhans and Johannes Janosovits. Pathfinding and routing for automated driving in the lanelet2 map framework. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–7. IEEE, 2020.

[11] Fabian Poggenhans, Jan-Hendrik Pauls, Johannes Janosovits, Stefan Orf, Maximilian Naumann, Florian Kuhnt, and Matthias Mayr. Lanelet2: A high-definition map framework for the future of automated driving. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 1672–1679. IEEE, 2018.

[12] CARLA Development Team. Carla simulator. Available at https://carla.org/.

[13] Simon Ulbrich, Tobias Nothdurft, Markus Maurer, and Peter Hecker. Graph-based context representation, environment modeling and information aggregation for automated driving. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pages 541–547. IEEE, 2014.

[14] J. Wei, J. M. Snider, T. Gu, J. M. Dolan, and B. Litkouhi. A behavioral planning framework for autonomous driving. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pages 458–464, 2014.