

HUMANAV: Human-like Semantic Navigation for Autonomous Vehicles

Augusto Luis Ballardini *Member, IEEE*

Abstract—The navigation of self-driving vehicles remains far from achieving the full automation level required to handle all driving scenarios. Understanding the driving environment is an essential component of any driving system. Despite recent advancements in deep learning, current approaches struggle to handle navigation plans involving unpredictable changes in road layouts, such as spontaneous detours or missing map data, due to their reliance on pre-existing cartographic information. In the HUMANAV project, we introduce the study and development of an innovative navigation system for autonomous vehicles, which leverages knowledge represented by environmental detections through a reasoning scheme using Answer Set Programming (ASP), which allows for flexible reasoning in uncertain and dynamic environments thanks to its non-monotonic reasoning scheme. The proposed semantics allow for navigation that closely resembles human behavior, without relying strictly on existing cartographic material. The advantages of this system include its flexibility in handling natural language navigation instructions, such as “turn left at the big tree” instead of “in 200 meters, turn left”, resulting in smoother and more intuitive interactions. Additionally, the system is adaptable to frequent changes in the road network, such as the replacement of intersections with roundabouts. Preliminary experiments demonstrated that our ASP-based approach could effectively adapt to dynamic scenarios, successfully generating context-aware navigation instructions in real-time.

Index Terms—Self-Driving, Localization, Scene Understanding, Urban Settings, Semantics, Answer Set Programming.)

I. INTRODUCTION AND RELATED WORK

THIS project proposes the development and integration of a semantic localization and navigation system into a real vehicle. Autonomous driving and assistance systems are set to transform the future mobility. According to the current state of the art, a key requirement for this transformation is the precise and robust localization of the vehicle, meaning the vehicle must know its position relative to its surroundings at all times. Currently available Advanced Driver Assistance Systems (ADAS) are basic autonomous driving systems that rely on data acquired from sensors such as cameras, sonars and LiDAR. These systems have rapidly evolved over

the past decade thanks to technological advancements and the introduction of machine learning techniques and neural networks. They currently represent the state of the art in commercially deployed systems, performing best on fast roads (highways and expressways), which are characterized by relatively simple areas of vehicle interaction. However, urban environments require a much more advanced understanding of the surroundings, as the potential levels of traffic interaction and routing dynamics requires a level of reliability that cannot always be provided by a system solely based on sensors. On the other hand, more advanced research vehicles such as the Waymo fleet rely on highly accurate maps commonly known as High-Definition Maps (HD-MAPS) and a sensor network that identifies the vehicle’s real position relative to the map [1].

Despite the progress made, even with the steady improvements in ADAS over recent years, the current state of technology is still far from achieving full automation as defined by the Society of Automotive Engineers (SAE) LEVEL-5 [2], where all driving tasks are automated and can be carried out reliably by the vehicle with no human intervention required.

The objective of the Human-like Semantic Navigation for Autonomous Vehicles (HUMANAV) project is to study and development of an innovative navigation system for autonomous vehicles based that leverages the detected environment without relying on cartographic maps. The aim is to create a localization system that enables semantic navigation, which means it is not confined to a purely metric context. This approach will allow, among other things, navigation that more closely resembles human experience, without being strictly dependent on existing cartographic material, while adapting to unexpected detours and changes. The advantages of this system include its adaptability to frequent modifications in the road network, such as the replacement of intersections with roundabouts or the request for a specific action like *turn right*.

To achieve this, the project aims to mimic human behavior during the driving of a vehicle, primarily in urban environments. Emulating the way we humans drive allows for a clear separation between the vehicle’s control system and the perception of the environment at

All authors are with the Computer Engineering Department, University of Alcalá, Alcalá de Henares, Madrid, Spain. Email: augusto.ballardini@uah.es,

a more abstract, semantic level, where vehicle’s actions are related to reasoning rather than simply following a predefined, static metric plan, as would be the case for a robot operating in a controlled environment.

II. WHY

Despite the improvements made in recent years in both the reliability and availability of commercial systems, most ADAS still perform best in environments where road geography is relatively simple and, therefore, can be accurately represented with maps or other cartographic references. On the other hand, in urban environments and scenarios where there is no “Manhattan-like” urban planning, that is, large blocks surrounded by straight avenues and simple geometries, these systems are only capable of driving under very specific conditions, are reliable most of the time at low speeds, and can only be operated under human supervision.

The transition to full automation requires reliability in all circumstances, including narrow streets that are difficult to represent using conventional mapping techniques. In 2019, an estimated 22 700 people were estimated to have died in traffic accidents in Europe [3]. Furthermore, according to the 2018 annual accident report [4], 19.4% of traffic fatalities occur in environments similar to intersections, and 38% occur in urban areas. Similar data is also provided in the DGT (Spanish Traffic Authority) report [5]. This demonstrates the significant potential for an autonomous driving system capable of recognizing complex urban environments, particularly areas of intersections, to improve road safety and reduce the number of casualties.

Typically, navigation in urban environments is based on the analysis of images and/or 3D point clouds created with Light Detection And Ranging (LiDAR) sensors, used to create a virtual representation of the vehicle’s surroundings. State-of-the-art methods for understanding these environments include systems based on machine learning techniques such as Support Vector Machine (SVM) [6], Conditional Random Field (CRF) [7], and various types of deep neural networks, including Convolutional Neural Network (CNN) and Recursive Neural Network (RNN) [8]. In non-urban environments, the analysis of images and point clouds is combined with cartographic information to achieve robust localization. Due to the unpredictable nature of urban environments, where unexpected obstacles or changes in road layout can occur, relying solely on virtual representations and maps for navigation is not always feasible. Figure 1 shows an example where Tesla’s AutoPilot system misinterpreted the position within an intersection, entering the wrong way on the left street. Another example: today, if an intersection is replaced by a roundabout, an automatic system using only cartography is unable to

handle this situation. Similarly, the lack of updated maps would render any map-based autonomous driving system useless. In such cases, a human driver would simply try to follow the direction of travel semantically, choosing the correct exit from the roundabout or following the high-level directions provided by a copilot or external user, such as “*follow the road and take the third exit at the roundabout*”. This is the behavior that the proposed project aims to emulate.

A similar approach to our proposal is found in [10], where the authors emphasize that the widespread adoption of autonomous vehicles will depend on their ability to effectively interact with human users. Following natural-language directions, and sharing our idea to remove the challenges posed by poor or outdated map coverage, they implement a control system leveraging images and linguistic input. However, despite the idea of following human instructions, the proposed system remains an end-to-end solution with no focus on explainability. With the broad use of Large Language Models (LLMs), several works that leverage these systems have been extensively analyzed in the last year. In [11], the authors explore the capability to generate highway autonomous driving tasks with a small-sized LLM, fine-tuning a LLaMA-7B model using LoRA [14]. Given a simple three-lane highway and 10k manually annotated driving scenarios with three driving tasks, they observed that their model achieved superior accuracy compared to GPT-3.5, thus validating the effectiveness of their training approach. Authors in [12] used a 2D simulator to create 10k different driving scenarios and 160k question-answer pairs of control commands. Their work then integrates LLMs into a driving task simulation and proposes a technique to evaluate the system’s answers in an open-loop system. An interesting proposal that comes closest to the idea behind our project is presented in [13]. Here, the authors present a framework that integrates LLMs with an autonomous driving system that enables vehicles to follow user instructions provided in natural language. The idea is similar to ours in a way that it generates Python code as driving policies, “which are then executed with classical planners or controllers”. However, all hopes are pinned on the reasoning capabilities of LLMs and, despite their remarkable capabilities and “sparks of intelligence” [15], their answers cannot yet be considered provable under strictly rigorous terms. The proposed architecture consists of a system for identifying basic road elements: road boundaries and lane detection, as well as high-level semantic features such as intersections and roundabouts, including their respective entry, navigation, and exit areas.



Fig. 1: Scene interpretation error. The vehicle control system is unable to correctly recognize the intersection indicated on the map and, as a result, enters the road to the left in the wrong direction. Images extracted from [9].

III. HUMAN LIKE DRIVING

A. Proposed Scenario

This project proposes address the problem of giving navigation instructions to a self-driving system in natural language form. The challenge lies not only in understanding informal driving instructions that are intrinsically vague but also in finding a way to formalize them for effective use by the system. These instructions would allow human-like navigation, such as “turn left at the big tree” instead of “in 200 meters, turn left”, reflecting the way most people naturally think, that is, not in terms of precise distances, but rather in recognizable waypoints. This approach would also enable the system to adapt to situations where an intersection has been replaced by a roundabout. To address this challenge, this work focuses on transforming these informal instructions into an Answer Set Programming (ASP) set of rules and constraints which are combined with a complementary program representing road-driving rules. This transformation enables the system to perform cognitively-inspired navigation by interpreting the vague, natural language instructions in a way that respects road regulations and safety constraints. Although in this work we rely on a set of rules manually created using commonsense reasoning, we can foresee that such driving rules may eventually be inferred automatically by large language models LLMs, starting from traffic regulations.

We demonstrate the approach considering the following scenario. Suppose we are in a new suburban area where road constructions are rapidly changing the road graph, and therefore there is no map available in our navigation system. We are looking for a gas station and ask a passerby on the street for directions to the nearest one. We are told that we can find a gas station by “turning right after the first intersection”. After a brief conversation, we feel confident that we have understood the instructions, as we can easily imagine a scenario like the one represented in Figure 2a, and we decide to proceed. After a little driving, we find a

roundabout that the passerby did not mention. We do not despair and assume that the intersection has been replaced with the roundabout, and we change our first abstract idea to Figure 2b. So we enter the roundabout, taking the exit that more closely matches the direction of “straight ahead”. Then, after a while, we reach a second intersection where we turn right, gladly finding the gas station and putting an end to our anxiety about running out of gas. This example suggests that this kind of flexibility cannot be handled by any current navigation system and is therefore far from the capabilities required for SAE Level 5 automation. Before delving into how to solve the problem of navigating from our initial position to the gas station, we need to recognize that we are essentially facing a planning problem. Differently from well-defined problems where the searching space is already provided, the reader can imagine, for example, the problem of going from home to the workplace. In a stable, unchanging road environment, where maps are consistently updated, this is a trivial problem nowadays already solvable with vehicle navigation systems. However, we are dealing with a situation where we have no map and therefore no commercial navigation system can actually solve it. A straightforward way to tackle this problem would be to couple a scene understanding system with a standard imperative program designed with comprehensive knowledge to recognize any type of driving situation. This could involve creating an abstract control flow that is easily tailored to specific needs. Instead, we propose solving the navigation task with an ASP program, describing it in terms of logical rules that allow our system to define the result rather than the process. Similar to the imperative program approach, we still provide our system with semantic information coming from scene understanding systems. This information represents the facts that the ASP program will process to compute all the available “answer-sets”, that is, the solutions that satisfy our requirements, thereby providing high-level instructions to the low-level controller. As the vehicle then proceeds towards its destination, new

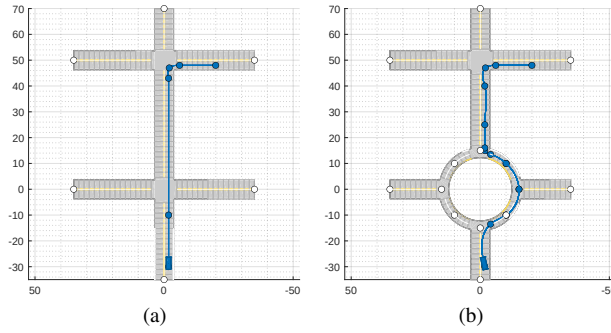


Fig. 2: Possible scenarios to reach a gas station.

information and facts will be continuously collected, and there may be the case where the new facts invalidate our previous plan. This is the core of the non-monotonic reasoning introduced with ASP, which allows us to handle possible incomplete information that evolves over time. In the following section, we discuss in more detail how we solved the problem of reaching the gas station. Although the gas station scenario is simple, it effectively illustrates the rationale behind the proposed approach.

B. Technical Details

First, although specific implementations exist for planning purposes in ASP systems, some of which refer back to STRIPS [16], here we consider the plain frontend of the DLV [17] implementation¹. Second, we have to consider how to handle the time in the DLV frontend. We model $T_i, i \in N$ discrete points in time for a vehicle moving along the road, where each time point corresponds to the introduction of one or more new facts. This is achieved through DLV arithmetic predicates, allowing us to simulate all possible scenarios and incorporate basic arithmetic into our model. Note that this temporal modeling will ultimately be replaced by the actual sensor detections. As an example, by using the DLV `#maxint` constant, the following program states that the set of valid integers is in the range $[0 \dots 3]$, and therefore the range of valid timestamps produces one answer-set containing the following four possible discrete times.

```
1 #maxint=3.
2 time(T) :- #int(T).
3 -----
4 {time(0), time(1), time(2), time(3)}
```

Listing 1:

In our sensing pipeline, it can be the case that we detect an intersection or a roundabout. Although each of these detections is associated with a **fact**, i.e. the actual input data, DLV allows us to create **predicates**,

¹<https://www.dlvsystem.it/dlvsite/dlv-download/>

which enables the use of variables (note that variables begin with an upper case character, different from the constants that begin with a lower case letter). Combined with the previously described arithmetic predicate, we can therefore let DLV find all models (answer-sets) that satisfy the following **rule**:

```
1 #maxint=1.
2 detected_intersection(T)
3   v detected_roundabout(T) :- #int(T).
```

Listing 2:

As a result, DLV can infer the four trivial answer-sets in Listing 3. Note that, as previously mentioned, when the system is in production, these facts will be generated directly from sensor data, without explicit enumeration. This enumeration is helpful here as it allows us to visualize, through DLV, all possible scenarios (answer-sets) that satisfy our plan, providing an easier way to debug the code.

```
1 {detected_roundabout(0), detected_roundabout(1)}
2 {detected_intersection(0), detected_roundabout(1)}
3 {detected_roundabout(0), detected_intersection(1)}
4 {detected_intersection(0), detected_intersection(1)}
```

Listing 3:

Notice that the rule in Listing 2 contains the disjunctive symbol “v”, which can be interpreted as “At some timestamp T , we can detect an intersection or a roundabout”. We can call this a **disjunctive rule**, which refers to a logical statement that allows for multiple alternatives. In this context, it means that the system can choose between different potential outcomes—such as detecting either an intersection or a roundabout—at a given point in time. This rule generates multiple potential models (or answer-sets), each corresponding to a possible detection at that time point. With this procedure, we can enumerate an arbitrary number of predicates, i.e. events at specific timestamps, allowing us to verify all possible scenarios that the vehicle can come across. In the final deployed version of the system, the code responsible for generating all possible scenarios, such as the code shown in Listing 1, will be replaced by the output from the software detection components.

As mentioned above, at each time T , our sensing pipeline may detect either an intersection or a roundabout. Furthermore, we also model scenarios where the detection could indicate the detection of the road itself or a default state of “nothing”. We refer to these cases as *DETECTION* predicates, representing entities that the vehicle can identify. In addition to these detections, we define a set of *ACTIONS* predicates that represent the actions the vehicle can perform. This includes predicates stating that at a specific time T_i the vehicle can be driving, crossing a generic

TABLE I: Predicate Description

	Predicates
Detected Entities	<i>detected_intersection, detected_road, detected_roundabout</i>
Actions	<i>car_parking, car_driving, cross_straight, cross_right, cross_left</i>
Car_at_Place	<i>parking_position, road, goal_point</i>

junction, or parking. Finally, the *CAR_AT_PLACE* predicates indicate the categorical position of the vehicle, specifying whether it is at a particular location.

Our code is divided into three main sections, with each section expressing:

- 1) Facts that may occur at a specific discrete time; some of these facts cannot be detected at the same time and are therefore modeled with disjunctions.
- 2) Rules and constraints that implement the common-sense navigation; apart from basic commonsense, current rules do not include any particular driving regulation, e.g. we do not consider regulated intersections with traffic lights, but we do include temporal guidelines, e.g. “*we must find first an intersection and then a roundabout*” and logic directives, e.g. we cannot detect an intersection and a roundabout at the same time. For each junction, whether it be an intersection or a roundabout, we assign an integer identifier as well as another integer representing the timestamp. All these components allow the system to create a list of driving directions.
- 3) Rules that are responsible for generating a set of verbose outputs indicating, for each step, what the planning proposes. While these rules are not required to obtain a valid answer-set, they add a layer of verbosity that makes the system’s decisions self-explanatory, providing insights into what is happening and enhancing the overall explainability of the process.
- 4) Constraints that indicate when the plan is complete; this allows for prompting only valid plans, meaning that all possible identified answer-sets must contain valid plans; this key predicate is referred to as the **goal_reached** predicate in our code.

C. Driving Rules

Without loss of generality, we start the description of our code by first making the assumption that we are currently driving the vehicle on an arbitrary road when we receive some voice instructions to proceed to a gas station. We achieve this by stating the fact that the vehicle is driving at time zero.

```
1 car_driving(0). % This is a fact
```

Listing 4:

Also, for the sake of simplicity, here we describe the code assuming that the facts are generated by means of the combinatorial process described in Section III-B. In this context, lines 2-4, 20 and 22 of Appendix A handle this generation (all lines with **#int(Time)**). The implemented driving rules are represented by a set of rules and constraints that prevents a combinatorial explosion of all possible actions that the vehicle can perform, ensuring that only valid answer-sets will be returned. Consequently, the rules and constraints are conceptually divided into different parts, depending on the associated semantics.

1) *Detected elements*: A first set of rules and constraints for the available detected elements is provided in Listing 5. Using the predicate *detected_something*, we start by adding a rule to model that something has been detected whenever an intersection, a roundabout, or road is identified. The same applies with a set of rules that summarizes all kinds of crossing actions, by abstracting them into the *cross_event* predicate. Therefore, the first part of constraints ensures that:

- it cannot be the case that the vehicle is already at the final state at time T_0 (*car_at_place(goal_point, 0)*),
- a vehicle cannot drive at time T_i (*car_driving(T_i)*) if no road is detected at time T_i (*not detected_road(T_i)*),
- a vehicle cannot cross an intersection at time T_i (*cross_event(T_i)*) if no junction is detected at time T_i (*not detected_junction(T_i)*),
- a vehicle cannot drive again after reaching its final position (goal or parking_position), (*car_driving(T_2), car_at_place(goal, T_1), $T_2 > T_1$*). The two predicates goal and parking_position are **TODO COMPLETE**.

```

1 detected_something(T, detected_intersection):-
   detected_intersection(T).
2 detected_something(T, detected_roundabout):-
   detected_roundabout(T, ID).
3 detected_something(T, detected_road):-detected_road(T
   ).
4
5 detected_junction(T):-detected_intersection(T).
6 detected_junction(T):-detected_roundabout(T, _).
7
8 cross_event:-cross_straight(T).
9 cross_event(T):-cross_right(T).
10 cross_event(T):-cross_left(T).
11 cross_event(T):-car_entering_roundabout(T, ID).
12
13 :-car_at_place(goal_point, 0).
14 :-car_driving(T), not detected_road(T).
15 :-cross_event(T), not detected_junction(T).
16 :-car_driving(T2), car_at_place(goal_point, T1), T2>
   T1.
17 :-car_driving(T2), car_at_place(parking_position, T1)
   , T2>T1.
18

```

```

19 *delete* driving(T):-car_driving(T).
20 *delete* driving(T):-cross_straight(T).
21 *delete* driving(T):-cross_right(T).
22 *delete* driving(T):-cross_left(T).
23 *delete* driving(T):-car_entering_roundabout(T, ID).

```

Listing 5:

Finally, a third set of constraints are directly derived from the driving instructions of the specific problem. In the case of the problem we are modeling, i.e. “*turning right after the first intersection*”, this statement can be translated into the following constraints:

```

1 :-#count{T : cross_straight(T)} = 0.
2 :-#count{T : cross_right(T)} = 0.
3 :-cross_straight(T2), cross_right(T1), T1<T2.

```

Listing 6:

which express the following concepts. Lines 1 and 2 correspondingly express that it cannot be the case that an answer-set contains no *cross_straight* or *cross_right* actions. This is made possible using the DLV aggregation predicate *#count* which returns the cardinality of the symbolic set to which is applied (each of the two crossing predicates, *crossing_straight* and *crossing_right*). Line 3 instead specifies that it cannot be the case that given two crossing actions at time T1 and T2, the *crossing_right* action happens before the *crossing_straight* one.

2) *Places*: Another set of constraints focuses on the specific characteristics of the detections, i.e. the *PLACES* that the sensors detect. An intuitive principle is that an intersection cannot be directly connected to another intersection, but instead a road must exist between intersections. This also allows us to avoid situations where an intersection is contained within another intersection, and the same applies to roundabouts as well as mixed scenarios. Also, we cannot detect two places at the same time. Therefore, it cannot be the case that we detect at the same time an intersection and a roundabout. Since an identifier is added to each detection, this ensures that two different junctions cannot be detected simultaneously and maintains the correct order of roundabout identifiers.

```

1 :-cross_event(T1), cross_event(T2), T2=T1+1.
2 :-detected_roundabout(T1, ID), detected_roundabout(T2, ID), T1 != T2.
3 :-detected_roundabout(T1, ID1), detected_roundabout(T2, ID2), T1 < T2, ID1 > ID2.
4 :-detected_roundabout(T1, ID1), detected_roundabout(T2, ID2), ID1 < ID2, T1 >= T2.

```

Listing 7:

3) *ACTION*: These constraints reduce the behavior of the vehicle (what the vehicle can do) when junction elements are detected. In this context, junction elements refer to road features such as intersections or roundabouts.

```

1 % cannot park before reaching the goal.
2 :- car_parking(T1), car_at_place(goal_point, T2), T1 <= T2.

```

```

3
4 % cannot drive again after reaching the goal.
5 :- car_at_place(road, T2), car_at_place(goal_point, T1), T2>T1.
6 :- car_driving(T2), car_at_place(goal_point, T1), T2>T1.
7
8 % cannot drive again after parking the car.
9 :- car_at_place(road, T2), car_at_place(parking_position, T1), T2>T1.
10 :- car_driving(T2), car_at_place(parking_position, T1), T2>T1.
11
12 % don't look at the clouds while driving (parked only after the goal)
13 :- car_at_place(parking_position, T2), car_at_place(goal_point, T1), T2<T1
14
15 % once parked, cannot detect, detectors are switched off.
16 :- car_at_place(parking_position, T1), detected_something(T2, _), T2=T1.
17
18 % cannot be two times in a row in goal_point (but can be in parked).
19 :- car_at_place(goal_point, T1), car_at_place(goal_point, T2), #succ(T1, T2).
20
21 % before parking must be at goal point.
22 :- not car_at_place(goal_point, T1), car_at_place(parking_position, T2), T1=T2-1.
23
24 % This rule checks if the goal of reaching the parking position has been reached.
25 % It evaluates whether the car is currently at the parking position at a given time.
26 % @param parking_position The position where the car should be parked.
27 % @param T The current time.
28 % return True if the car is at the parking position at time T, false otherwise.
29 goal_reached :- car_at_place(parking_position, T).
30
31 % Make sure all answer-sets contains the state goal_reached.
32 :- not goal_reached.

```

Listing 8:

In the case of any ACTION predicates, a set of rules (lines 26-30 in Appendix A) is responsible for creating a new *driving* predicate that will be used to create/update the list of steps.

4) *Explainability*: As previously mentioned, we introduced a set of additional rules to generate verbose output. The following rules create a list of predicates that simulates a step-by-step procedure, indicating the actions taken by the vehicle at each discrete time. This is particularly useful for explaining why a specific decision was made at a given time. Considering the industry’s reluctance to adopt new AI-based algorithms, mainly because of explainability issues, we strongly believe that this addition is valuable. The rules for the explainability are provided in Listing 9 and examples of the output of these rules are shown in Table II and Table III (see the STEPS box) and the screenshot in Figure 5.

```

1 step(N, driving_towards, Where) :- driving(N), car_at_place(Where, N2), N2=N+1, not car_at_place(parking_position, N2).
2 step(N, stopping, Where) :- cross_event(N), car_at_place(Where, N2), N2=N+1, not car_driving(N2).

```

```

3 step(N, start_parking, close_to_goal) :- car_at_place
   (goal_point, N), goal_reached.
4 step(N, car_stopped, goal_reached) :- car_at_place(
   parking_position, N).

```

Listing 9:

D. Experiments

The experiments were initially conducted by enumerating all feasible possibilities for arriving at a gas station following the instructions to “*turn right after the first intersection*”. This was achieved without using any real detector, instead we used the aforementioned disjunctive rules to generate a list of all feasible predicates. Since the driving instructions only provide turning directions and do not explicitly indicate the presence of an intersection or a roundabout, the number of feasible answer-sets depends mainly on the permutation with repetition of the two junction elements, which, in our case, corresponds $n^k = 4$. Of course, this number can increase depending on the number of simulated discrete points T , which would correspond to the case where a general *straight_road* is detected between the junctions.

Since DLV is used from its command line frontend and the generated answer-sets are just verbose outputs, we created a simple code to transform the verbose output into a webpage, which facilitated visualization of the solutions. The answer-set predicates are organized into four categories: i) a list of positions, indicating where the vehicle will be at each discrete time T_i ii) the corresponding driving actions at each T_i , iii) a step-by-step list showing, for each T_i , the action to be performed with respect to the final goal, and iv) a list of all detections events for each T_i , including all the simulated discrete times and the *goal_reached* predicate that indicates that the plan has been achieved. In Table II and Table III the reader can review two of the provided answer-sets, along with their parsing into the four categories webpage. This first experiment allowed us to verify that all the feasible solutions were correctly identified and that the use of DLV effectively addressed the navigation problem.

We then moved to the second goal of the HUMANAV project, that is, to verify whether informal driving rules can be translated into a set of DLV rules and constraints by the use of LLMs. Towards this goal, we investigated the use of the OpenAI ChatGPT-4o model. We provide to the model i) the actual DLV code and ii) the prompt shown in Box 1, where we provide the description of the problem together with several specific terms of DLV, which are different from generic ASP implementations, e.g. this is the case of the lack of the $!$ = operator that

must be decomposed into something else. The results of the aforementioned prompt were the following:

```

1 % The car must turn left exactly once.
2 :-#count{T : cross_left(T)}=0.
3 :-#count{T : cross_left(T)}>1.
4
5 % The car must cross straight exactly once.
6 :-#count{T : cross_straight(T)}=0.
7 :-#count{T : cross_straight(T)}>1.
8
9 % The car must turn right exactly once.
10 :-#count{T : cross_right(T)}=0.
11 :-#count{T : cross_right(T)}>1.
12
13 % The left turn must happen first.
14 :-cross_left(T1), cross_straight(T2), T1>=T2.
15 :-cross_left(T1), cross_right(T3), T1>=T3.
16
17 % The straight crossing must happen before the
18 % right turn.
19 :-cross_straight(T2), cross_right(T3), T2>=T3.

```

Listing 10: Response by ChatGPT-4o

We tested these rules by incorporating them into our DLV code, which yielded the same results as our initial proposal. As demonstrated, the rules generated by the LLM are more verbose, often breaking down complex rules and constraints into simpler atomic components with specific functions. While this approach may lack elegance, it does make the rules easier to understand, thereby facilitating a quicker debugging phase.

As a final experiment, we simulated a photorealistic driving scenario using the Carla simulator. The intention behind this experiment was twofold: on the one hand, we aimed to move from the enumeration of all feasible scenarios that can be derived from informal voice instructions to a more realistic use case, where the output of DLV is actively used to provide high-level semantical instructions for driving the vehicle while sensor detections are continuously received by the vehicle. On the other hand, we aimed to evaluate the cross-modality capabilities of the same LLM used for generating the DLV code, specifically ChatGPT-4o, by generating a textual description of the scene. For this purpose, we used the images generated by the Carla simulator and some examples are provided in Figure 3 and Figure 4, where an intersection and a roundabout are shown respectively. Both images were provided to the LLM, along with a request to describe the image while focusing on the road components of interest, specifically road intersections, roundabouts, and straight roads. The prompt, fully detailed in Box 2, includes comprehensive instructions and descriptions of each of the three road features of interest. The goal of this description was to guide the model in providing contextually relevant information while minimizing ambiguity. Given the variability of the conditions in road scenarios (for example, an intersection

Prompt for DLV code generation

The attached file defines a program in DLV (a variant of ASP, or Answer Set Programming) that models all possible actions in a driving environment scenario during a set of defined time steps (set to 4 with #maxint=4 in this specific case, but this can be changed). In a driving scenario, a car is driving on roads. The roads are connected with each other with junctions, that can be intersections or roundabouts (these are the only junctions modelled). To go from a starting point at time T1 to a goal at time T2, the vehicle has to drive. The program identifies:

- Detections: what the vehicle sees with its sensors: roads, intersections and roundabouts.
- Actions: what vehicle does, drive, cross junctions (straight, left or right) or parking.
- Places: where the vehicle is at each timestamp, on the road, at goal point, or at its parking position.

The program is intended to logically support the actions needed to go from place A to place B given a set of instructions given in a natural language, for example if someone says "to go to the gas station, go straight and then the second to the right", which in terms of driving actions this means "at the first junction go straight and then in the second junction turn right". However, junction is a term to identify where two roads merge, but it doesn't define a strict kind of junction, it can be a roundabout, or an intersection with traffic lights or something else (here the program models only intersections with traffic lights and roundabouts, so just consider these two). Therefore, the plan "to go to the gas station, go straight and then the second to the right" is just a semantic scheme that the program is modeling.

We can add a list of constraints that model all the different scenarios that can arise from the latter plan. An example of these constraints in DLV programming language are the following, please notice that the symbol % represents comments in DLV:

```
% car can't cross straight more than once.
:- cross_straight(T1), cross_straight(T2), T1 != T2.
% car can't cross right more than once.
:- cross_right(T1), cross_right(T2), T1 != T2.
% it cannot be the case that there are no cross_straight events (car never cross straight).
:- #count{T : cross_straight(T)} = 0.
% it cannot be the case that there are no cross_right events (car never cross right)
:- #count{T : cross_right(T)} = 0.
% cannot cross right before crossing straight.
:- cross_straight(T2), cross_right(T1), T1 < T2.
% it cannot be the case that the car cross to the left
:- cross_left(T).
```

Now give me a list of constraints to achieve the following instructions: **"To go to the gas station, turn left at the next junction and then go straight in the next one, finally turn right and you'll find it"**. Here some other indications and properties of DLV:

- In DLV, variables like T1, T2, etc., represent different time steps. These variables are not fixed but range over the time domain (defined by time(0..#maxint) or another domain). Each variable must be grounded (linked to a specific domain), such as with time(T1), to be safely used in constraints. When writing constraints in DLV, you should avoid using constructs like ":- time(T1), not cross_left(T1)." because this would enforce that the vehicle must turn left at every time step, which would invalidate any answer-set since "time(T1)"^a holds for all time steps. To ensure the vehicle turns left at least once, you should instead use aggregate constraints that count the number of times the vehicle turns left, like so ":- #count{T : cross_left(T)} = 0." This constraint checks whether the vehicle never turns left, which is the intended behavior.
- In DLV ":- #count{T : cross_straight(T)} != 1." is not valid. In DLV, the proper comparison operators for aggregate functions are =, !=, <, >, <=, and >=, but != isn't allowed directly in this form for aggregate functions. Instead of using !=, you can achieve the same effect using these two constraints:
 ":- #count{T : cross_straight(T)} < 1."
 ":- #count{T : cross_straight(T)} > 1."
- avoid strict constraints like ":- car_driving(T), not cross_left(T), not cross_straight(T), not cross_right(T)." because car_driving(T) will be grounded between junctions.

Finally: put all constraints in a single box so I can copy paste them.

Box 1: Prompt describing our request to ChatGPT-4o model. The response is shown in Listing 10.

may or may not have specific road markings), we found it essential to provide the clearest possible instructions to limit potential misinterpretations. The responses to each of the images are also shown in the bottom section of Box 2. As the reader can observe, ChatGPT-4o model provided confident answers for both images, capturing information not only about the road but also about contextual elements such as buildings and road markings. **CHECK IF THIS IS REALLY NEEDED, DELETE FOR EXAMPLE THE LINES FROM CARLA, MAKE SOME TESTS. NUMBERS TO SAY THAT ENUMERATING ALL THE POSSIBILITIES IS INFEASIBLE.**

IV. CONCLUSION

In this work we presented our initial experiments with the DLV Answer Set Programming (ASP) system. The aim of this project is to explore the capabilities of ASP in addressing planning problems, specifically by leveraging modern Large Language Models (LLMs) both to translate vague, informal driving instructions into a set of formal rules and constraints. These rules, when combined with a program that encodes commonsense driving rules, enables the system to generate a sequence of high-level road instructions. The advantages of this approach are threefold: first, it decouples navigation from a the reliance on a pre-recorded path over a road graph, allowing the vehicle navigation software to be self-sufficient in case of unprogrammed detours, thereby eliminating dependence on updated cartography. Second, it facilitates a more natural interaction between the system and a human supervisor. Finally, despite using black-box LLM, our system intrinsically provides a layer of explainability through the traceable reasoning contained in the generated answer-sets. A comprehensive evaluation conducted through our initial experimental activities corroborates our intuitions, paving the way for further developments.

V. FUTURE WORKS

Despite the promising results achieved, we acknowledge that this work represents only a small step in the broader effort we need to undertake. However, given the efforts of the scientific community, as reflected in similar contributions discussed in the “state of the art” section, we believe we are heading in the right direction and see significant potential for future work. While the example of reaching the gas station is quite simple, obtaining positive results here paves the way for further development that builds directly on what we have accomplished. In future work, we aim to enhance the system’s ability to handle unexpected situations, like encountering a road closure due to construction without access to an updated city map. In such cases, a human driver might

rely on simple instructions, e.g., “at the big tree, turn left”, and use their spatial awareness and intuition to find alternative routes, keeping track of turns to get back on the intended path. We want to replicate this kind of commonsense reasoning in autonomous vehicles, so they can adapt to changing environments much like humans do when they face these kind of problems in common traffic scenarios. Another possibility we will explore is using LLMs to automatically derive driving rules from country-specific traffic regulations. This could lead to adaptable and context-aware autonomous navigation systems that can interpret and apply local driving laws. By leveraging LLMs to translate legal text into actionable rules, we aim to make our system more flexible and applicable in different regulatory environments, helping it integrate smoothly wherever it is used. Furthermore, we plan to continue exploring the cross-modality capabilities of LLMs by using real images of intersections instead of simulated ones. By using real-world images, we hope to see how well these models can interpret and describe actual road scenarios, which could enhance the system’s understanding and decision-making in real driving conditions.

ACKNOWLEDGMENT

This research has been funded by the María Zambrano Grants for the attraction of international talent in Spain and the HUMANAV project granted with the Programa Propio de Investigación 2023 PIUAH23/IA-039 of the Universidad de Alcalá, Madrid, Spain.



Fig. 3: Intersection scenario simulated with Carla simulator.



Fig. 4: Roundabout scenario simulated with Carla simulator.

Prompt

The attached image is recorded from a camera on-board a vehicle. The camera is facing the road in front of the vehicle. The image can describe the following scenes:

- A straight road. This includes roads where a curve is visible, but the thing is that there is no other road intersecting the current road where the vehicle is driving. Lines on the road commonly identifies the driving lanes.
- An intersection scene. to be an intersection, more than one road have to cross, meet into a junction. It can be the case that the whole intersecting road is not visible, but always there is a presence of a piece of the road that is not the continuation of the straight road. It can be the case where traffic lights are present. Also lines painted on the road can be used to identify the intersection: these lines are different from the driving lanes, where here the concept is to identify where the vehicle has to stop before entering the intersection. It might include the word STOP, but it is not necessary. Usually this line is perpendicular to the direction of traveling (horizontal in the image).
- A roundabout: it is a special case of intersection where a the intersection area has a circle in the middle. The circle might not be completely visible from the point of view of the camera, and resemble a semi-circle. Road lines also identify this junction as in the intersection scene, however instead of a STOP here usually the road lines identifies a YIELD area. The main difference between intersections and roundabouts, apart from the rounded shape (remember that the shape can be seen from above, usually not from onboard cameras), is that no traffic lights are present.

Given these descriptions, please describe the image. It contains one of the followings: a straight road, an intersection, or a roundabout.

Response for Figure 2a

- Intersection Scene: This is an intersection as multiple roads meet. You can see traffic lights present at the junction, which is one of the defining features of an intersection. The roads are lined with crosswalk lines and markings, indicating where vehicles are supposed to stop.
- Buildings and Road Structure: The intersection is surrounded by urban buildings, with a tall building on the left side and a smaller structure on the right. The geometric structure seen at the end of the road, likely an abstract sculpture or monument, further signifies that the area may be a plaza or an important junction.
- Road Markings and Streetlights: You can see clear road markings like lane divisions, and streetlights are present along the road, contributing to the overall urban setup of the scene.
- So, this image most closely aligns with the description of an intersection scene.

Response for Figure 2b

- Roundabout Scene: The center of the road features a large circular area with a distinctive monument or sculpture, which is a key indicator of a roundabout. The monument consists of a geometric design with large blocks forming a central structure. Roads curve around this circle in multiple directions, allowing vehicles to navigate the roundabout.
- Road Layout: The roundabout is clearly marked with road lines guiding traffic around the central circle, which is typical for roundabouts. The road splits in two directions around the monument, indicating that vehicles must travel in a circular path around the central structure before exiting onto different roads.
- Surrounding Area: The area around the roundabout includes a combination of residential-style buildings in the background on the right side and taller buildings on the left. Trees and other greenery are also visible around the roundabout, adding to the sense of an urban or suburban setting.
- Thus, this image is clearly depicting a roundabout, characterized by the circular central structure and the roads that curve around it.

Box 2: Two images used with the prompt describing our request to ChatGPT-4o and the corresponding responses.

TABLE II: Answer Set 1

Position Car at Place:	Driving Action Driving Events:	Steps Step by Step:	Detections Detection Events:
car_at_place(road,0) car_at_place(road,1) car_at_place(road,2) car_at_place(goal_point,3) car_at_place(parking_position,4)	car_driving(0) cross_straight(1) car_driving(2) cross_right(3) car_parking(4)	step(0,driving_towards,road) step(1,driving_towards,road) step(2,driving_towards,goal_point) step(3,stopping,parking_position) step(3,start_parking,close_to_goal) step(4,car_stopped,goal_reached)	goal_reached time(0) time(1) time(2) time(3) time(4) detected_road(0) detected_junction(1) detected_intersection(1) detected_road(2) detected_junction(3) detected_roadabout(3,0) detected_something(1,detected_intersection) detected_something(0,detected_road) detected_something(2,detected_road) detected_something(3,detected_roundabout)

Full Answer Set: *time(0) time(1) time(2) time(3) time(4) id(0) id(1) car_driving(0) driving(0) car_at_place(road,0) car_at_place(road,1) car_at_place(road,2) car_at_place(goal_point,3) car_at_place(parking_position,4) goal_reached cross_straight(1) car_driving(2) cross_right(3) car_parking(4) detected_road(0) detected_intersection(1) detected_road(2) detected_roundabout(3,0) detected_nothing(3,1) detected_nothing(4,0) detected_nothing(4,1) cross_event(1) cross_event(3) driving(1) driving(2) driving(3) detected_something(1,detected_intersection) detected_something(0,detected_road) detected_something(2,detected_road) detected_something(3,detected_roundabout) detected_junction(1) detected_junction(3) step(4,car_stopped,goal_reached) step(0,driving_towards,road) step(1,driving_towards,road) step(2,driving_towards,goal_point) step(3,stopping,parking_position) step(3,start_parking,close_to_goal).*

TABLE III: Answer Set 2

Position Car at Place:	Driving Action Driving Events:	Steps Step by Step:	Detections Detection Events:
car_at_place(road,0) car_at_place(road,1) car_at_place(road,2) car_at_place(goal_point,3) car_at_place(parking_position,4)	car_driving(0) cross_straight(1) car_driving(2) cross_right(3) car_parking(4)	step(0,driving_towards,road) step(1,driving_towards,road) step(2,driving_towards,goal_point) step(3,stopping,parking_position) step(3,start_parking,close_to_goal) step(4,car_stopped,goal_reached)	goal_reached time(0) time(1) time(2) time(3) time(4) detected_road(0) detected_junction(1) detected_roundabout(1,1) detected_road(2) detected_junction(3) detected_intersection(3) detected_something(0,detected_road) detected_something(1,detected_roundabout) detected_something(2,detected_road) detected_something(3,detected_intersection)

Full Answer Set: *time(0) time(1) time(2) time(3) time(4) id(0) id(1) car_driving(0) driving(0) car_at_place(road,0) car_at_place(road,1) car_at_place(road,2) car_at_place(goal_point,3) car_at_place(parking_position,4) goal_reached cross_straight(1) car_driving(2) cross_right(3) car_parking(4) detected_road(0) detected_nothing(1,0) detected_roundabout(1,1) detected_road(2) detected_intersection(3) detected_nothing(4,0) detected_nothing(4,1) cross_event(1) cross_event(3) driving(1) driving(2) driving(3) detected_something(3,detected_intersection) detected_something(0,detected_road) detected_something(2,detected_road) detected_something(1,detected_roundabout) detected_junction(1) detected_junction(3) step(4,car_stopped,goal_reached) step(0,driving_towards,road) step(1,driving_towards,road) step(2,driving_towards,goal_point) step(3,stopping,parking_position) step(3,start_parking,close_to_goal).*

REFERENCES

- [1] Waymo Support, "Waymo Help Center," [Online]. Available: <https://support.google.com/waymo/answer/9775645>. [Accessed: 01-Oct-2024].
- [2] SAE, "SAE Levels of Driving Automation," 2021. [Online]. Available: <https://www.sae.org/blog/sae-j3016-update>. [Accessed: 01-Oct-2024].
- [3] European Commission, "Road safety key figures 2020," [Online]. Available: https://ec.europa.eu/transport/road_safety/sites/roadsafety/files/pdf/scoreboard_2020.pdf. [Accessed: 01-Oct-2024].

DLV executed with maxint set to 4
 Number of Answer Sets found: 8
 The number of crossing events including all possible solutions is 12
 including the following crossing events: cross_straight | cross_right

Answer Set 1			
cross_straight (detected_intersection), cross_right (detected_roundabout)			
Position Car at Place (parked road goal_point) car_at_place(road,0) car_at_place(road,1) car_at_place(road,2) car_at_place(goal_point,3) car_at_place(parking_position,4)	Driving Action (Car Driving Cross Event Car Parked) car_driving(0) cross_straight(1) car_driving(2) cross_right(3) car_parking(4)	Steps step(0,driving_towards,road) step(1,driving_towards,road) step(2,driving_towards,goal_point) step(3,stop_parking_position) step(4,start_parking_close_to_goal) step(4,car_stopped,goal_reached)	Detections goal_reached time(0) time(1) time(2) time(3) time(4) detected_intersection(1) detected_junction(1) detected_road(0) detected_road(2) detected_roundabout(3,0) detected_something(1,detected_intersection) detected_something(0,detected_road) detected_something(2,detected_road) detected_something(3,detected_roundabout) detected_junction(3)

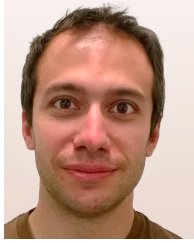
Full Answer Set:
 time(0) time(1) time(2) time(3) time(4) id(0) id(1) car_driving(0) driving(0) car_at_place(road,0) car_at_place(road,1) car_at_place(road,2) car_at_place(goal_point,3) car_at_place(parking_position,4) goal_reached cross_straight(1) car_driving(2) cross_right(3) car_parking(4) detected_road(0) detected_intersection(1) detected_road(2) detected_roundabout(3,0) detected_something(1) detected_intersection detected_something(0,detected_road) detected_something(2,detected_road) detected_something(3,detected_roundabout) detected_junction(1) detected_junction(3) step(4,car_stopped,goal_reached) step(0,driving_towards,road) step(1,driving_towards,road) step(2,driving_towards,goal_point) step(3,stop_parking_position) step(3,start_parking_close_to_goal)

Answer Set 2			
cross_straight (detected_roundabout), cross_right (detected_intersection)			
Position Car at Place (parked road goal_point) car_at_place(road,0) car_at_place(road,1) car_at_place(road,2) car_at_place(goal_point,3) car_at_place(parking_position,4)	Driving Action (Car Driving Cross Event Car Parked) car_driving(0) cross_straight(1) car_driving(2) cross_right(3) car_parking(4)	Steps step(0,driving_towards,road) step(1,driving_towards,road) step(2,driving_towards,goal_point) step(3,stop_parking_position) step(3,start_parking_close_to_goal) step(4,car_stopped,goal_reached)	Detections goal_reached time(0) time(1) time(2) time(3) time(4) detected_road(0) detected_roundabout(1,1) detected_intersection(0) detected_road(2) detected_something(3,detected_intersection) detected_something(0,detected_road) detected_something(2,detected_road) detected_something(1,detected_roundabout) detected_junction(1) detected_junction(3)

Full Answer Set:
 time(0) time(1) time(2) time(3) time(4) id(0) id(1) car_driving(0) driving(0) car_at_place(road,0) car_at_place(road,1) car_at_place(road,2) car_at_place(goal_point,3) car_at_place(parking_position,4) goal_reached cross_straight(1) car_driving(2) cross_right(3) car_parking(4) detected_road(0) detected_something(1,detected_roundabout) detected_junction(1) detected_junction(3) step(4,car_stopped,goal_reached) step(0,driving_towards,road) step(1,driving_towards,road) step(2,driving_towards,goal_point) step(3,stop_parking_position) step(3,start_parking_close_to_goal)

Fig. 5: Screenshot

- [4] European Commission, “European union annual accident report 2018,” [Online]. Available: https://ec.europa.eu/transport/road_safety/specialist/observatory/statistics/annual_accident_report_archive_en. [Accessed: 01-Oct-2024].
- [5] Dirección General de Tráfico (DGT), “Avance de las principales cifras,” 2020. [Online]. Available: https://www.dgt.es/export/sites/web-DGT/galleries/downloads/dgt-en-cifras/24h/Las-principales-cifras-2020_v6.pdf. [Accessed: 01-Oct-2024].
- [6] D. Habermann, C. E. Vido, F. S. Osório, and F. Ramos, “Road junction detection from 3D point clouds,” in *International Joint Conference on Neural Networks (IJCNN)*, Vancouver, BC, Canada, 2016.
- [7] A. L. Ballardini, D. Cattaneo, S. Fontana, and D. Sorrenti, “An online probabilistic road intersection detector,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017.
- [8] A. L. Ballardini, Á. Hernández, S. Carrasco, J. Lorenzo, I. Parra, N. Hernández, I. García, and M.A. Sotelo, “Urban intersection classification: A comparative analysis,” *Sensors*, vol. 21, p. 6269, 2021.
- [9] “FSD Beta v10.0 First Drive & Impressions — Dog, Stops, Peds Xing — 2021.24.15 FSD Beta 10,” Youtube, 11-Sep-2021. [Online]. Available: https://youtube.com/clip/UgkxS2jDErt524J7FPvSjdXU6qYVELVYyB_z [Accessed: 01 October-2024].
- [10] J. Roh, C. Paxton, A. Pronobis, A. Farhadi, and D. Fox, “Conditional driving from natural language instruction,” in *Proceedings of the Conference on Robot Learning*, 2019.
- [11] Y. Ma, C. Cui, X. Cao, W. Ye, P. Liu, J. Lu, A. Abdelraouf, R. Gupta, K. Han, A. Bera, J. M. Rehg, and Z. Wang, “LaMPilot: An open benchmark dataset for autonomous driving with language model programs,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.
- [12] L. Chen, O. Sinavski, J. Hünermann, A. Karnsund, A. J. Willmott, D. Birch, D. Maund, and J. Shotton, “Driving with LLMs: Fusing object-level vector modality for explainable autonomous driving,” in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024.
- [13] Y. Wang, Z. Huang, Q. Liu, Y. Zheng, J. Hong, J. Chen, L. Xiong, B. Gao, and H. Chen, “Drive as veteran: Fine-tuning of an onboard large language model for highway autonomous driving,” in *2024 IEEE Intelligent Vehicles Symposium (IV)*, pp. 502-508, 2024. doi: 10.1109/IV55156.2024.10588851.
- [14] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, “LoRA: Low-rank adaptation of large language models,” *arXiv preprint arXiv:2106.09685*, 2021.
- [15] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg, H. Nori, H. Palangi, M. T. Ribeiro, and Y. Zhang, “Sparks of artificial general intelligence: Early experiments with GPT-4,” *arXiv preprint arXiv:2303.12712*, 2023. [Online]. Available: <https://arxiv.org/abs/2303.12712>.
- [16] R. E. Fikes and N. J. Nilsson, “STRIPS: A new approach to the application of theorem proving to problem solving,” *Artificial Intelligence*, vol. 2, no. 3, pp. 189-208, 1971. doi: 10.1016/0004-3702(71)90010-5.
- [17] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello, “The DLV system for knowledge representation and reasoning,” *ACM Transactions on Computational Logic*, vol. 7, no. 3, pp. 499-562, Jul. 2006. doi: 10.1145/1149114.1149117.



Augusto Luis Ballardini was born in Buenos Aires, Argentina, in 1984. He completed his M.Sc. and Ph.D. degrees in Computer Science from the Università degli Studi di Milano - Bicocca, Italy, in 2012 and 2017 respectively. Following his post-doctoral activities in the IRALAB Research Group for two years, Dr. Ballardini joined the INVETT Research Group at the Universidad de Alcalá, Spain, in 2019. During his time at INVETT, he was awarded a Marie Skłodowska-Curie

Actions research grant and a research grant within the Maria Zambrano/NextGenerationEU project from the Spanish Ministry of Science, Innovation, and Universities. His research focuses on developing advanced systems for autonomous vehicle localization and data fusion, using heterogeneous data sources such as digital maps, LiDAR, and image data, combined with cutting-edge computer vision and machine learning algorithms.

APPENDIX A

DLV CODE

```

1  % Description: PLACES
2  detected_nothing(T, ID) v detected_intersection(T)
3      v detected_road(T)
4      v detected_roundabout(T, ID):-#int(T), id(ID).
5
6  detected_junction(T):-detected_intersection(T).
7  detected_junction(T):-detected_roundabout(T,_).
8
9  % This constraint avoids to detect two roundabouts at the same time.
10 :-detected_roundabout(T1, ID), detected_roundabout(T2, ID), T1 != T2.
11
12 % This rule checks if a roundabout is detected at two different times (T1 and T2) and if the
13 % ID of the first roundabout (ID1) is greater than the ID of the second roundabout (ID2).
14 :-detected_roundabout(T1, ID1), detected_roundabout(T2, ID2), T1 < T2, ID1 > ID2.
15
16 %: Ensures that a roundabout with a higher ID cannot be detected before a roundabout with a lower ID.
17 :-detected_roundabout(T1, ID1), detected_roundabout(T2, ID2), ID1 < ID2, T1 >= T2.
18
19 % Description: ACTIONS
20 car_parking(T) v car_driving(T) v cross_straight(T) v cross_right(T) v cross_left(T):-#int(T).
21 cross_right(T) v cross_left(T) v cross_straight(T):-detected_roundabout(T, _).
22 car_at_place(parking_position, Time) v car_at_place(road, Time) v car_at_place(goal_point, Time):-#int(T).
23
24 % The "driving" predicate is true for a given time point T if any of the following conditions are satisfied:
25 % "car_driving" | "cross_straight" | "cross_right" | "cross_left" predicates are true at some time T.
26 driving(T):-car_driving(T).
27 driving(T):-cross_straight(T).
28 driving(T):-cross_right(T).
29 driving(T):-cross_left(T).
30 driving(T):-car_entering_roundabout(T, ID).
31
32 % Description: This rule defines that a crossing event occurs when a straight crossing happens at a specific time.
33 % Description: This rule defines that a crossing event occurs when a right crossing happens at a specific time.
34 % Description: This rule defines that a crossing event occurs when a left crossing happens at a specific time.
35 % Description: This rule states that it is not allowed to have two crossing events occur consecutively.
36 cross_event(T):-cross_straight(T). % all kinds of crossing are a crossing_event.
37 cross_event(T):-cross_right(T). % all kinds of crossing are a crossing_event.
38 cross_event(T):-cross_left(T). % all kinds of crossing are a crossing_event.
39 cross_event(T):-car_entering_roundabout(T, ID). % all kinds of crossing are a crossing_event.
40
41 % CONSTRAINTS.
42
43 :-cross_event(T1), cross_event(T2), T2=T1+1. % NEW_RULE: cannot cross two times in a row.
44
45 :-car_at_place(goal_point, 0). % goal is not at step zero.
46 :-car_driving(T), not detected_road(T). % cannot drive if no road is detected.
47 :-cross_event(T), not detected_junction(T). % cannot cross intersection if no intersection is detected.
48 :-cross_event(T), detected_road(T). % cannot cross intersection if there is a road.
49
50 :-car_parking(T1), car_at_place(goal_point, T2), T1<=T2. % cannot park before reaching the goal.
51
52 :-car_at_place(road, T2), car_at_place(goal_point, T1), T2>T1. % cannot drive again after reaching the goal.
53 :-car_driving(T2), car_at_place(goal_point, T1), T2>T1. % cannot drive again after reaching the goal.
54 :-car_at_place(road, T2), car_at_place(parking_position, T1), T2>T1. % cannot drive again after parking the car.
55 :-car_driving(T2), car_at_place(parking_position, T1), T2>T1. % cannot drive again after parking the car.
56
57 :-car_at_place(parking_position, T2), car_at_place(goal_point, T1), T2<T1. % Don't look at the clouds while driving,
58 % parked only after the goal.
59 :-car_at_place(parking_position, T1), detected_something(T2, _), T2=T1. % once parked, cannot detect, detectors
60 % are switched off.
61
62 %:-car_driving(T2), car_at_place(goal_point, T1), #succ(T1, T2). % OPTIONAL: after reaching the goal, stay
63 % there, no drive/parked.
64
65 :-car_at_place(goal_point, T1), car_at_place(goal_point, T2), #succ(T1, T2). % cannot be two times in a row in
66 % goal_point (but can be in parked).
67
68 :-not car_at_place(goal_point, T1), car_at_place(parking_position, T2), T1=T2-1. % before parking must be at goal point.
69
70 % This rule checks if the goal of reaching the parking position has been reached.
71 goal_reached:-car_at_place(parking_position, T).
72
73 % If the goal_reached predicate is not true, the rule is satisfied.
74 :-not goal_reached.
75
76 % OTHER RULES
77 step(N, driving_towards, Where):-driving(N), car_at_place(Where, N2), N2=N+1, not car_at_place(parking_position, N2).

```

```
78 step(N, stopping, Where):-cross_event(N), car_at_place(Where, N2), N2=N+1, not car_driving(N2).
79 step(N, start_parking, close_to_goal):-car_at_place(goal_point, N), goal_reached.
80 step(N, car_stopped, goal_reached):-car_at_place(parking_position, N).
```

The code is available at <https://github.com/invett/HUMANAV>.