Journal of
Sensor and Actuator Networks

MDPI

Article

Hybrid Verification Technique for Decision-Making of Self-Driving Vehicles

Mohammed Al-Nuaimi [1,*], Sapto Wibowo [2], Hongyang Qu [1], Jonathan Aitken [1] and Sandor Veres [1]

[1] Department of Automatic Control and Systems Engineering, The University of Sheffield, Sheffield S10 2TN, UK; h.qu@sheffield.ac.uk (H.Q.); jonathan.aitken@sheffield.ac.uk (J.A.); s.veres@sheffield.ac.uk (S.V.)
[2] Department of Electronic Engineering, State Polytechnic of Malang, Jawa Timur 65141, Indonesia; sapto.wibowo@polinema.ac.id
* Correspondence: m.al-nuaimi@sheffield.ac.uk

Abstract: The evolution of driving technology has recently progressed from active safety features and ADAS systems to fully sensor-guided autonomous driving. Bringing such a vehicle to market requires not only simulation and testing but formal verification to account for all possible traffic scenarios. A new verification approach, which combines the use of two well-known model checkers: model checker for multi-agent systems (MCMAS) and probabilistic model checker (PRISM), is presented for this purpose. The overall structure of our autonomous vehicle (AV) system consists of: (1) A perception system of sensors that feeds data into (2) a rational agent (RA) based on a belief–desire–intention (BDI) architecture, which uses a model of the environment and is connected to the RA for verification of decision-making, and (3) a feedback control systems for following a self-planned path. MCMAS is used to check the consistency and stability of the BDI agent logic during design-time. PRISM is used to provide the RA with the probability of success while it decides to take action during run-time operation. This allows the RA to select movements of the highest probability of success from several generated alternatives. This framework has been tested on a new AV software platform built using the robot operating system (ROS) and virtual reality (VR) Gazebo Simulator. It also includes a parking lot scenario to test the feasibility of this approach in a realistic environment. A practical implementation of the AV system was also carried out on the experimental testbed.

Keywords: self-driving vehicle; formal verification; model checking; rational agent; decision-making; ROS

Citation: Al-Nuaimi, M.; Wibowo, S.; Qu, H.; Aitken, J.; Veres, S. Hybrid Verification Technique for Decision-Making of Self-Driving Vehicles. *J. Sens. Actuator Netw.* **2021**, *10*, 42. https://doi.org/10.3390/jsan10030042

Academic Editors: Lei Shu, Rafael C. Cardoso, Angelo Ferrando, Daniela Briola, Claudio Menghi and Tobias Ahlbrecht

Received: 17 March 2021
Accepted: 23 June 2021
Published: 29 June 2021

1. Introduction

The Defense Advanced Research Projects Agency (DARPA) sponsored competitions between 2004–2007 [1,2] presented new results on autonomous ground vehicles that showed large steps forward in the field. However, the results still primarily address non-complex driving environments [3]. AVs, which operate in complex environments, require methods that can handle unpredictable circumstances and reason in a timely manner in complex urban situations, where informed decisions require accurate perception.

The development and deployment of AVs on some of our roads are not only realistic but can also bring significant benefits. In particular, they promise to solve various problems related to: (i) the improvement of traffic congestion, (ii) the reduction of the number of accidents (iii) automate the parking operation including looking for a free parking space, and (iv) encourage shared use of AVs to reduce overall fuel consumption [4]. Studies show that more than 90% of all car accidents are caused by human errors and only 2% by vehicle failures [5].

Considerable research and development resources are spent in industry and academia on hardware and algorithms, which cover different challenges such as perception, planning, and controls. Decision-making while driving is a vital process that needs special attention.

The primary cause of human accidents comes from incorrect decisions, and there will be limited benefits in developing AVs that continue to make those incorrect decisions at a similar rate to humans. Hence, we need to make sure that any decision the vehicle is going to take has been thoroughly verified.

AVs depend on many sensors to find their way among static and dynamic obstacles; each of those sensors has strengths and weaknesses. Cameras and LiDARs are usually used together in perception systems to provide a high level of certainty. LiDAR often provides excellent odometry, localization, mapping, and range information but with a limit to object identification. Cameras provide better recognition but with limits to localization accuracy [6]. A multi-sensor system can provide reliable information for perception in joined-up software architecture for timely processing of the sensory data in the context of localization and mapping, planning, dynamic obstacle detection, and avoidance [7].

Intelligent software agents have been in development for the past two decades. Some well-known agent types are reactive, deliberative, multi-layered, and belief–desire–intention (BDI) agents [8,9]. The limited instruction set agent (LISA) [10] is a new multi-layered approach to rational agents based on the BDI agent architecture, which is particularly suitable for achieving goals by autonomous systems.

With the increasing demand for machine learning techniques and advanced planning and decision-making methods, verification and guaranteed performance of the autonomous driving process has become a challenging problem. Reconfigurable and adaptive RA-based control systems are capable of controlling a vehicle in a trajectory to avoid other vehicles and people [11]. Integration is essential to enable decision-making based on behavior rules and experience in order to make decisions with foresight and consideration to other traffic participants. RAs have demonstrated significant robustness in the implementation of various applications. However, for real-world critical applications, some safety concerns can still be raised even after extensive testing, creating the need for an appropriate verification framework. It is important to note that validation and verification usually needs to be performed together to check the system. However, this paper focuses on a new verification framework for the safety of autonomous vehicles.

The testing of systems through prototype development only answers some of the components of operational safety questions. The best that can be achieved in testing is to use a representative set of scenarios on real vehicles. Simulations can provide illustrations of the correct dynamic and social behavior of the AV. However, it is difficult to take into account rare combinations of events that may arise during the run-time of the autonomous system. It is unlikely that the designer will think of all potential scenarios to ensure complete coverage. Formal verification methods try to answer the rest of the questions by accounting for all the probabilities for a given scenario [12,13]. If accurate dynamical models are available to represent robotic skills of sensing and action, then formal verification can rely on a finite interaction model of the vehicle with a bounded model of the environment, that is based on known characteristics of traffic participants.

This paper describes a novel method for the verification of the decision-making system of an AV with a proposed architecture that lends itself to verification. We take into account the computer-based system consisting of AV design and simulation for the new verification platform. Safety and ease of implementation of the system are the two central themes in this paper, with the prime focus on the safety aspect. This paper presents a prototype system of an AV parking lot scenario with the ability to deal with the most vulnerable traffic participants: vehicles and pedestrians. In general, the level of autonomy of a vehicle can vary from fully human-operated (level 0) to a fully autonomous vehicle (level 5). Our vehicle is designed to work at level 4, where it can work autonomously in a restricted environment until it is interrupted [14].

The architecture of our proposed perception system is divided into four subsystems: LiDAR-based, vision-based, tracking-classification, and coordinate transformation. The perception system is used for localization and mapping, including calculating the relative positions of objects around the AV. The cameras are responsible for object recognition and

detection of free parking spaces, with the aid of the LiDAR to provide an occupancy grid. The position of the objects is converted to the camera coordinate system, defining a region of interest (ROI) in the image space, then it obtains the depth information that belongs to that object from the LiDAR point cloud.

Most autonomous robotic agents use logic-based inference to keep themselves safe and within permitted behavior by providing the basis of reasoning for a robot's behavior [15]. Given a set of rules, it is essential that the robot can establish the consistency between its rules, its perception-based beliefs, its planned actions, and their consequences. In this paper, we are concerned with the high-level software components responsible for decisions in an AV capable of navigation, obstacle detection and avoidance, and autonomous parking. These logic-based decisions can either be implemented through a rational agent [9,10,16–19] or through fuzzy logic [20–22] depending on the level of performance guarantee required.

To achieve this, we have established the following stages. First, we have built an AV system and its environment in ROS [23] and the Gazebo Simulator [24]. Second, we investigated how a robotic agent can use model checking through the use of the MCMAS model checker [25] to examine the consistency and stability of its rules, beliefs, and actions through computational tree logic (CTL) for the RA that has been implemented within the LISA agent programming framework [10,26]. Third, we have formally specified some of the required RA properties through probabilistic timed programs (PTPs) and probabilistic computation tree logic (PCTL) formula, which are then formally verified with the PRISM Model checker [27] during run-time operation of the AV. Finally, within the proposed verification framework, which comprises both MCMAS and PRISM verification tools, we have obtained formal verification of our AV agent for some specific behaviors.

We used Gazebo Simulator in this work because of its full compatibility with ROS, and the huge support from the robotics community. PTP is a formalism for modeling systems whose behavior incorporates both probabilistic and real-time characteristics. In PTP, the location/space is discrete, while time is continuous. It is a good compromise between computational complexity and accurate mathematical modeling. Efficient verification algorithms have been developed to verify PTPs.

The development and deployment of these autonomous vehicles will rely on their situational awareness [28–32]. The vehicles will be required to co-exist alongside vehicle controlled by humans and this presents a significant problem. Whilst simulation can be used to explore edge-cases and boundaries of operation this relies on the imagination of the designer of these systems. Therefore vehicles could look to learn and adapt to situations to improve their awareness and performance. The application of this situation-based learning is out of scope for this paper but provides motivation for future work.

Contribution

This work is a continuation of our previous work [33,34] to present a new and complete verification framework for the decision-making of an AV that combines both the design-time and run-time verification. The main contribution can be summarized as follows:

1. New verification framework for decision-making of a self-driving vehicle that merges design-time verification represented by the MCMAS model checker and the run-time verification represented by the PRISM probabilistic model checker, which provide a comprehensive approach for the verification of AV's agent decisions.
2. Design, simulation, and implementation of an AV through ROS open-source physics-based system for a Tata Ace vehicle. Both the AVs in simulation and experimental implementation use the same perception, rational agent, planning, and control system software designed for a parking lot environment.

2. Related Work

Autonomous vehicles have been a major area of research interest for the research community since the DARPA Grand Challenge, which inspired the development of many AV testbeds across the industry and academia. An example is the Stanford's Junior [35],

which provides a testbed with multiple sensors for planning and recognition. It is capable of dynamic object detection and tracking and also localization. Other examples are Talos from MIT [36], and Boss from CMU [37], among many others.

In this section, we discuss some recent platforms and techniques related to our work, developed for safe self-driving vehicle operation.

In ref. [38], the authors presented a testbed called cognitive and autonomous test (CAT) vehicle, which is comprised of a simulation-based self-driving vehicle, with a straightforward transition to hardware-in-the-loop testing and execution, to support research in autonomous driving technology. The idea is to support researchers who want to demonstrate new results on self-driving vehicles but do not have an access to a physical platform to mimic the dynamics of a real vehicle in the simulation and then provide a seamless transition to the reproduction of use cases with hardware. The Gazebo Simulator utilizes ROS with a physics-based vehicle model, including simulated sensors and actuators. Gazebo comes as a default simulator with ROS and is a physics-based simulator that has also been used in our work. Gazebo is not the only option that is available to design and test self-driving vehicles. Other simulators include CARLA [39], which has been developed to support the development, training, and validation of autonomous urban driving systems. CARLA is also compatible with ROS and supports flexible specification of sensor suites and environmental conditions. Another simulator that is also useful to develop and test self-driving vehicles is LGSVL [40], which is a multi-robot AV simulator. It has been designed as an open-source simulator based on the Unity game engine to test autonomous vehicle algorithms. LGSVL also supports ROS where it helps to connect the simulator to a physical platform for tests.

Self-driving vehicles use a perception system to perceive the environment. Sensor fusion is used to bring together inputs from multiple radars, LiDARs, and cameras to form a single model or image of the environment around a vehicle. The resulting model is more accurate because it balances the strengths of the different sensors. In ref. [41], the researchers developed a perception fusion architecture based on the evidential framework to solve the detection and tracking of moving objects problem by integrating the composite representation and uncertainty management. They tested their fusion approach with a physical testbed from the interactive IP European project, which includes three main sensors: camera, LiDAR, and radar by using real data from different driving scenarios and focusing on four objects of interest: pedestrian, bike, car, and truck. The sensor fusion provides necessary information for different parts of the autonomous driving system, such as simultaneous localization and mapping (SLAM) and planning, which include both path planning and motion planning.

Other methods in the literature include the distance sensor-based parking assistance system, which recognizes an empty space using ultrasonic and LiDAR sensors as explained in refs. [42–44]. The problem with this system is that it will recognize a free space as a parking slot when the space is equal to the width of the vehicle is detected, even if the space is not a parking slot. This method is usually applied to a parking assistance system where the driver can determine a parking space. However, it is not compatible with a fully autonomous parking system, where the system judges a parking space and moves the car.

The around view monitoring (AVM) [45] can compensate for the disadvantages of distance-sensor-based detection as it can detect parking spaces based on parking slot lines instead of empty spaces. However, a false-positive (FP) can be detected from shadows and 3D objects, or the parking slot lines may be occluded by a nearby vehicle. Hence, the researchers proposed a probabilistic occupancy filter to detect parking slot lines. This filter uses a series of AVM images and onboard sensors to improve the occlusion problem and reduce the false-positive from other objects. However, this method still not very accurate and could mislead the AV in some cases.

The main topic we discuss in this paper is the verification of decision-making for our self-driving vehicle. This area of research has received more attention in the last few years as the complexity of the autonomous software has increased while the safety and feasibility

of these decisions have been under investigation due to series of fatal incidents that occurred with these autonomous systems as listed in ref. [46]. In ref. [19], the authors show how formal verification can contribute to the analysis of these new self-driving vehicles. An overall representation for vehicle platooning is a multi-agent system implemented within the GWENDOLEN agent programming language in which each agent captures the "autonomous decisions" carried out by each vehicle. They used formal verification to ensure that these autonomous decision-making agents in vehicle platoons never violate any safety requirements. The authors presented a method to verify both the agent behavior using Agent Java PathFinder (AJPF) and the real-time requirement of the system using the Uppaal model checker where the system is represented as timed automata.

In ref. [47], Fernandes et al. modeled an AV with a rational agent for decision-making. To achieve this, they have established the following stages. First, the agent plans and actions have been implemented within the GWENDOLEN agent programming language. Second, they have built a simulated automotive environment in the Java language. Third, they have formally specified some of the required agent properties through LTL formulae, which are then formally verified with the AJPF verification tool. Finally, within the model checking agent programming language (MCAPL) framework they have obtained formal verification of the AV agent in terms of its specific behaviors and plans.

In ref. [48], Giaquinta et al. presented probabilistic models for autonomous agent search and retrieve missions derived from Simulink models for an unmanned aerial vehicle (UAV) and they show how probabilistic model checking using PRISM model checker has been used for optimal controller generation. They introduced a sequence of scenarios relevant to UAVs and other autonomous agents such as underwater and ground vehicles. For each scenario, they demonstrated how it can be modeled using the PRISM language, give model checking statistics and present the synthesized optimal controllers.

In our work, our system is different in the following aspects: we focused on adapting and developing different techniques and methods that contribute towards the design of a safe self-driving operation. We used ROS to design the main system functions such as the perception and control subsystems. We used a similar method presented in ref. [38] where the system built-in ROS supports hardware-in-the-loop. The difference is that our ROS system was designed to satisfy the needs for our testbed—the TATA ACE electric vehicle. Further, it provides additional functions to connect to the main decision-maker onboard and the verification system. This represents a modular overall system that supports adding more subsystems when needed.

For the perception system, we used a similar set of sensors usually used by others, where this is represented by a stereo camera, mono-cameras, and LiDAR. The combination of this set provides sufficient data to perceive the vehicle's surroundings. The perception system is presented in Section 4.

We tested our system for autonomous parking scenarios. The AV needs to look for the attached Aruco markers on each parking slot; this method is used for its simplicity, reliability, and compatibility with the fully autonomous driving mode compared with other methods mentioned in the literature. However, this method needs to be supported in the parking lot by installing Aruco markers on some or all of the parking slots to be used by AVs.

As we mentioned, this work focused on the verification of the decision-making of AVs. The novelty of our work comes from the fact that we tried to thoroughly verify the reasoner and the decisions from multiple aspects to make sure that any decision that could be made is safe to apply. We applied the verification for the reasoner offline during the design-time and online during the run-time operation.

The reasoner software has been designed by natural language programming using software called sEnglish, as explained in Section 5. This method is used for its simplicity and compatibility with the ROS system and the verification tools. However, this method comes with some limitations and it is difficult to be used with a higher-level autonomous system presented in level five autonomy.

For the design-time verification, we used the MCMAS model checker to check the consistency and stability of the logic predicates. The PRISM model checker is used to verify the decisions made by the agent during run-time operation. Details of this method are explained in Sections 6 and 7.

3. System Overview

A standard AV has a control architecture that incorporates both low-level and high-level components. The low-level components include sensors and actuators, while the high-level system components are often responsible for decision-making based on data provided by low-level components.

Our perception system provides a stream of images and 3D point cloud data obtained from sensors commonly used in AVs. It consists of eight mono-cameras (three on each side and two at the back), a stereo camera on the front, and a LiDAR on top that can be shifted left and right and tilted with a specific angle by the high-level system for better coverage. The stereo camera in front of our vehicle uses a deep-learning-based object detector that is capable of detecting different objects, including those that could exist in a parking lot environment. The perception system can also localize free parking spaces by using fiducial markers. These data are converted to high-level abstract statements that can be used by the RA onboard the vehicle. A case study of a parking lot scenario has been carried out to demonstrate the verification methods and to show the feasibility of our approach.

The AV system in Figure 1 is based on a modular design that makes practical implementations relatively simple and allows for future updates. The decision agent is central to the system design. We used the LISA agent paradigm due to its capability to execute actions based on decision-making to pursue goals while also not being too complicated to enable verification. The decision process also uses rules and abstractions from future predictions (consequences of future events) and can re-plan the path of the AV when needed.

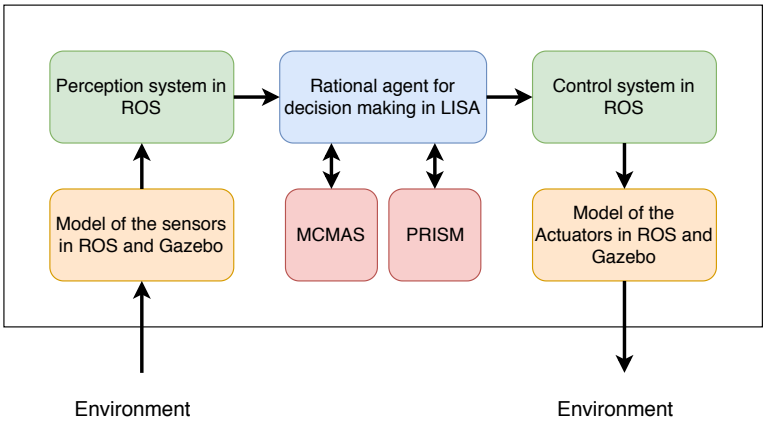

Figure 1. Block diagram showing the different components of the autonomous vehicle system in simulation. Blocks in green and yellow represent the sensing and actuating systems, the block in blue represents the rational agent that communicate with the verification system (MCMAS and PRISM).

The rational agent (RA) is capable of communicating with the perception system to sense the environment and instruct the actuators to move the vehicle in a collision-free path without the need for human support. To achieve this, the perception system builds a model of the environment, localizes objects around, and keeps updating its model after each perception cycle. The software agent has rule-based reasoning, planning capability, and some feedback control skills for steering and velocity regulation. The RA has been implemented using natural language programming (NLP) in sEnglish [17], as mentioned in Section 5.

The vehicle in simulation supports a scalable, modular design to ease the implementation of different system parts and further development. The physics-engine-based

simulation shown in Figure 2 consists of a model of the Tata Ace electric vehicle shown in Figure 3, with the same specifications and parameters for the vehicle and sensors.

The AV is based on packages designed with ROS, using Python and C++. ROS provides tools and libraries for writing perception and control algorithms and other applications for AVs. With various levels of hardware and software abstractions, device drivers for a seamless interface of sensors, libraries for simulating sensors, and a visualizer for diagnostics purposes, ROS provides middleware and interoperability to simulation software, and the software installation is straightforward. Being a distributed computing environment, it implicitly handles all the communication protocols. The hardware used in this work has been selected through experimental tests, a similar set has been widely adopted by other prototypes design of AVs.

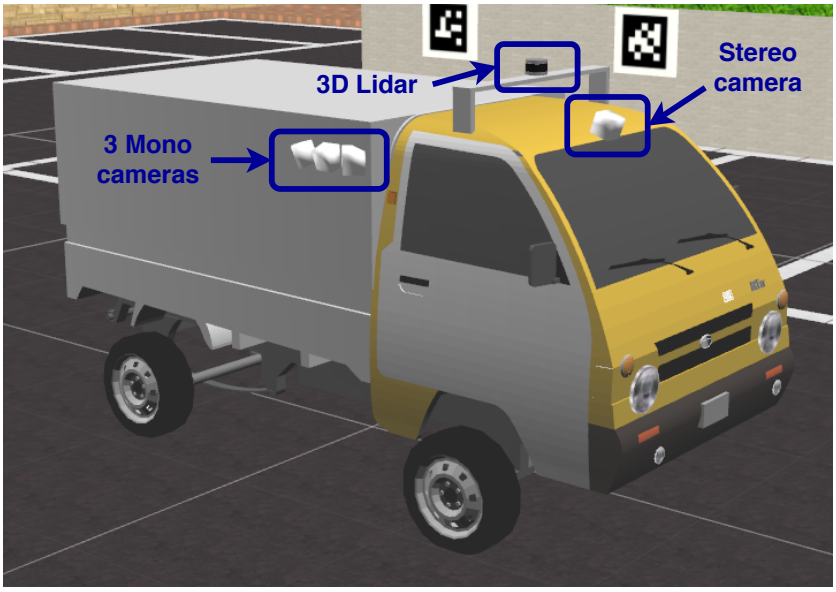

Figure 2. The test vehicle we designed in ROS and Gazebo showing sensor configuration.

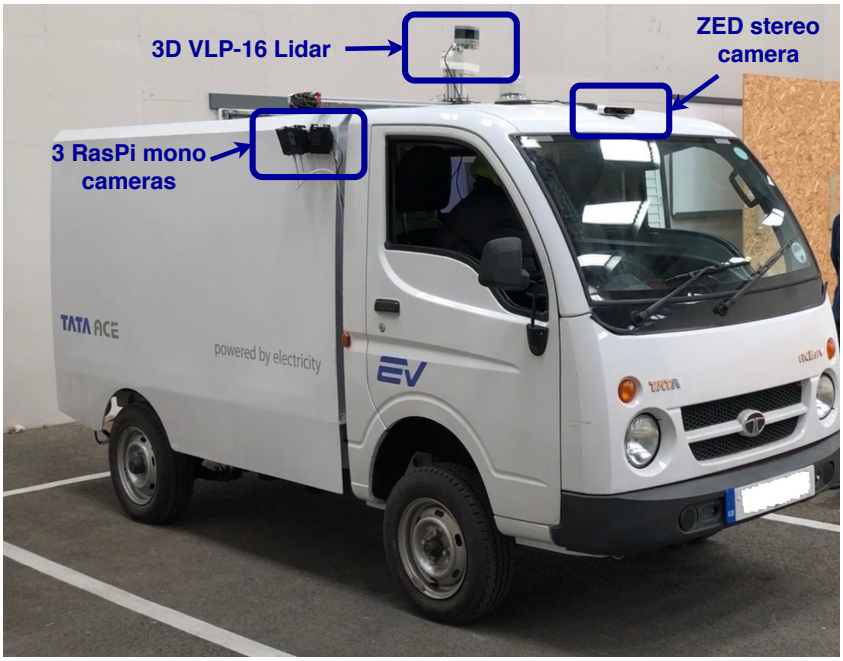

Figure 3. Our electric testbed showing the hardware and sensor configuration.

However, standard ROS packages lack domain-specific requirements for experimentation with a car-like robot. A typical setup of an AV consists of a controller and a set of sensors tested and mounted to provide sensing modality that provide a complete view of the environment around the test vehicle. In order to control motion seamlessly, we created interfaces for control and consistent consumption of sensor data. We had tested the current state of the vehicle and issued control signals well before the real platform was engaged. Once algorithms were tested in simulation, they could be implemented in the real vehicle, and the physical platform was then replaced the AV in simulation. The simulated version was used as a proving ground for the algorithms, to build confidence in their operation before transferring to the, naturally more complex, physical system.

We created models of the AV, parking lot, pedestrians, and other vehicles in the parking lot mainly using the SkechUp software [49] to create 3D models recognized by Gazebo Simulator.

In this work, we are interested in both design-time and run-time verification; this process involves the analysis of the system to detect behaviors violating the required properties. Design-time architecture verification is performed using MCMAS and probabilistic run-time verification using PRISM. We have programmed a compiler from LISA to build the models for MCMAS. The latter was then used to check the consistency and stability of beliefs, rules, and actions of the AV in its environment. When the logic predicates are inconsistent or unstable, a counterexample is generated to demonstrate the violation and help developers to correct the system [50]. PRISM is used by the RA at run-time to ask questions such as 'what is the probability of success of the current action' or 'what is the probability of achieving the current goal within a time limit' [27], the parameters used to estimate the probabilities depending on the driving scenario were, for example, the speed of the AV, speed of moving objects, and the direction of movements. For example, the agent can ask what is the probability of success if the AV were to move to a specific location within a specific period, taking into account the dynamic models (generated by the agent) of other objects moving around.

Driving in urban environments is characterized by uncertainty over the intentions and behavior of other traffic participants, which is usually considered in the behavioral layer responsible for decision-making using probabilistic planning formalisms, such as Markov decision processes (MDPs) to formulate the decision-making problem in a probabilistic framework. We used a different approach in probabilistic systems represented by probabilistic timed programs (PTPs) [51] to model the behavior of the AV and the proposed behavior of the other participants. A detailed explanation is presented in the following sections.

4. Design and Implementation of Self-Driving Vehicle

The hierarchical system is decomposed into four components, as shown in Figure 4: The perception system is used to receive information about the environment and feeds this information to the second stage. Here the agent makes decisions on the suitable progress of the car towards the destination by rules of interaction and rules of the road. The next stages are the global path planner and the local path planner, which are responsible for generating the path of the AV from the starting point to its destination based on the directions and speed profile set by the RA, then select a continuous motion plan through the environment to achieve a local navigation task. The last component is the control system that executes the motion using actuators and corrects errors in the execution in a feedback loop. In the remainder of the section, we discuss each of these components briefly.

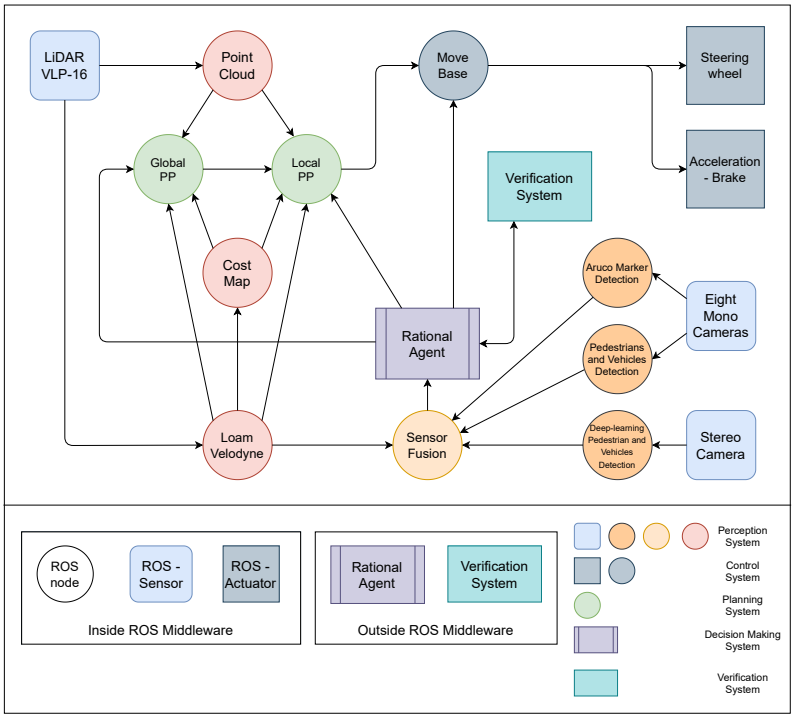

Figure 4. Our AV system showing the main nodes designed in ROS for perception, planning, and control (secondary and supporting nodes are not shown here); RA was designed in sEnglish [17,52], whereas the verification system was designed in MCMAS and PRISM verification tools.

4.1. Perception System

Modern vision-based detection techniques work by extracting image features to segment regions of interest (ROI) then detect different objects within those regions. In particular, the detection of people and vehicles has made significant progress in the autonomous and assisted driving areas [53,54]. Radar is a robust and invaluable information source for perceptual tasks; however, the spatial resolution of radar is typically poor compared to camera and LiDAR. Thus, much recent perception research is focused toward cameras and LiDAR. We have designed a parking lot scenario in ROS and Gazebo to explain the use of different sensors as shown in Figure 5.

Detection methods based on mono-cameras suffer in two ways: despite the methods proposed for moving mono-cameras, fast and accurate range measurement remains an issue, which is vital for critical object detection in autonomous driving applications. Optical sensors can suffer from a limited field of view and poor operation during low lighting conditions. On the other hand, LiDAR is usually paired with the advanced driver assistance system (ADAS) applications and has become part of the AV perception system because of the high precision range measurements and the wide field of view that it provides. The main issue for the LiDAR-based system is that the data from scans do not contain information that easily allows different objects to be distinguished between, especially in a dense environment.

Stereo cameras can provide more precise depth data and a wider angle compared with mono-cameras. However, the detection angles are smaller than LiDAR, and it is also less precise in providing depth information, especially over the long distances that are often vital for AV decision-making. The integration of cameras and LiDAR sensors can enhance fast object detection and recognition performance [41]. This type of sensor fusion system is known as the classic LiDAR-camera fusion system.

In this simulation-based system, we tried to mimic our approach for the experimental AV system, where we used a Velodyne VLP-16 LiDAR, one ZED stereo camera, and eight Raspberry Pi 3 model B mono-cameras (8 megapixels each), The properties for these sensors are mentioned in Table 1. The LiDAR is connected directly to ROS for point cloud

data processing. The front-facing stereo camera is connected to the Jetson TX2 running YOLOv3 [55] deep-learning-based object detection. The mono-cameras use the processing power of their host Raspberry Pi system for aggregated channel features (ACFs) object detector of pedestrians and vehicles [56]. Those mono-cameras, along with the stereo-camera, cover a 360° FOV. The camera system has also been equipped with a method for fiducial-follow that uses Aruco markers to detect the location and orientation of free parking slots, as shown in Figure 6 (right camera one and two). Along with the occupancy grid data generated by the LiDAR, the AV is capable of detecting free parking spaces simply and efficiently. Figure 6 also shows the detection of vehicles and pedestrians, which is a vital process for normal operation of the AV.

When a known object is detected by one of the cameras, the associated LiDAR measurements are processed for the distance calculation by matching the location of the detected object with the 3D point cloud data belonging to the same object. Based on the generated depth map, the position and direction of the object are calculated from the ROI, those measurements from LiDAR are calculated according to the coordinates transformation.

We used the LiDAR odometry and mapping (LOAM) [57] ROS package for Velodyne VLP-16 3D LiDAR. This package provides a real-time method for mapping and state estimation by applying two parallel threads: The odometry thread to measure (at a higher frame rate) the motion of the LiDAR between two movements and to eliminates distortion in the point cloud. The second is a mapping thread that incrementally builds the map (at a lower frame rate) based on the undistorted point cloud, and also to compute the pose of the LiDAR on the map.

Table 1. Properties of sensors for both simulation and real testbed.

Sensor Type	No. of Sensors	Resolution	No. of Frames/Speed of Rotation
LiDAR	1	3D 16-layer (up to 50 m)-Simulation 3D 16-layer (up to 100 m)-Real testbed 360°H/30°V 1864 PPS-Simulation 300,000 PPS-Real testbed	300 RPM-Simulation 600 RPM-Real testbed
Stereo camera	1	Color 1344 × 376	10 FPS
Mono camera	8	Color 640 × 480	6 FPS

Figure 7 shows the map built for the AV current path in the parking lot shown in Figure 5. The sides of the objects that are facing the LiDAR are shown on the map with white lines. We added another layer of protection using a cost map function, which helps the AV to keep an extra safe distance from any object within a specific inflation distance where this could be set according to the environment type; it is represented on the map in Figure 7 with the blue lines surrounding the white lines. Finally, the data for the detected objects and their locations are sent to the RA for further processing.

Figure 5. Parking lot scenario developed in ROS and Gazebo Simulator to check the proposed system. The AV is looking for a free parking space in the parking lot and it is navigating among pedestrians and other vehicles depending on the data coming from the perception system and analyzed with the rational agent onboard the AV. The information obtained from the perception system is shown below in Figures 6 and 7.

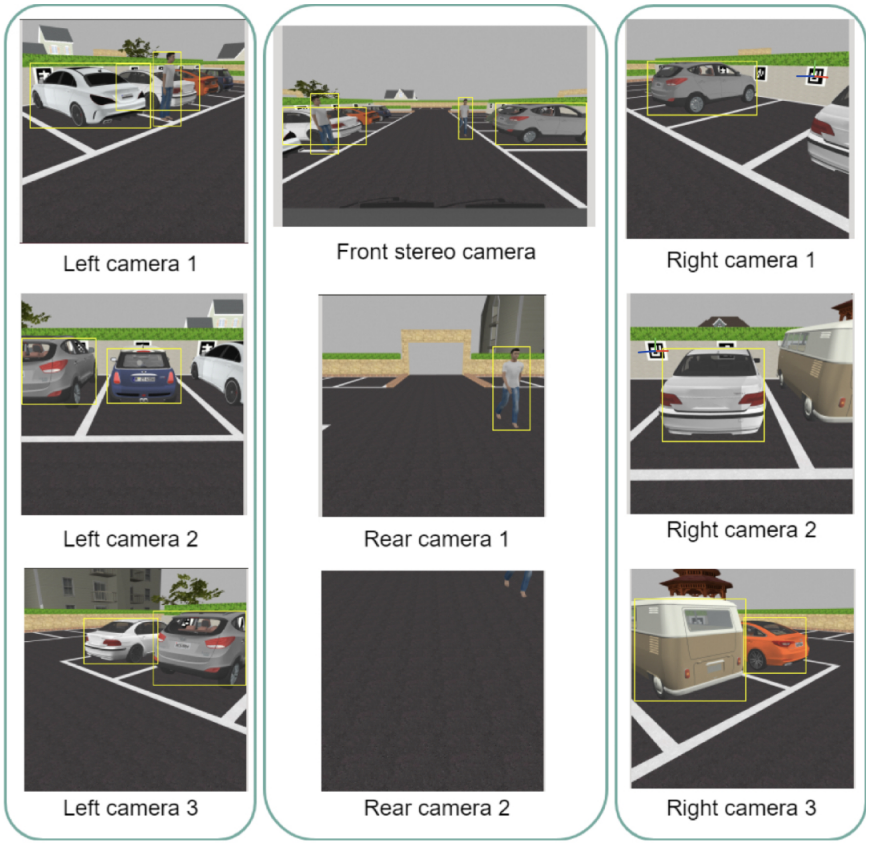

Figure 6. Pedestrians and cars recognized by the AV using camera sensors. "Right camera 1" and "Right camera 2" show the Aruco marker detection attached to the parking spaces. This can be combined with the occupancy grid generated by the LiDAR in Figure 7 to detect the free parking spaces.

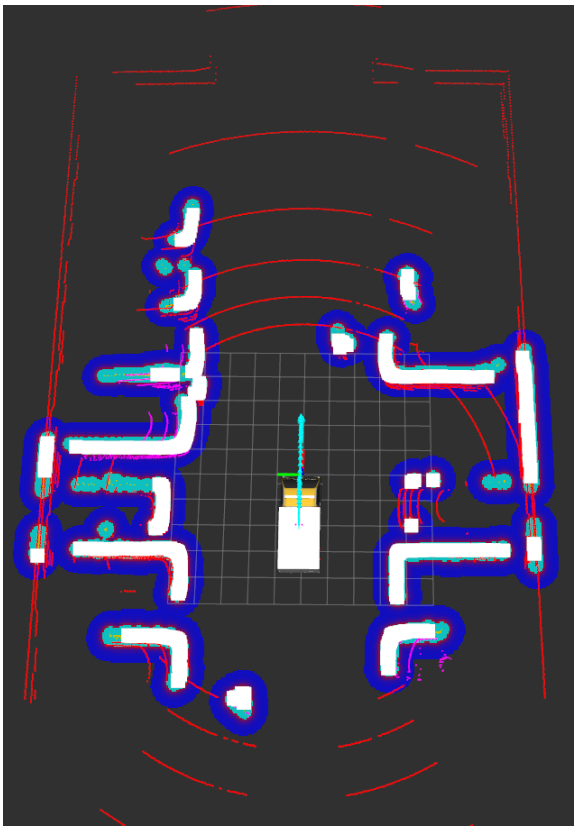

Figure 7. Data from the LiDAR sensor placed on the top of the AV. The odometry and mapping data are shown for the parking scenario; this is based on LOAM Velodyne ROS package shown in Figure 4. The pedestrians and the parked vehicles around have been detected and the system adds an inflation layer for extra protection from collision.

4.2. Autonomous Behavior

Recent approaches for AVs have used prediction methods in order to avoid collision by estimating the future trajectory of the surrounding traffic participants. However, real traffic scenarios include complex interactions among various road users and need to handle complex clutter and modeling interactions with other road users to ensure safety. In the DARPA urban challenge, various solutions for planning were proposed; most of those solutions were specifically tailored to the competition demands. Many approaches (e.g., refs. [36,37]) use a state machine for AV to switch between predefined behaviors. These rule-based approaches need a safety assessment in order to deal with uncertainties. AV with human-like driving behavior requires interactive and cooperative decision-making.

Other vehicle's intentions need to be modeled and integrated into a planning framework that allows for intelligent, cooperative decision-making without the need for inter-vehicle communication. While AVs need the ability to reason the intentions of other participants, those also need to infer the AV's intention reasonably. This results in inter-dependencies and interactions based on the scene and shown behavior without the need for explicit communication [58].

By simulating the proposed traffic scenario, we can search for a possible best policy measured against the AV's cost function, and then the best policy is executed from the set of available policies for the AV. Possible trajectories can also be sampled, and the reaction to the environment can be determined according to the RA model.

The AV must be able to interact with other road users in accordance with codes of conduct and road traffic rules. For a given sequence of road segments specifying the selected route, the behavior layer is responsible for selecting appropriate driving behavior based on the perceived behavior of other road users and the road conditions. For instance, when the AV searches for a vacant parking space in a parking lot, the behavioral layer

instructs the vehicle to observe the behavior of other vehicles, pedestrians, and other objects during its movement and let the vehicle proceed once it is safe to go. Since the driving contexts and behaviors available in each context can be modeled as finite sets, a natural approach to automating this decision-making is to model each behavior as a state in a finite state machine with transitions controlled by the perceived driving context as the relative position to the planned route and nearby vehicles. Finite state machines, combined with different heuristics specific to the driving scenarios considered, have been adopted by most DARPA Urban Challenge teams as a behavioral control mechanism.

In the literature, similar approaches have been made by reducing decisions to a limited set of options and conducting evaluations with an individual set of policy assignments for each option. In general, the probabilistic representation of system models can be divided into four main types: discrete-time Markov chain (DTMC), continuous-time Markov chain (CTMC), Markov decision process (MDP), and probabilistic timed automata (PTA). A typical implementation of learning different driving styles in a highway simulation showed the potential of the probabilistic approach represented by MDP [59,60]. In ref. [61], the authors showed an enhanced version of the algorithm and its performance by generating human-like trajectories in parking lots, with only a few demonstrations required during learning. A partially observable MDP (POMDP), an extension of MDP, has been used in refs. [62,63] to integrate the road context and the motion intention of another vehicle in an urban road scenario.

Our RA and its physical environment have been modeled as a probabilistic timed program (PTP) model, which is an extension of PTA with the addition of discrete-valued variables that can be encoded into locations [51], while we used the probabilistic computational tree logic (PCTL) for specification logic [64]. A PTP incorporate probability, nondeterminism, dense real-time, and data. Its semantics are defined as infinite-state MDPs consists of the states of the environment and the transition between those states, which, through the conditional probabilities of the environment, correspond to triggering of predicates through the sensor system of the AV.

Pre-programmed rules are used to set the relationship between the perception predicates (beliefs) and the available actions. When this combination is verified by MCMAS during design time, then there will be no space for an unfeasible, repeated, or misleading action in the agent's actions list. The advantages of this operation are clearer when we deal with general real-life driving scenarios that could have a large number of predicates; in this case, the manual checking of those predicates would be unfeasible.

4.3. Planning System

Planning modules are concerned with vehicle motion and behavior in the perceived environment. Typically they comprise trajectory generation and reactive control for collision risk mitigation. They are organized into sub-maps to provide the flexibility of updating and to handle large environment maps. These are integrated and corrected for changes by a graph optimization approach with critical landmarks as nodes [65]. The planning system consists of two different components.

4.3.1. Path Planning

The RA is setting the waypoints to move the AV in the environment. These high-level commands are sent to the path planning ROS node to generate the route for the AV from the starting point to the desired destination. The entrance of the parking lot represents the starting point, and the destination is a free parking slot, which is an unknown place that needs to be discovered by the perception system while exploring the area. We used an ROS-based Dijkstra algorithm [66,67] for its simplicity and efficiency. This method represents the roads as a directed graph with weights represents the cost of passing a road segment. This process starts with a set of nodes (free space) that the AV can navigate and assigning a cost value to each one of them, this value is then increased with the next nodes, and the algorithm needs to find a path with minimum cost.

After finding the appropriate path, all the nodes in that path are translated into positions $P_i = (X_i, Y_i)^T$ in the reference axes. The outcome is not smooth, and some points are not compliant with the vehicle kinematics, and geometry, hence the second stage (motion planning) is necessary.

4.3.2. Motion Planning

In order to transform the global path into suitable waypoints, the timed-elastic band (TEB) motion planner creates a shorter set of waypoints $P_i = (x_i, y_i, \theta_i)^T$ within the original path planner waypoints. This takes into account, as much as possible, the vehicle constraints and the dynamic obstacles. Hence the map is reduced to the area around the AV and is continuously updating. When the path planning node determines the path of driving to be performed in the current context, then the ROS-based TEB local planner algorithm [68,69] will be used to translate this path into shorter continuous trajectories that are feasible for the control system and actuators to track and follow. This trajectory should also avoid collision with obstacles, detected by the sensors on-board, and should also be comfortable for the passengers. In case there is an object nearby, then the agent will check the possibility of collision using the PRISM model checker and then modify the trajectory when needed.

4.4. Control System

A control system is needed to execute the proposed trajectory of the AV. The move_base control node operates the AV by executing acceleration, braking, and steering messages. The control node takes the list of waypoints as input, and target velocities generated by the planning subsystem. Then sends these waypoints and velocities to an algorithm that calculates the amount of direction, acceleration, or deceleration to reach the target path.

Via a feedback controller, the appropriate actuator inputs are selected to perform the intended motion and correct the tracking of errors. These errors generated during the execution of a planned movement are due in part to the inaccuracies of the vehicle model. Therefore, the focus is placed on the robustness and stability of the controlled system. Different feedback controllers have been proposed in ref. [70] for executing the reference motions provided by the motion planning system.

Running the move_base node on the AV that is appropriately configured results in attempting to achieve a goal pose with its base to within a user-specified tolerance.

5. Rational Agent Design

5.1. Background

LISA [10] is a reactive agent that uses information from the environment in order to make a decision. These decisions are based around a set of *beliefs*, *desires*, and *intentions* that define its behavior [8,71]. Beliefs represent the knowledge derived from sensors to provide an observation of the current state of the environment. For example, if the sensors of an AV detect a person, the agent would hold the belief that a human was nearby. Desires correspond to the long-term goals of the agent. These long-term goals can correspond to states within the environment, for example, the position of an AV in a parking space, which the agent will use to attempt to establish in its behavior. Intentions, contrasting with beliefs, represent short-term goals of the agent, for example, once a person is detected, the RA will have the intention to avoid that person while they are nearby.

5.2. sEnglish

AV's decision-making programming is complicated, time-consuming, error-prone, and requires expertise in both the proposed tasks and the platform. There are many proprietary design tools in the industry [58,72] that require specialized knowledge. In order to simplify this process and to broaden the understanding of how decisions for AV actions are taken by non-experts to understand and verify the system in case of a legal need, such a method

could be crucial to law enforcement agencies, insurance companies, and lawyers in the event of an accident to review the program and the reason for AV taken a specific action.

The RA is implemented using sEnglish [17,52] natural language programming. Within sEnglish, the plans operate over a description of the world, which is captured within the system and environment ontology and maintained by data from sensors in the world model. The system ontology provides a simple, translatable description between concepts that a programmer and end-user would equally understand, such as common nouns, and those that an agent can use or manipulate, such as variables or pieces of data. In sEnglish, the agent's plans are described using English sentences in a structured text, including conditioning. The meaning of sentences is explained by an sEnglish text using sentences until further decomposition of meaning reaches the signal processing level when C++ is used to define the meaning. At this C++ level, no interpretable concepts need to be defined by the ontology.

The agent takes its decisions relying on information coming from its environmental model or knowledge base, which is a database regularly updated via sensors and perception mechanisms, and potential any learned inferences. This database is organized into a high-level ontology and provides information about the system and especially the current state of the environment.

Plans are declared by the programmer. Although this makes the agent less creative at run-time, as the plan library is fixed and not dynamically generated by the agent, this has significant advantages in terms of fast execution and viable formal verification [73]. In many safety-critical systems, such as formal verification, the core agent is crucial. Hence, this kind of BDI agent combines the advantages of deliberative agents with the advantages of reliability and explainability.

5.3. Mathematical Representation of the Agent

The LISA rational agent definition of our AV will follow [9,16,17] and it is based on AgentSpeak-like BDI architectures of robotic agents.

A rational agent in LISA can be fully defined and implemented by listing the following characteristics:

- *Initial Beliefs*.
  Initially, once the agent is initialized, it will have a set of beliefs about the environment. These beliefs are referred to as $B_0 \subset \mathcal{F}$ that are a set of literals that are automatically copied into the *belief base* B_t (the set of current beliefs) on initialization.
- *Initial Actions*.
  The initial actions $A_0 \subset A$ are a set of actions that are executed when the agent is first to run. Typically these actions are general goals that activate specific initial plans set up by the programmer.
- *Logic rules*.
  A set of logic-based implication rules, $L = R^P \cup R^B$, describes *theoretical* reasoning about physics and behavior rules to enable the agent to adjust its current knowledge about the world and influence its decision on actions to be taken.
- *Executable plans*.
  A set of *executable plans* or *plan library* Π. Each plan π_j is described in the form:

$$p_j : c_j \leftarrow a_1, a_2, \ldots, a_{n_j} \tag{1}$$

  where $p_j \in P_t$ is a *triggering predicate*, which allows the plan to be retrieved from the plan library whenever it comes true. Next the $p_j \in P_t$ allows the plan to be retrieved from the plan library whenever the belief base dictates that its triggering conditions are true; $c_j \in B$ is called the *context*, which allows the agent to check the state of the world, described by the current belief set B_t, before applying a particular plan; the $a_1, a_2, \ldots, a_{n_j} \in A$ then form a list of actions that the agent will execute.

The LISA rational agent defined in this paper will follow these rules and is defined:

$$\mathcal{R} = \{\mathcal{F}, B, L, \Pi, A\} \tag{2}$$

where:

- $\mathcal{F} = \{p_1, p_2, \ldots, p_{n_p}\}$ is the set of all predicates. In practice, this set can be infinitely large for general driving scenarios, however we are presenting this new approach to be tested on a specific limited driving scenario, which is driving in a parking lot. With some modifications and improvements this method could be generalized for other driving scenarios for future work.
- $B \subset \mathcal{F}$ is the atomic belief set, the set of all possible beliefs that the agent may encounter during operation. The current belief base at time t is defined as $B_t \subset B$. During operation, beliefs will always be changed. This occurs through *events* so that at a time t, beliefs may be added, deleted, or modified. These *events* are represented in the set $E_t \subset B$, which is called the *Event set*. Events may be based on *internal* or *external* actions. Internal actions are described as "mental notes". External inputs will appear through input from a sensor and are called "percepts" as they represent a measurement of the environment.
- $L = R^P \cup R^B = \{l_1, l_2, \ldots l_{n_l}\}$ is a set of implication rules. These are logic-based and represent a description of how the predicates B can be linked together and interpreted.
- $\Pi = \{\pi_1, \pi_2, \ldots, \pi_{n_\pi}\}$ is the set of executable plans or more formally *plans library*. At any time, t, there will be a collection of plans that could be activated. These are a subset of the complete plan library, $\Pi_t \subset \Pi$, which is commonly named the *Desire* set. A set $I \subset \Pi_t \subset \Pi$ of intentions is also defined. This set, l, contains plans that the agent is committed to executing. Each plan is built up as a sequence $\pi_j(\lambda_j)$ of actions where $\pi(0)$ is a triggering condition for the plan, and $\lambda_j > 0 \in A$ provides the subsequent series of actions that will be carried out.
- $A = \{a_1, a_2, \ldots, a_{n_a}\} \subset \mathcal{F} \setminus B$ is a set of all available actions. Actions may be either *internal*, when they either modify the knowledge base or generate internal events, or *external*, when they are linked to functions that operate in the environment.

This completes the definition of the AV agent used. The above list of steps are cyclically repeated to run the reasoning process of a robotic agent. Part of the agent program is shown in Figure 8 that has been used to generate PTP models for the AV and the other traffic participants based on perception predicates, the values shown are tailored to the physical characteristics of the vehicle.

In the example, the formation of plans is shown for an agent undertaking an autonomous parking maneuver. In this case, eight plans are presented that represent the agent's actions; each is represented by a triggering condition. The perception process represents sensing data that are collected on every evaluation. The '^[...]' represents the evaluation of a belief condition that can be set by an internal event. In this case, both plans start by evaluating whether a specific belief is matched. Should this belief be matched, a series of actions is then planned, again any element headed '^[...]' shows then update of a belief, elements shown within square brackets are executable sentences that contain code defined deeper within the structure which links to actuation.

Plan 1 can be read as follows: if I believe that no free parking space is detected, then I believe that I need to explore the parking lot. This is then extended by Plan 2, which can be read as if I believe I need to explore the parking lot, then a set of exploration waypoints should be generated, and these should be uploaded to activate the drive mode. Plan 3 is used to capture the condition when a parking space is detected and can be read: if I believe that I have detected a free space, then I can remove the belief that I need to explore the parking lot, and I believe I can commence parking operation.

Plan 4 contains the high-level code with trigger for planning this movement: if I believe that I can commence the parking operation, I should generate a set of waypoints for the parking and update the drive mode to reflect this. Plans 5, 6, and 7 can be read as two

pairs; each deals with the detection of an object, either a person or a moving vehicle. In each case, if it is detected at a distance between 12 m and 6 m then new set of waypoints is generated to avoid the object, if the distance between 6 m and 3 m, then the drive mode is switched to a slower mode, and a new set of waypoints is generated; otherwise, the vehicle is stopped.

```
PERCEPTION PROCESS
Monitor the following Boolean:
Parking space located.{[],[0,5]}
Pedestrian detected.{[],[-2,4]}
Generate PTP for Pedestrian.
{[I am at global waypoint],[0,0]}

EXECUTABLE PLANS
//Plan 1
If ^ ~[Parking space located] then +^[Need to explore parking lot.].
//Plan 2
If ^[Need to explore parking lot] then [Generate exploration waypoints.
    ]
[Update drive mode.].
//Plan 3
If ^[Free parking lot detected] then -^[Need to explore parking lot]
+^[Commencing parking operation].
//Plan 4
If ^[Commencing parking operation] then [Generate parking waypoints.]
[Update drive mode.].
//Plan 5
If ^[Pedestrian detected] while  ^[Distance more than 3m and less that
    6m] and
^[Object getting closer] then [Activate slow mode.]
[Generate object avoidance waypoints.]
+^[Object PTP generated.]
[Update drive mode.].
//Plan 6
If ^[pedestrian detected] while  ^[distance less than 3m] then
[Activate stop mode.]
[Update drive mode.].
//Plan 7
If ^[moving vehicle detected] while  ^[distance more than 6m and less
    that 12m] and ^[object getting closer] then
[Generate object avoidance waypoints.]
+^[object PTP generated]
[Update drive mode.].
.
.
.
```

Figure 8. Part of the agent code used to control the AV.

5.4. Connecting the RA to ROS

The sEnglish agent is natively compatible with ROS. The collection of sEnglish sentences that are set up by the programmer can comprise more complex sentences until atomic actions are then reached. These atomic actions can either be represented as sentences linked to libraries or native C++ code. The programmer can directly interface this C++ code to existing ROS libraries; therefore, the agent can be directly linked to the distributed ROS system.

A recent example of this operation is shown in handling nuclear material [18,74] for a robot arm. In this case, an sEnglish agent is developed and linked to an ROS network, in one case controlling a KUKA IIWA manipulator. In another, the agent is plugged into a different, but compatible drive for a KUKA KR180 manipulator. The only difference is the underlying drivers, providing an identical interface is provided, typically through topics and services available in ROS. The programmer can rapidly configure an sEnglish agent to operate within a distributed network for different applications.

6. Verification Methodology

In our decision framework, the agent uses model checkers MCMAS and PRISM to make appropriate and safe decisions for run-time operation. At design time, MCMAS can check if the logical reasoning system of the agent is consistent and stable [50]. The set of consistent and stable actions are fed into PRISM to find the most likely-to-succeed trajectory and action for that moment during the run-time operation.

6.1. Design-Time Verification

MCMAS is a symbolic model checker for multi-agent systems. It enables the automatic verification of specifications that use standard temporal modalities as well as the correctness, epistemic, and cooperation modalities. These additional modalities are used to capture the properties of various scenarios.

Agents can be described in MCMAS by the interpreted systems programming language (ISPL). The approach is symbolic and uses ordered binary decision diagrams (OBDDs), thereby extending standard techniques for temporal logic to other modalities distinctive of agents.

The logical reasoning system in the agent has a set of reasoning rules, which can be formulated as a Boolean evolution system (BES).

Definition 1 (Boolean evolution system). *$BES = \langle \mathcal{B}, \mathcal{R} \rangle$, where:*

- *$\mathcal{B} = \mathcal{B}^{known} \cup \mathcal{B}^{unknown}$ is a set of predicates (Boolean variables) $\mathcal{B} = \{b_1, \cdots, b_n\}$,*
- *$\mathcal{B}^{known} = \mathcal{B}^{true} \cup \mathcal{B}^{false}$ is a set of known predicates,*
- *$\mathcal{B}^{unknown}$ is a set of unknown predicates in its initial evaluation that could be determined later as $\mathcal{B}^{known}$ ($\mathcal{B}^{true} \vee \mathcal{B}^{false}$), or continue to be* Unknown,
- *$\mathcal{R}$ is a set of reasoning rules (evolution rules) of the form $X \rightarrow Y$, $\mathcal{R} = \{r_1, \cdots, r_m\}$ defined over $\mathcal{B}$.*

In the logical system, a Boolean variable in $\mathcal{B}^{known}$ usually represents a sensing event, e.g., a pedestrian comes close (e.g., within 5 m) to a vehicle. A pseudo-Boolean variable in $\mathcal{B}^{unknown}$ can express a belief, an action, or a consequence of an action, whose value is unknown at the beginning of a reasoning cycle.

When a guard g of a rule is evaluated to *true* on a valuation $\overline{\mathcal{B}}$ of $\mathcal{B}$, we say that the rule is *enabled*. After applying all enabled evolution rules over $\overline{\mathcal{B}}$ simultaneously, we obtain a new valuation $\overline{\mathcal{B}}'$. If two enabled rules set a variable to different values in $\overline{\mathcal{B}}'$, then the reasoning system is *inconsistent*. Starting from valuation $\overline{\mathcal{B}}^0$, we can apply the evolution rules infinitely and obtain valuations $\overline{\mathcal{B}}^1, \ldots, \overline{\mathcal{B}}^i, \ldots$ if the reasoning system is consistent. However, the system is *unstable* if for any pair of adjacent valuations $\overline{\mathcal{B}}^i$ and $\overline{\mathcal{B}}^{i+1}$, we have $\overline{\mathcal{B}}^i \neq \overline{\mathcal{B}}^{i+1}$.

6.2. Run-Time Verification

PRISM is a probabilistic model checker [27], a verification tool for modeling and formal analysis of systems that present probabilistic behavior. PRISM has been used to analyze different kind of systems from different domains, such as planning and synthesis, communication, game theory, performance and reliability, security protocols, etc. PRISM can build and analyze several probabilistic models including Markov decision processes (MDPs) plus extensions of these models with costs and rewards.

PTPs is an extension of MDPs with real-valued clocks and state variables. For timed automata formalisms, discrete variables are typically considered to be a straightforward syntactic extension since their values can be encoded into locations.

Given a set S, $\mathcal{P}(S)$ denotes the power set of S and $\mathcal{D}(S)$ the set of discrete probability distributions over S. A PTP contains a set of state variables and a set of clock variables. The state variables model the discrete events in the environment and the clock variables model the time elapse, which is a continuous process. Let $\mathcal{X}$ be the set of *clock variables*. The set of

clock valuations is defined as $\mathbb{R}_{\geq 0}^{\mathcal{X}} = \{t : \mathcal{X} \to \mathbb{R}_{\geq 0}\}$. Given a clock valuation t and $\delta \geq 0$, a *delayed* valuation $t + \delta$ is defined as $(t + \delta)(x) = t(x) + \delta$ for all $x \in \mathcal{X}$. Given a subset $Y \subseteq \mathcal{X}$, a new valuation $t[Y := 0]$ is defined by setting all clocks in Y to 0, i.e., $t[Y := 0](x)$ is 0 if $x \in Y$, and keeping other clocks unchanged. We used probabilistic discrete time and space in this work, hence it is necessary to use clock zones to set the time for each state and the transitions between states. A *clock zone* can be defined as a set of clock valuations that satisfy a number of clock difference constraints of the form: $\rho = \{t \in \mathbb{R}_{\geq 0}^{\mathcal{X}_0} \mid t_i - t_j \lesssim b_{ij}\}$. Let $Zones(\mathcal{X})$ be the set of all zones. Given a set $\mathcal{V}$ of state variables, let $Asrt(\mathcal{V})$, $Val(\mathcal{V})$, and $Assn(\mathcal{V})$ be a set of *assertions*, *valuations*, and *assignments* over $\mathcal{V}$, respectively.

Definition 2 (Probabilistic Timed Program (PTP) [75]). *A PTP is a tuple of the form:* $P = (L, l_0, \mathcal{X}, \mathcal{V}, v_i, \mathcal{I}, \mathcal{T})$ *where:*

- L *is a finite set of* locations*;*
- $l_0 \in L$ *is the* initial location*;*
- $\mathcal{V}$ *is a finite set of* state variables*;*
- $v_0 \in Val(\mathcal{V})$ *is the* initial valuation*;*
- $\mathcal{X}$ *is a finite set of* clocks*;*
- $\mathcal{I} : (L, \mathcal{V}) \to Zones(\mathcal{X})$ *is the* invariant condition*;*
- $\mathcal{T} : (L, \mathcal{V}) \to \mathcal{P}(Trans(L, \mathcal{V}, \mathcal{X}))$ *is the* probabilistic transition relation*, where:*

$$Trans(L, \mathcal{V}, \mathcal{X}) = Asrt(\mathcal{V}) \times Zones(\mathcal{X}) \times \mathcal{D}(Assn(\mathcal{V}) \times \mathcal{P}(\mathcal{X}) \times L)$$

A state of a PTP contains the valuation of L, $\mathcal{V}$, and $\mathcal{X}$, and written as (l, v, t). A new state can be reached by either an elapse of some time $\delta \in \mathbb{R}_{\geq 0}$ or a *transition* $\tau = (\mathcal{G}, \mathcal{E}, \Delta) \in \mathcal{T}(l)$ where $\mathcal{G} \in Asrt(\mathcal{V})$ is the guard, $\mathcal{E} \in Zones(\mathcal{X})$ is the enabling condition, and $\Delta = \lambda_1(f_1, r_1, l_1) + \cdots + \lambda_k(f_k, r_k, l_k))$ is a probability distribution over an *update* $f_j \in Assn(\mathcal{V})$, clock *resets* $r_j \subseteq \mathcal{X}$ and a *target location* $l_j \in L$.

When the agent starts a reasoning cycle, it will obtain a set of actions that can be safely applied, given the characteristics of the vehicle and measurement of the environment. This set of actions is predefined in the agent code during the design stage. If the set contains more than one action, then we use PTP to find the most suitable action for the AV to take. The most suitable action is the one that will not cause a collision, also compatible with the driving rules predefined for the agent, and will ultimately participate in reaching the destination in a shorter time and path can be considered as a safe action to apply, all of these parameters will be thought of by the agent and checked by the verification system while driving to make sure it is safe to apply. A PTP models the dynamic and uncertain physical environment containing the AV itself and other static or moving objects, such as pedestrians and other vehicles.

7. Verification of Decision-Making

This section presents an example of a parking lot scenario, where the AV is searching for a free parking space. During this process, the RA will continuously monitor the road users in its environment and decides its actions and trajectory based on the data from the perception system. The RA then checks all the probability of success of the intended actions before any execution using PRISM model checker.

MCMAS is used to verify (during design time) the beliefs, rules, actions, and their consequences that need to be considered within zone 1 and 2 of the AV, as shown in Figure 9. We used a limited set of rules and predicates for the parking lot scenario for proof of concept; real-life driving scenarios will need more rules and predicates to determine the proper behavior of the AV.

The AV needs to build a feasible trajectory and to maximize the distance from the objects around a suitable cost-map. The movements of the traffic participants are usually amenable to a probabilistic model based on the environment situation. A trajectory for a pedestrian walking in a parking lot is estimated by a prediction method [76,77], also ac-

counting for previously collected data sets in similar scenarios, e.g., Ref. [78]. A pedestrian may keep walking at the same speed if there is a car passing nearby or could reduce the speed, stop, or change the path; the same idea can be implemented for car drivers taking into account the vehicle dynamics.

In this work, the agent generates probabilistic behavior models for the non-stationary objects based on the observed situation and from previously recorded behavior of pedestrians and drivers in real-life scenarios. The method used for trajectory prediction has been combined with prior statistics for better estimation of the object's behavior. The verification system will take into consideration probabilities for the moving objects, verifying the intended actions against them using the PRISM probabilistic model checker, to select the most likely-to-succeed action for execution. The agent keeps updating the probabilistic models of the dynamic objects and sends it to an onboard PRISM in each reasoning cycle of the agent.

This operation is repeated as long as there are no objects within zone 1 of the AV shown in Figure 9, If there is any moving object within zone 2, then the AV will halve the speed. As soon as one of the moving objects comes across zone 1, then the AV will stop based on pre-programmed rules.

7.1. Design Time Verification in MCMAS

Here we define three sets of predicates: *sensing abstractions*, *future events consequences*, and *actions*, as listed below. The operational logic of the RA is restricted to the parking lot scenario. The RA will choose its decisions based on the sensory abstractions and a set of rules, as shown in Figure 10, those rules determine the best action to be carried out by the AV based on the sensing abstractions and the possible future event consequences. MCMAS is used to compute with the resulting Boolean evolution system to verify the logical stability and consistency of those predicates.

The number of those rules could rapidly increase depending on the driving scenario and the environmental situation. While it is challenging for the designer to check that there is no conflict between them manually for this simple case study, it will be even harder when taking into account other general driving scenarios.

1. The sensory abstractions of moving objects (Zone 1/outside the trajectory of the AV) are:
   - SONO1: pedestrian detected.
   - SONO2: car detected.
   - SONO3: object detected.
2. The sensory abstractions of moving objects (Zone 1/outside the trajectory of the AV but predicted to come across) are:
   - FSNE1: pedestrian detected.
   - FSNE2: car detected.
   - FSNE3: object detected.
3. The sensory abstractions of moving objects (Zone 1/within the trajectory of the AV) are:
   - SON1: pedestrian detected.
   - SON2: car detected.
   - SON3: object detected.
4. The sensory abstractions of moving objects (Zone 2/outside the trajectory of the AV but predicted to come nearer) are:
   - FSFE1: pedestrian detected.
   - FSFE2: car detected.
   - FSFE3: object detected.

5. The sensory abstractions of moving objects (Zone 2/within the trajectory of the AV) are:
   - SOF1: pedestrian detected.
   - SOF2: car detected.
   - SOF3: object detected.
6. The sensory abstractions of moving objects moving fast (Zone 2/outside the trajectory of the AV but predicted to come nearer) are:
   - FASP1: pedestrian detected.
   - FASP2: car detected.
   - FASP3: object detected.
7. The sensory abstractions of moving objects moving fast (Zone 2/within the trajectory of the AV) are:
   - FISP1: pedestrian detected.
   - FISP2: car detected.
   - FISP3: object detected.
8. The sensory abstractions for parking the AV:
   - PSA: parking space available.
   - PSNA: parking space not available.
9. The future events (consequences) for moving objects (Zone 1) are:
   - FCN1: pedestrian detected and will be collide.
   - FCN2: car detected and will be collide.
   - FCN3: object detected and will collide.
10. The future events (consequences) for moving objects (Zone 2) are:
    - FCF1: pedestrian detected and may collide.
    - FCF2: car detected and may collide.
    - FCF3: object detected and may collide.
11. The movement actions available to AV:
    - AM1: brake to stop.
    - AM2: proceed in reduced speed (2 mph).
    - AM3: proceed in normal speed (5 mph).
12. The parking actions available to AV:
    - AA1: generate new motion plan for parking.
    - AA2: return to previous motion plan.

7.1.1. Predicates Definition

Here we define three sets of predicates: *sensing abstractions*, *actions*, and *future events consequences*, as listed in Section 7.1. The operational logic of the RA is restricted to the parking lot scenario. The RA will choose its decisions based on the sensory abstractions and a set of logic rules, as shown in Figure 10; those rules determine the best action to be carried out by the AV based on the sensing abstractions and the possible future event consequences. MCMAS is used to compute the resulting Boolean evolution system to verify the logical stability and consistency of those rules.

7.1.2. Worst Case Mathematical Model

Each rule can be verified by computing the minimum space-time distance of the evolution of the progress of the oncoming car/pedestrian/object (denoted by E) and that of the AV (denoted by V):

$$E: \; E_c + [v_e t \cos(\alpha), \; v_e t \sin(\alpha), st], \; t > t_c \tag{3}$$

$$V: \; V_c + [v_a t \cos(\beta), \; v_a t \sin(\beta), st], \; t > t_c \tag{4}$$

Which describes the future movements of the environmental object and the AV, respectively. t_c is the current time when sensing of E and AV decisions have been completed, and E_c and V_c are the oncoming objects and the AV position at the time of the sensor measurement are abstracted, and the decision is made by the AV what to do. We say that no collision occurs in the worst case if the geometric distance (in 3D) of these two lines is greater than 1 m for any possible heading angle α and positions E_c outside Zone 1 and Zone 2 in Figure 9. s is a time separation factor defined as $s = 1$ m/s to make the dimensions in-space time compatible and used as a scaling factor for time equivalence of space separation (the smaller s is chosen, the bigger will be the time difference requirement for two objects occurring in the same place). The validity of all rules in Figure 10 has been checked using this type of simple worst-case analysis.

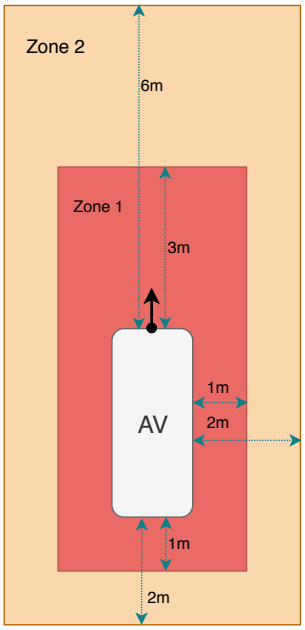

Figure 9. Schematic for the worst case analysis of the evolution rules in Figure 10. The black arrows illustrate approaching E with its sensing and decision paths extended on both sides of the 3 m and 6 m zones ahead and on the side of the AV (rectangular box).

```
Evolution:

    AM1=true if FSNE1=true or FSNE2=true or FSNE3=true or SON1=true
    or SON2=true or UON1=true or UON2=true or UON3=true;

    AM1=false if FSNE1=false and FSNE2=false and FSNE3=false and
    SON1=false and SON2=false and UON1=false and UON2=false
    and UON3=false or AA1=true;

    AM2=true if (SONO1=true or SONO2=true or SONO3=true or
    FSFE1=true or FSFE2=true or FSFE3=true or SOF1=true or SOF2=true
    or UOF1=true or UOF2=true or UOF3=true) and
    (AM1=false and AA1=false and AA2=false);

    AM2=false if (SONO1=false and SONO2=false and SONO3=false
    and FSFE1=false and FSFE2=false and FSFE3=false and SOF1=false
    and SOF2=false and UOF1=false and UOF2=false and UOF3=false) or
    (AM1=true or AA1=true and AA2=true);

.
.
.
.
.
```

Figure 10. Sample of the agent's evolution rules in MCMAS [33].

7.1.3. Stability and Consistency Check

Computation tree logic (CTL) [79] has been used in the verification of transition systems to specify properties that a system under investigation may possess. CTL is a branching-time logic, which considers all reasonable possibilities of future behavior for our limited parking lot driving scenario. We use CTL to formulate stability and consistency checks due to the efficient implementation of CTL model checking.

CTL is given by the following grammar:

$$\begin{aligned}\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid EX\varphi \mid EG\varphi \mid EF\varphi \mid E(\varphi\, \mathcal{U} \varphi) \mid \\ AX\varphi \mid AG\varphi \mid AF\varphi \mid A(\varphi\, \mathcal{U} \varphi)\end{aligned}$$

Lemma 1. *Inconsistency in the belief base can be verified by the following CTL formula:*

$$AG(\neg(EX\mathtt{B}_1 \wedge EX\neg\mathtt{B}_1) \wedge \cdots \wedge \neg(EX\mathtt{B}_n \wedge EX\neg\mathtt{B}_n)). \tag{5}$$

Proof. If a system is inconsistent, then there must exist two successor states after a specific state such that one of them is evaluated to *true* and the other to *false*. The formula $EX\mathtt{B}_i \wedge EX\neg\mathtt{B}_i$ captures this case for variable b_i. The negation $\neg(\ldots)$ captures the occurrence of inconsistency through b_i. Operator AG formulates that inconsistency does not occur in any state, and we do not need to consider unknown valued variables as they cannot be assigned to unknown during evolution. □

The Boolean evolution system is consistent in case the above formula evaluated to true.

Lemma 2. *The instability problem can be checked by the following CTL formula:*

$$\begin{aligned}AF((AG\ \mathtt{B}_1 \vee AG\neg\mathtt{B}_1 \vee AG\ \mathtt{K}_1) \wedge \cdots \wedge \\ (AG\ \mathtt{B}_{n_1} \vee AG\neg\mathtt{B}_{n_1} \vee AG\ \mathtt{K}_{n_1}) \wedge \\ (AG\ \mathtt{B}_{n_1+1} \vee AG\neg\mathtt{B}_{n_1+1}) \wedge \cdots \wedge \\ (AG\ \mathtt{B}_n \vee AG\neg\mathtt{B}_n)).\end{aligned} \tag{6}$$

The Boolean evolution system is stable in case the above formula evaluated to true.

Proof. For stability, we need that every path ends with a stable state, where unknown variables will not change their values anymore. Therefore, the *unknown* variable b_i will hold one of three cases $AG\ \mathtt{B}_i$, $AG\ \neg\mathtt{B}_i$, or $AG\ \mathtt{K}_i$ in the stable state. The latter means that the *known* variable cannot take value *unknown* during the evolution, and the *unknown* variables cannot take value *known*. Thus, they will not be considered in the CTL formula. AF means that eventually a stable state will be reached. □

Consistent rules cannot generate contradictory conditions throughout the whole reasoning process, which means that at no time, a predicate can be assigned to *true* and *false* simultaneously. Stable rules make the reasoning process terminate in finite steps. In another word, a *stable* evaluation is reached eventually such that this stable evaluation is obtained by extending the reasoning process one step further. The detailed proofs of those lemmas are illustrated in our previous work [33,50]. Table 2 below is showing the properties of the verified system.

Table 2. Properties of the verification of the agent logic during design-time in MCMAS model checker.

Execution Time in (s)	Number of Reachable States	BDD Memory in Use	Peak Number of Nodes
0.038	1,052,670	6,641,468	16,352

7.2. Run-Time Verification in PRISM

Probabilistic decision-making and threat-assessment methods assign probabilities to different events, e.g., how likely it is to collide with another object in the next few seconds given some assumptions on uncertainties. However, when assumptions are violated (e.g., pedestrian walks/runs faster than anticipated) then sensors onboard will detect the speed of moving objects in real-time. When necessary, the vehicle will stop depending on speed measurements for the AV and other objects, and will deal with a pedestrian running towards the AV as a possible threat. The vehicle will stop if the pedestrian is within a specific distance from the driving path of the vehicle to avoid collision.

Figure 11 illustrates the proposed scenario for the AV in terms of trajectory generation based on the possible behavior of other objects around where the AV is moving forward looking for a free parking space, at the same time, a pedestrian and a vehicle (P2, V1) is moving towards the AV, another pedestrian (P1) standing in position (x = 3.5 m, y = 4 m) from the vehicle in relative coordinates. The RA will generate PTP models for the two traffic participants and also for the AV to find the best trajectory and speed under the current circumstances. The RA will keep updating the PTP models with every reasoning cycle (100 ms) and verifying those PTPs using PRISM model checker.

Because the object (P1) is not moving and it is outside (zone 2), also not in the same path of the AV, hence the agent will ignore it, and the AV will continue moving at the same speed (5 mph). However, if the pedestrian (P1) entered (zone 2), then the AV will reduce the speed according to sensory abstraction (SOF1) and action (AM2). For demonstration purposes, we discretized the trajectory by one meter apart. We also discretized the possible pedestrian's and vehicle's trajectories. While in the implementation, the RA is getting these data continuously in real-time from the perception system without the need for discretization.

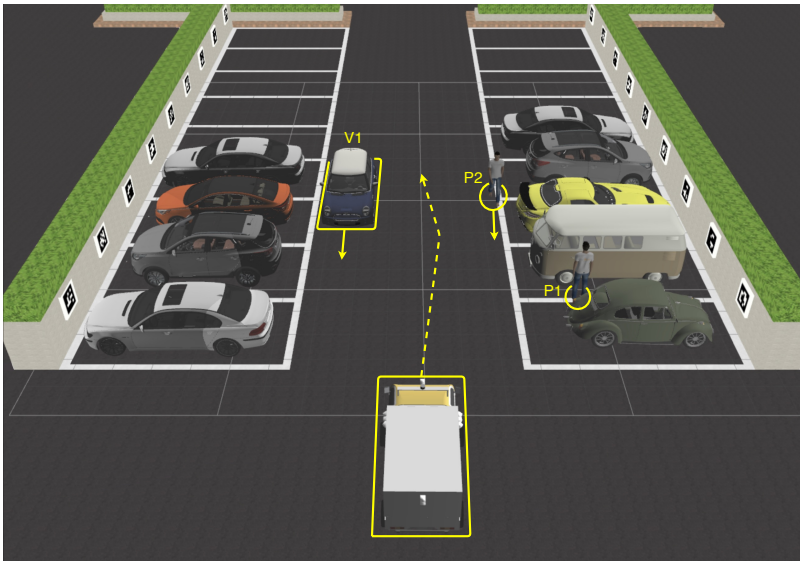

Figure 11. A driving scenario in simulation.

For the agent to build a meaningful PTP model while the AV is moving, it has been formerly equipped with a possible probabilistic behavior for both pedestrians and drivers in such an environment. Usually, when a pedestrian notices an oncoming vehicle, they may slow down with a high probability. The pedestrian may also choose to stop at some point or even change the lane to a safer one, it is also common that the pedestrian may be distracted by something, e.g., using a mobile device, and hence, does not notice the AV. If this is the case, the pedestrian continues to walk at the average speed. The last case could be included in the generated model of the traffic participants using methods explained in [80]. While there are some similar probabilities for the driver with limits to the dynamic movements of the vehicle, the driver may decide to continue driving the same speed, reduce the speed, or to stop in order to give a chance for the AV to pass easily. From the above scenario in

Figure 11, we can see that the vehicle and the pedestrian are in the same horizontal line, and this gives a small gap for the AV to pass through.

To simulate a realistic scenario, and to equip the RA with the possible behavior of pedestrians and drivers, we recorded some data and objects behavior manually for parking lots, and we used JAAD dataset [81] for pedestrians and drivers reactions to vehicles around them in different scenarios. A next step implementation would be to equip the agent with probabilistic behavior prediction method, e.g., Refs. [82,83] instead of the limited approach adopted in this work.

Generating PRISM Models from the Agent Code

We designed a translator that works as an Eclipse plugin (part of the sEnglish system environment) to translate the agent reasoning code to PTP models that can be verified by PRISM; it is a direct text processing algorithm in C++ that can run in a few milliseconds (hence its time is neglected). The agent will also translate the properties of the models in PCTL and the query of questions the agent needs to ask. As soon as the equivalent PTP models verified, then the agent will know about different properties expressed in PCTL. A Boolean variable for each belief is defined, and transition probabilities are taken from the probability distributions defined in the sEnglish code.

7.3. Verification Example of a Parking Scenario

As mentioned before, all the possible states of the system can be explored during formal verification, including some extreme cases that may be difficult to discover during testing. A general parking scenario will be presented here to illustrate the use of the RA predicates (sensory abstractions, beliefs, actions, and future event consequences) designed for this case study: we have defined two regions around the AV for safety purposes depending on the direction of movement, as shown in Figure 9, assuming the AV is moving forward, as soon as the AV detect an object within (6 m) in front or (2 m) any other side represented by (zone 2), the AV will slow down from average speed of (5 mph) to (2 mph), as soon as this object become within (3 m) from the front of the AV or (1 m) from any other side represented by (zone 1), the AV will stop. Here it is essential to mention that the experimental AV has been equipped with a means of communication with other pedestrians and drivers using audio to prevent a deadlock state when the AV stop and wait for others to move and vice versa, the AV will play a voice to say to others that "you are free to move and the AV will wait for you". We have defined further details for the AV to deal with the traffic participants around by calculating the speed of those objects using the LiDAR sensor. Assuming there is an object moving fast towards the AV, as soon as this object enter (zone 2) the AV will stop instead of slowing down, this will give more time for the other object (running pedestrian or fast-moving car) to reduce their speed, change direction, or to stop, and this will reduce or eliminate any possible collision.

A simple proposed example of how the agent chooses its actions is as follows: based on the scenario in Figure 11, a car (V1) is coming in the opposite direction to the AV from a distance of (8 m) and the driver starts to slow down when they notice another vehicle coming, the AV is moving at its average speed and building its trajectory based on the map and the moving objects around. As soon as the other vehicle (V1) enters (zone 2), the sensing event (SOF2) from Section 7.1 will be activated, and this will activate the future event (FCF2) then this will trigger action (AM2), which leads to slow down. In the meanwhile, the walking pedestrian (P2) enters (zone 2), sensing event (FSFE1) is triggered and this may lead to collision according to a future event (FCF1), the AV here will not take any further action because it is already working in reduced speed. However, as soon as the car or the pedestrian or both enter (zone 1) (SON2 or FCNE1 or both), future events (FCN1 or FCN2, or both), this will trigger the action (AM1) to execute stop action. All the stationary parked cars in the parking lot will not be considered as a threat because they are not in the proposed path of the AV, and they are not moving.

The regulation for the speed of vehicles in a parking lot is limited to (10 mph), based on this and for safety reasons and prototype development we set the speed of our AV to be (5 mph) in case of no moving objects within (zone 1 and 2). As mentioned, both the RA and the planning system will send control commands to the move base system to set the movements of the AV. However, actions such as (AA1 and AA2) have a pre-programmed sequence for performing a parking maneuver, as shown in the video link we referred to in the abstract. In case there are two or more rules in conflict with each other, MCMAS will present this case by a counterexample showing how the inconsistency is reached. Further, it cannot be the case that two different actions are activated at the same time.

For the run-time verification, the initial PTP model generated by the RA for the AV's trajectory is shown in Figure 12. We use a relative coordinate system considering that the LiDAR position on the top of the AV is the center of the coordinates at any time, knowing that the RA is taking the dimensions of the AV into calculations while processing. In this example we will refer to the coordinates of the participants according to a fixed moment at a particular time interval (x_1, y_1) to represent the coordinates of the AV, (x_2, y_2) for object (P2), the (x_3, y_3) for object (V1). The (C) letter in the PTP models represents the clock, which will be counting and resetting with every transition. The complexity of solving PTPs with two or more clocks is EXPTIME-complete. Our previous work [64] shows experiments on several complex models and properties and the results are promising.

Figure 13 shows the PTP model for the pedestrian's possible behavior. For this example, we assume that the average speed of the pedestrian is near the speed of the AV inside the parking lot. The pedestrian may prefer to stop after noticing the AV with the probability of (0.1) or to stop later when the distance became critical. We assume that the pedestrian will keep walking in the same lane with a probability of (0.6), they could also decide to change the lane and walk behind the moving car (V1) for more safety with a probability of (0.3). In both cases, the pedestrian may prefer to walk at the same speed or to reduce it with some probabilities, as shown in Figure 13.

Figure 14 shows the possible PTP model for (V1). We assumed that the driver might notice the AV and decide to stop with probability (0.1). With a probability of (0.6), the driver may decide to slow down, or may prefer to continue the same speed with a probability of (0.3). The RA will then modify the AV's PTP model according to the newly generated behavior model of the other objects around.

Note that the parameters used to generate the PTP models, such as the speed and probability, may not reflect the exact behavior of the AV, P2, or V1. The RA is building those PTPs based on the location, speed, and direction of the moving objects. In general, this framework will help to predict a possible behavior for the different objects around, then to verify the current trajectory/action for the AV against the possible trajectory/action of the nearby objects, and this will help in reducing the possibility of collision. More accurate PTP models could be generated after collecting more behaving data through real tests.

To avoid any possible collision, we require that the pedestrian and/or the vehicle is at least (1 m) away from the AV. This can be represented by the following expression:

$$\phi \equiv (x_1 - x_2)^2 + (y_1 - y_2)^2 >= 1. \tag{7}$$

$$\phi \equiv (x_1 - x_3)^2 + (y_1 - y_3)^2 >= 1. \tag{8}$$

where (x_2, y_2), represent the coordinate of the pedestrian, (x_3, y_3) is the coordinate for the car. As PRISM cannot deal with real numbers, we multiply the distance by 2 (we partition the distance by 0.5 m. Therefore, the location would have values such 0.5 and 1.5 m, by multiplying it by 2, we obtain an integer). We compute the maximum probability of the violation of Equations (7) and (8), by the following PCTL property:

$$\mathtt{P}_{max=?}[\mathtt{F}\ \neg\phi]. \tag{9}$$

Due to the discretization of the trajectory, the negation of Equation (7) is translated into the following expression:

$$(((x_2 > x_1 \wedge x_2 - x_1 \leq 1) \vee (x_1 > x_2 \wedge x_1 - x_2 \leq 1)) \wedge$$
$$((y_2 > y_1 \wedge y_2 - y_1 \leq 1) \vee (y_1 > y_2 \wedge y_1 - y_2 \leq 1)))$$

While the negation of Equation (8) is translated into:

$$(((x_3 > x_1 \wedge x_3 - x_1 \leq 1) \vee (x_1 > x_3 \wedge x_1 - x_3 \leq 1)) \wedge$$
$$((y_3 > y1 \wedge y_3 - y_1 \leq 1) \vee (y_1 > y_3 \wedge y_1 - y_3 \leq 1)))$$

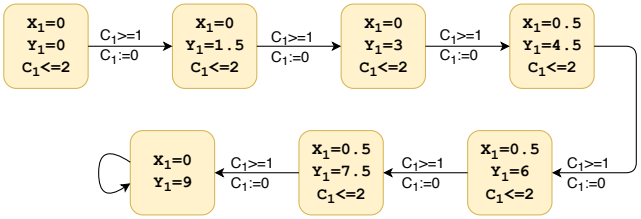

Figure 12. Initial PTP model for the AV's behavior.

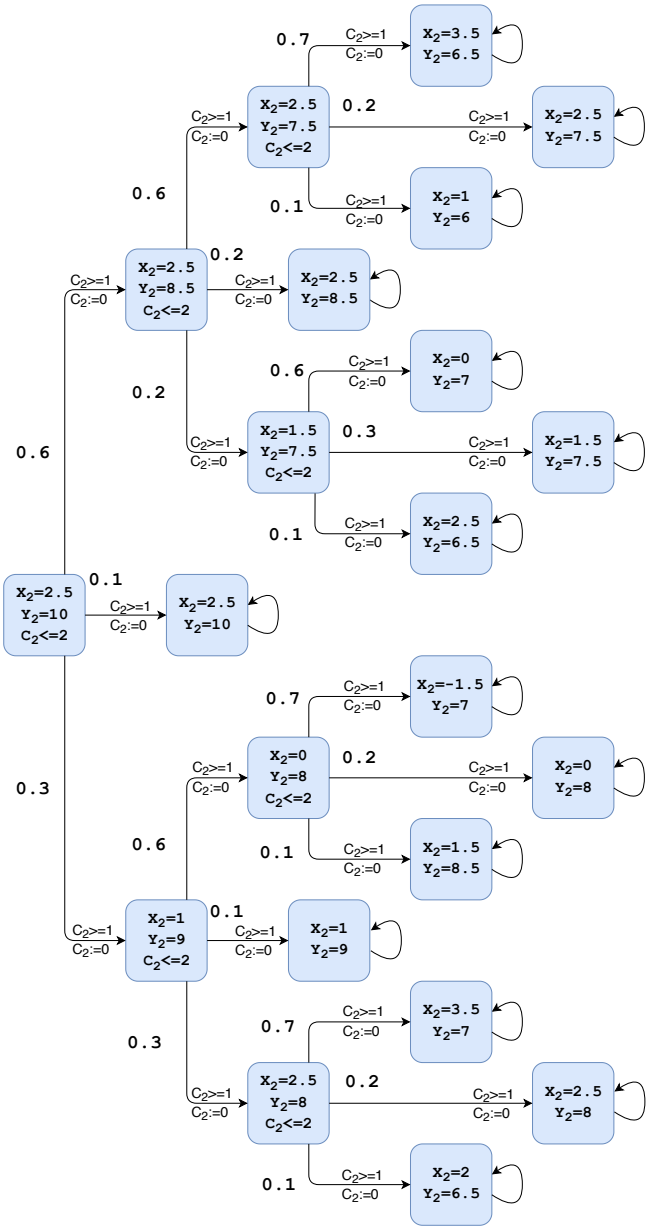

Figure 13. PTP model for the pedestrian's behavior.

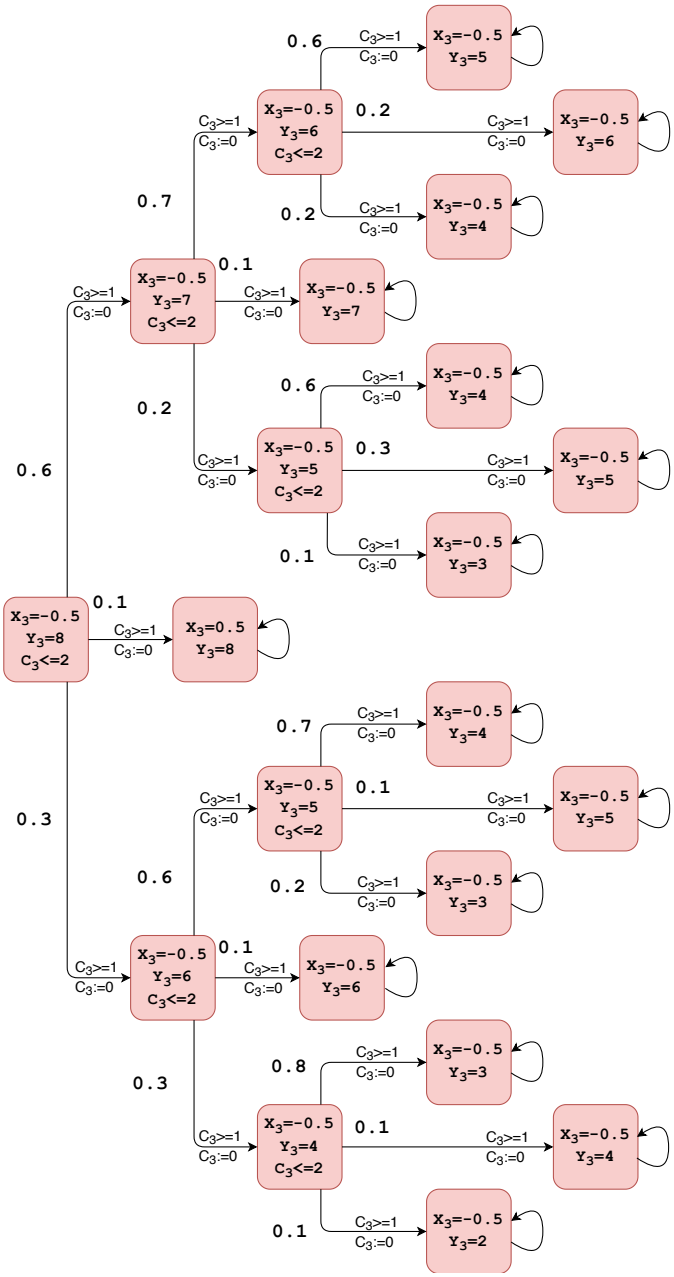

Figure 14. PTP model for the vehicle's behavior.

The verification results for the proposed scenario are shown in Table 3 returned from PRISM for Formula (9), which indicates information about the model generated for both the pedestrian and the car and the chance of collision with every one of them under the current motion plan.

Table 3. Verification results for the proposed scenario using PRISM model checker.

PTP Model	States	Transitions	Choices	Ver. Time	Maximum Collision Probability
Pedestrian	1238	3884	3624	0.036 s	0.252
Car	659	2230	1960	0.019 s	0.003

All the computations in this work were carried out using two computers running on Ubuntu OS version 16.04, first is equipped with (Intel core i7 CPU, 16 GB of RAM, and GTX 1070 GPU) for simulation, perception, planning, and control systems and the

second with (Intel core i7, 16 GB of RAM, and GTX 860 GPU) to run the agent code and the verification platform.

8. Conclusions and Future Work

A new approach is presented for the verification of an agent-based decision-making system for a self-driving vehicle. The approach considers both the design-time and run-time verification. To contribute towards the open-source development of the self-driving vehicles, a self-driving vehicle is presented in the simulation that is available in ROS and the Gazebo Simulator.

A rational agent in a real traffic scenario usually faces a vast amount of situations with related behavior rules. Many of these can be identified during the design stage. Remaining scenarios, with possible probabilistic events in the environment, can then be handled by run-time evaluations. Our approach is presented through a case study. The power of the combination of the two verification tools can help the designer to eliminate any conflict and redundancy in the agent predicates. Further, the verification tools can help to check the agent rules for any possible instability or inconsistency with the benefit of obtaining a counterexample when a faulty state has been reached.

The second stage of verification deals with the possible behavior of the traffic participants to determine the probability of success for the best AV action. A limited set of beliefs, rules, and actions are presented to provide a proof of concept and to illustrate the proposed platform. For higher levels of rationality, the agent could yet be equipped, during design time, with a methodology for rules and predicates generation. Such a system would be able to learn new driving scenarios for run-time verification by implementing a machine learning approach.

Author Contributions: Conceptualization, M.A.-N. and S.V.; methodology, M.A.-N.; software, M.A.-N. and S.W.; validation, M.A.-N.; formal analysis, M.A.-N. and H.Q.; investigation, M.A.-N. and S.W.; resources, M.A.-N. and J.A.; data curation, M.A.-N.; writing—original draft preparation, M.A.-N., H.Q. and J.A.; writing—review and editing, M.A.-N., H.Q. and J.A.; visualization, M.A.-N., H.Q. and J.A.; supervision, S.V.; project administration, M.A.-N. and S.V. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data available on request due to restrictions. The data presented in this study are available on request from the corresponding author. The data are not publicly available due to privacy.

Acknowledgments: A special thanks to: (1) TATA Motors European technical center for providing the electric vehicle and some sensors used in this work. (2) AMRC research center for hosting the researcher and the vehicle and for providing a space for tests. (3) Mark Tucker and Maradona Rodrigues, TATA Motors, for their valuable comments and suggestions on this project.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Buehler, M.; Iagnemma, K.; Singh, S. *The 2005 DARPA Grand Challenge: The Great Robot Race*; Springer: Berlin/Heidelberg, Germany, 2007; Volume 36.
2. Buehler, M.; Iagnemma, K.; Singh, S. *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*; Springer: Berlin/Heidelberg, Germany, 2009; Volume 56.
3. Furgale, P.; Schwesinger, U.; Rufli, M.; Derendarz, W.; Grimmett, H.; Mühlfellner, P.; Wonneberger, S.; Timpner, J.; Rottmann, S.; Li, B.; et al. Toward automated driving in cities using close-to-market sensors: An overview of the v-charge project. In Proceedings of the 2013 IEEE Intelligent Vehicles Symposium (IV), Gold Coast, Australia, 23–26 June 2013; pp. 809–816.
4. Chan, C.Y. Advancements, prospects, and impacts of automated driving systems. *Int. J. Transp. Sci. Technol.* **2017**, *6*, 208–216. [CrossRef]

5. Singh, S. Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey. In *Technical Report, National Highway Traffic Safety Administration*; 2015. Available online: https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812506 (accessed on 5 April 2019).
6. Zhang, F.; Clarke, D.; Knoll, A. Vehicle detection based on lidar and camera fusion. In Proceedings of the 17th International IEEE Conference on Intelligent Transportation Systems (ITSC), Qingdao, China, 8–11 October 2014; pp. 1620–1625.
7. Okumura, B.; James, M.R.; Kanzawa, Y.; Derry, M.; Sakai, K.; Nishi, T.; Prokhorov, D. Challenges in perception and decision making for intelligent automotive vehicles: A case study. *IEEE Trans. Intell. Veh.* **2016**, *1*, 20–32. [CrossRef]
8. Rao, A.S.; Georgeff, M.P. Modeling rational agents within a BDI-architecture. *KR* **1991**, *91*, 473–484.
9. Veres, S.M.; Molnar, L.; Lincoln, N.K.; Morice, C.P. Autonomous vehicle control systems—A review of decision making. *J. Syst. Control Eng.* **2011**, *225*, 155–195. [CrossRef]
10. Izzo, P.; Qu, H.; Veres, S.M. A stochastically verifiable autonomous control architecture with reasoning. In Proceedings of the 55th IEEE Conference on Decision and Control, CDC'16, Las Vegas, NV, USA, 12–14 December 2016; pp. 4985–4991.
11. Dennis, L.A.; Fisher, M.; Lincoln, N.K.; Lisitsa, A.; Veres, S.M. Practical verification of decision-making in agent-based autonomous systems. *Autom. Softw. Eng.* **2016**, *23*, 305–359. [CrossRef]
12. Seshia, S.A.; Sadigh, D.; Sastry, S.S. Formal methods for semi-autonomous driving. In Proceedings of the 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 8–12 June 2015; pp. 1–5.
13. Luckcuck, M.; Farrell, M.; Dennis, L.A.; Dixon, C.; Fisher, M. Formal specification and verification of autonomous robotic systems: A survey. *ACM Comput. Surv. CSUR* **2019**, *52*, 1–41. [CrossRef]
14. SAE On-Road Automated Vehicle Standards Committee. Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems. *SAE Stand. J.* **2014**, *3016*, 1–16.
15. Huang, X.; Kwiatkowska, M.Z. Reasoning about cognitive trust in stochastic multiagent systems. In Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 4–9 February 2017.
16. Wooldridge, M. *An Introduction to MultiAgent Systems*; Wiley: Chichester, UK, 2002.
17. Lincoln, N.K.; Veres, S.M. Natural Language Programming of Complex Robotic BDI Agents. *J. Intell. Robot. Syst.* **2013**, *71*, 211–230. [CrossRef]
18. Dennis, L.A.; Aitken, J.M.; Collenette, J.; Cucco, E.; Kamali, M.; McAree, O.; Shaukat, A.; Atkinson, K.; Gao, Y.; Veres, S.M.; et al. Agent-based autonomous systems and abstraction engines: Theory meets practice. In *Annual Conference Towards Autonomous Robotic Systems*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 75–86.
19. Kamali, M.; Dennis, L.A.; McAree, O.; Fisher, M.; Veres, S.M. Formal verification of autonomous vehicle platooning. *Sci. Comput. Program.* **2017**, *148*, 88–106. [CrossRef]
20. Al-Shihabi, T.; Mourant, R.R. A framework for modeling human-like driving behaviors for autonomous vehicles in driving simulators. In Proceedings of the Fifth International Conference on Autonomous Agents, Montreal, QC, Canada, 28 May–1 June 2001; pp. 286–291.
21. da Costa Sousa, J.M.; Palm, R.; Silva, C.; Runkler, T.A. Optimizing logistic processes using a fuzzy decision making approach. *IEEE Trans. Syst. Man Cybern. Part A Syst. Hum.* **2003**, *33*, 245–256. [CrossRef]
22. Palm, R.; Bouguerra, A.; Abdullah, M.; Lilienthal, A.J. Navigation in human-robot and robot-robot interaction using optimization methods. In Proceedings of the 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Budapest, Hungary, 9–12 October 2016; pp. 004489–004494.
23. Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; Ng, A.Y. ROS: An open-source Robot Operating System. In *ICRA Workshop on Open Source Software*; McGill University, Centre for Intelligent Machines (CIM): Kobe, Japan, 2009; Volume 3, p. 5.
24. Koenig, N.; Howard, A. Design and use paradigms for gazebo, an open-source multi-robot simulator. In Proceedings of the Intelligent Robots and Systems, (IROS 2004), Sendai, Japan, 28 September–2 October 2004; Volume 3, pp. 2149–2154.
25. Lomuscio, A.; Qu, H.; Raimondi, F. MCMAS: A model checker for the verification of multi-agent systems. In Proceedings of the International Conference on Computer Aided Verification, Grenoble, France, 26 June–2 July 2009; Springer: Berlin/Heidelberg, Germany, 2009; pp. 682–688.
26. Izzo, P.; Qu, H.; Veres, S.M. Reducing complexity of autonomous control agents for verifiability. *arXiv* **2016**, arXiv:1603.01202.
27. Kwiatkowska, M.; Norman, G.; Parker, D. PRISM 4.0: Verification of probabilistic real-time systems. In *Computer Aided Verification*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 585–591.
28. Sadigh, D.; Sastry, S.; Seshia, S.A.; Dragan, A.D. Planning for autonomous cars that leverage effects on human actions. In *Robotics: Science and Systems*; University of Michigan: Ann Arbor, MI, USA, 2016; Volume 2.
29. McAree, O.; Aitken, J.M.; Veres, S.M. Towards artificial situation awareness by autonomous vehicles. *IFAC-Pap.* **2017**, *50*, 7038–7043.
30. Gleirscher, M.; Kugele, S. Defining risk states in autonomous road vehicles. In Proceedings of the 2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE), Singapore, 12–14 January 2017; pp. 112–115.
31. Sadigh, D.; Landolfi, N.; Sastry, S.S.; Seshia, S.A.; Dragan, A.D. Planning for cars that coordinate with people: Leveraging effects on human actions for planning and active information gathering over human internal state. *Auton. Robot.* **2018**, *42*, 1405–1426. [CrossRef]

32. Li, L.; Ota, K.; Dong, M. Humanlike driving: Empirical decision-making system for autonomous vehicles. *IEEE Trans. Veh. Technol.* **2018**, *67*, 6814–6823. [CrossRef]
33. Al-Nuaimi, M.; Qu, H.; Veres, S.M. Computational Framework for Verifiable Decisions of Self-Driving Vehicles. In Proceedings of the 2018 IEEE Conference on Control Technology and Applications (CCTA), Copenhagen, Denmark, 21–24 August 2018; pp. 638–645.
34. Al-Nuaimi, M.; Qu, H.; Veres, S.M. A stochastically verifiable decision making framework for autonomous ground vehicles. In Proceedings of the 2018 IEEE International Conference on Intelligence and Safety for Robotics (ISR), Shenyang, China, 24–27 August 2018; pp. 26–33.
35. Levinson, J.; Askeland, J.; Becker, J.; Dolson, J.; Held, D.; Kammel, S.; Kolter, J.Z.; Langer, D.; Pink, O.; Pratt, V.; et al. Towards fully autonomous driving: Systems and algorithms. In Proceedings of the 2011 IEEE Intelligent Vehicles Symposium (IV), Baden-Baden, Germany, 5–9 June 2011; pp. 163–168.
36. Leonard, J.; How, J.; Teller, S.; Berger, M.; Campbell, S.; Fiore, G.; Fletcher, L.; Frazzoli, E.; Huang, A.; Karaman, S.; et al. A perception-driven autonomous urban vehicle. *J. Field Robot.* **2008**, *25*, 727–774. [CrossRef]
37. Urmson, C.; Anhalt, J.; Bagnell, D.; Baker, C.; Bittner, R.; Clark, M.; Dolan, J.; Duggins, D.; Galatali, T.; Geyer, C.; et al. Autonomous driving in urban environments: Boss and the urban challenge. *J. Field Robot.* **2008**, *25*, 425–466. [CrossRef]
38. Bhadani, R.K.; Sprinkle, J.; Bunting, M. The cat vehicle testbed: A simulator with hardware in the loop for autonomous vehicle applications. *arXiv* **2018**, arXiv:1804.04347.
39. Dosovitskiy, A.; Ros, G.; Codevilla, F.; Lopez, A.; Koltun, V. CARLA: An open urban driving simulator. In *Conference on Robot Learning, Proceedings of the Machine Learning Research*; 2017; pp. 1–16. Available online: http://proceedings.mlr.press/ (accessed on 5 April 2019).
40. Rong, G.; Shin, B.H.; Tabatabaee, H.; Lu, Q.; Lemke, S.; Možeiko, M.; Boise, E.; Uhm, G.; Gerow, M.; Mehta, S.; et al. Lgsvl simulator: A high fidelity simulator for autonomous driving. In Proceedings of the 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC), Rhodes, Greece, 20–23 September 2020; pp. 1–6.
41. Chavez-Garcia, R.O.; Aycard, O. Multiple sensor fusion and classification for moving object detection and tracking. *IEEE Trans. Intell. Transp. Syst.* **2015**, *17*, 525–534. [CrossRef]
42. Park, W.J.; Kim, B.S.; Seo, D.E.; Kim, D.S.; Lee, K.H. Parking space detection using ultrasonic sensor in parking assistance system. In Proceedings of the 2008 IEEE Intelligent Vehicles Symposium, Eindhoven, The Netherlands, 4–6 June 2008; pp. 1039–1044.
43. Agarwal, V.; Murali, N.V.; Chandramouli, C. A cost-effective ultrasonic sensor-based driver-assistance system for congested traffic conditions. *IEEE Trans. Intell. Transp. Syst.* **2009**, *10*, 486–498. [CrossRef]
44. Kianpisheh, A.; Mustaffa, N.; Limtrairut, P.; Keikhosrokiani, P. Smart parking system (SPS) architecture using ultrasonic detector. *Int. J. Softw. Eng. Appl.* **2012**, *6*, 55–58.
45. Lee, M.; Kim, S.; Lim, W.; Sunwoo, M. Probabilistic occupancy filter for parking slot marker detection in an autonomous parking system using avm. *IEEE Trans. Intell. Transp. Syst.* **2018**, *20*, 2389–2394. [CrossRef]
46. Wikipedia. Self-Driving Car Incidents. 2019. Available online: https://en.wikipedia.org/wiki/List_of_self-driving_car_fatalities (accessed on 13 February 2021).
47. Fernandes, L.E.; Custodio, V.; Alves, G.V.; Fisher, M. A rational agent controlling an autonomous vehicle: Implementation and formal verification. *arXiv* **2017**, arXiv:1709.02557.
48. Giaquinta, R.; Hoffmann, R.; Ireland, M.; Miller, A.; Norman, G. Strategy synthesis for autonomous agents using PRISM. In *NASA Formal Methods Symposium*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 220–236.
49. Trimble Inc. SkechUp Software. 2017. Available online: https://www.sketchup.com/ (accessed on 24 April 2018).
50. Qu, H.; Veres, S.M. Verification of logical consistency in robotic reasoning. *Robot. Auton. Syst.* **2016**, *83*, 44–56. [CrossRef]
51. Kwiatkowska, M.; Norman, G.; Parker, D. A framework for verification of software with time and probabilities. In *International Conference on Formal Modeling and Analysis of Timed Systems*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 25–45.
52. Veres, S.M. *Natural Language Programming of Agents and Robotic Devices*; Sysbrain Ltd.: London, UK, 2008.
53. Seenouvong, N.; Watchareeruetai, U.; Nuthong, C.; Khongsomboon, K.; Ohnishi, N. A computer vision based vehicle detection and counting system. In Proceedings of the Knowledge and Smart Technology (KST), Chiangmai, Thailand, 3–6 February 2016; pp. 224–227.
54. Kato, T.; Guo, C.; Kidono, K.; Kojima, Y.; Naito, T. SpaFIND: An effective and low-cost feature descriptor for pedestrian protection systems in economy cars. *IEEE Trans. Intell. Veh.* **2017**, *2*, 123–132. [CrossRef]
55. Redmon, J.; Farhadi, A. YOLOv3: An Incremental Improvement. *arXiv* **2018**, arXiv:1804.02767.
56. Dollár, P.; Appel, R.; Belongie, S.; Perona, P. Fast feature pyramids for object detection. *IEEE Trans. Pattern Anal. Mach. Intell.* **2014**, *36*, 1532–1545. [CrossRef]
57. Zhang, J.; Singh, S. LOAM: Lidar Odometry and Mapping in Real-time. In *Robotics: Science and Systems*; University of California: Berkeley, CA, USA, 2014; Volume 2, p. 9.
58. Schwarting, W.; Alonso-Mora, J.; Rus, D. Planning and decision-making for autonomous vehicles. *Annu. Rev. Control. Robot. Auton. Syst.* **2018**, *1*, 187–210. [CrossRef]
59. Abbeel, P.; Ng, A.Y. Apprenticeship learning via inverse reinforcement learning. In Proceedings of the Twenty-First International Conference on Machine Learning, Banff, AB, Canada, 4–8 July 2004; ACM: New York, NY, USA, 2004; p. 1.

60. Karasev, V.; Ayvaci, A.; Heisele, B.; Soatto, S. Intent-aware long-term prediction of pedestrian motion. In Proceedings of the 2016 IEEE International Conference on Robotics and Automation (ICRA), Stockholm, Sweden, 16–21 May 2016; pp. 2543–2549.
61. Abbeel, P.; Dolgov, D.; Ng, A.Y.; Thrun, S. Apprenticeship learning for motion planning with application to parking lot navigation. In Proceedings of the 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, Nice, France, 22–26 September 2008; pp. 1083–1090.
62. Hubmann, C.; Schulz, J.; Becker, M.; Althoff, D.; Stiller, C. Automated driving in uncertain environments: Planning with interaction and uncertain maneuver prediction. *IEEE Trans. Intell. Veh.* **2018**, *3*, 5–17. [CrossRef]
63. Gindele, T.; Brechtel, S.; Dillmann, R. Learning driver behavior models from traffic observations for decision making and planning. *IEEE Intell. Transp. Syst. Mag.* **2015**, *7*, 69–79. [CrossRef]
64. Dräger, K.; Kwiatkowska, M.; Parker, D.; Qu, H. Local abstraction refinement for probabilistic timed programs. *Theor. Comput. Sci.* **2014**, *538*, 37–53. [CrossRef]
65. Paden, B.; Čáp, M.; Yong, S.Z.; Yershov, D.; Frazzoli, E. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Trans. Intell. Veh.* **2016**, *1*, 33–55. [CrossRef]
66. Skiena, S. Dijkstra's algorithm. In *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*; Addison-Wesley: Reading, MA, USA, 1990; pp. 225–227.
67. Jihoonl. Dijkstra algorithm-ROS Package. 2014. Available online: http://wiki.ros.org/asr_navfn (accessed on 13 February 2018).
68. Rösmann, C.; Feiten, W.; Wösch, T.; Hoffmann, F.; Bertram, T. Efficient trajectory optimization using a sparse model. In Proceedings of the 2013 European Conference on Mobile Robots, Barcelona, Spain, 25–27 September 2013; pp. 138–143.
69. Roesmann, C. teb_local_planner. 2018. Available online: http://wiki.ros.org/teb_local_planner (accessed on 15 February 2019).
70. Adouane, L. *Autonomous Vehicle Navigation: From Behavioral to Hybrid Multi-Controller Architectures*; CRC Press: Boca Raton, FL, USA, 2016.
71. Rao, A.S.; Georgeff, M.P. An Abstract Architecture for Rational Agents. In *Principles of Knowledge Representation and Reasoning*; Morgan Kaufmann Publishers Inc.: Cambridge, MA, USA, 1992; pp. 439–449.
72. Bhat, A.; Aoki, S.; Rajkumar, R. Tools and methodologies for autonomous driving systems. *Proc. IEEE* **2018**, *106*, 1700–1716. [CrossRef]
73. Fisher, M.; Dennis, L.A.; Webster, M. Verifying Autonomous Systems. *ACM Commun.* **2013**, *56*, 84–93. [CrossRef]
74. Aitken, J.M.; Veres, S.M.; Shaukat, A.; Gao, Y.; Cucco, E.; Dennis, L.A.; Fisher, M.; Kuo, J.A.; Robinson, T.; Mort, P.E. Autonomous nuclear waste management. *IEEE Intell. Syst.* **2018**, *33*, 47–55. [CrossRef]
75. Hazim, M.Y.; Qu, H.; Veres, S.M. Testing, Verification and Improvements of Timeliness in ROS processes. In Proceedings of the Conference Towards Autonomous Robotic Systems, Sheffield, UK, 26 June–1 July 2016; Springer: Berlin/Heidelberg, Germany, 2016; pp. 146–157.
76. Ridel, D.; Rehder, E.; Lauer, M.; Stiller, C.; Wolf, D. A literature review on the prediction of pedestrian behavior in urban scenarios. In Proceedings of the 2018 21st International Conference on Intelligent Transportation Systems (ITSC), Maui, HI, USA, 4–7 November 2018; pp. 3105–3112.
77. Rasouli, A.; Kotseruba, I.; Tsotsos, J.K. Understanding pedestrian behavior in complex traffic scenes. *IEEE Trans. Intell. Veh.* **2017**, *3*, 61–70. [CrossRef]
78. Kooij, J.F.P.; Schneider, N.; Flohr, F.; Gavrila, D.M. Context-based pedestrian path prediction. In *European Conference on Computer Vision*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 618–633.
79. Clarke, E.M.; Emerson, E.A.; Sistla, A.P. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. *ACM Trans. Program. Lang. Syst.* **1986**, *8*, 244–263. [CrossRef]
80. Rangesh, A.; Trivedi, M.M. When Vehicles See Pedestrians with Phones: A Multicue Framework for Recognizing Phone-Based Activities of Pedestrians. *IEEE Trans. Intell. Veh.* **2018**, *3*, 218–227. [CrossRef]
81. Rasouli, A.; Kotseruba, I.; Tsotsos, J.K. Agreeing to cross: How drivers and pedestrians communicate. In Proceedings of the Intelligent Vehicles Symposium (IV), Los Angeles, CA, USA, 11–14 June 2017; pp. 264–269.
82. Saleh, K.; Hossny, M.; Nahavandi, S. Intent prediction of pedestrians via motion trajectories using stacked recurrent neural networks. *IEEE Trans. Intell. Veh.* **2018**, *3*, 414–424. [CrossRef]
83. Wiest, J.; Höffken, M.; Kreßel, U.; Dietmayer, K. Probabilistic trajectory prediction with Gaussian mixture models. In Proceedings of the 2012 IEEE Intelligent Vehicles Symposium, Alcala de Henares, Spain, 3–7 June 2012; pp. 141–146.