Proceedings of the 2018 IEEE International
Conference on Intelligence and Safety for Robotics
Shenyang, China, August 24-27, 2018

A stochastically verifiable decision making framework for autonomous ground vehicles

Mohammed Al-Nuaimi[1], Hongyang Qu[2] and Sandor M. Veres[3]

Abstract— **A framework is presented for probabilistic verification of Autonomous Vehicles' (AV) decisions by rational agents onboard. The AV's are assumed to be equipped with the necessary perception and control systems, which are needed for their awareness of the environment and enable them to make decisions. The decisions arrived at by the agents are verified by the probabilistic model checking techniques presented. The objective of the new framework is to reduce design complexity of decision making while ensuring verifiability of the decisions made by the rational agents. Probabilistic Timed Programs (PTPs) are used to model the environmental scenarios and Probabilistic Computational Tree Logic (PCTL) to specify the properties to be verified. Both PTP and PCTL are deployed by use of the PRISM model checker. For demonstration purposes, the Robot Operating System (ROS) and the Gazebo simulator are used to model and test the vehicle systems and their signal processing. sEnglish Publisher, the high abstraction level programming tool, is used along with a Jason agent architecture to provide rules and plans for decision making. Matlab is relied on to assist in programming the perception and control systems in the demonstration.**

I. Introduction

It is a common requirement that an AV must be able to progress without human control from its initial position to its final destination and handle various interruptions during its journey. This process involves making decisions by a software onboard the AV, based on information observed by sensors. This software can be a mission planner [1] [2] or an intelligent agent [3] [4], or some other approach, including agent-oriented programming (AOP) [5]. Sensors normally used are cameras, lidars, radars, GPS sensors and IMUs, usually in various combinations, to build continuously updated models of the environment and their predictions, which are then used for decision making in view of the movement options of the vehicle. The agent needs to have some skills of situation analysis in order to make decisions and to take action based on simultaneous localisation and mapping (SLAM), path planning and motion control [1] [6] [7].

Reconfigurable, adaptive and predictive control systems are well capable of robustly progressing a vehicle on its planned path collisions free. However, to make decisions with foresight in a social context, and to also prevent strange behaviour and triggering errors by human drivers, integration into an overall logic and experience-based decision-making process is important. A promising framework, agent-based systems, have been rapidly developing during the past three decades. Some notable agent types are reactive, deliberative, multi-layered and belief-desire-intention (BDI) agents [8] [9] [10]. Jason is a multi-layered BDI approach to intelligent agents with belief, desire and intention paradigms, which are particularly suitable for achieving robotic and vehicle system goals [11] [12], as it can also integrate path planning through external function calls.

Autonomous systems are characterised by their ability to determine what they need to do and how to do it. Naturally, when applying those systems, many questions will arise regarding their efficiency, reliability and the degree of their safety. Testing of these systems through prototype development will try to answer operational safety questions partially. However, the best that can be done by testing is the generation of a representative set of scenarios on real vehicles. Tests and simulation are unable to account for rare combination of environmental events that may appear in real driving scenarios. Hence formal verification, based on detailed analysis, will remain an essential tool in this context. Simulation can only provide illustrations of the continuous dynamics and behaviour of the AV and its surrounding environment but cannot thoroughly check all combinations of states and inputs that lead to a particular action. If good dynamical models are available to encapsulate robotic skills of sensing and action, then formal verification can be used, as based on model checking, to verify properties of a finite representation of an interacting model of the robot and its environment. The autonomous agent's decisions depend on the current state of the environment, and the way it perceives data from its sensors. However, when model checking is used to verify an autonomous agent's decisions and actions, the process should define the uncertainty associated with the environment's state by using probabilistic model checking [13]. Other research in this area include [1] [2] [14] focus on hardware implementation of AV system, [7] [15] have addressed the challenges of the simulation of an AV system, while [16] [4] present different methods for verification, preceding the one in this paper.

The contribution of this paper is represented by the overall system design that can perceive the environment, process the data and then decide the next action autonomously with a focus on the verification of those decisions for real-time application.

Section 2 presents the general framework of the project. In Section 3 we go through the main components of the

[1]Mohammed Al-Nuaimi is a PhD student at Department of Automatic Control and Systems Engineering (ACSE), University of Sheffield. myhaziml@sheffield.ac.uk

[2]Dr Hongyang Qu is Senior Research Fellow at ACSE, University of Sheffield. h.qu@sheffield.ac.uk

[3]Sandor M. Veres is a Professor at ACSE, University of Sheffield. s.veres@sheffield.ac.uk

proposed system, starting with a simulation of the AV along with its sensors and actuators in ROS [17] and Gazebo simulator [18] then the perception system for the AV. In section 5 we cover the Jason agent design in sEnglish, followed by an explanation of the verification method used in this work, and finally, we cover the main goal of this paper, which is probabilistic run-time verification of the decision-making process, further explained in the case study section.

II. General Framework

The general approach of this work is to model an AV's system by models with all the necessary parts, including the dynamic model of the vehicle, sensors needed for perception, control systems and the rational agent as a decision maker onboard the vehicle. We briefly introduce the various components of the system and then focus our attention on a new run-time verification tool for the chosen actions of the agent in real-time.

The agent has to generate a model of the surrounding areas it passes through, with the ability to localise the static and dynamic objects and keep updating the model after each perception cycle. Such a software agent needs to have both logic-based reasoning and a set of skills (perception and control system).

The logic based reasoning system has been implemented in natural language programming (NLP) of agents in sEnglish [19] that compiles into the Jason agent [9]. The skills needed by the AV have been developed in MATLAB, including localisation, mapping, path planning, motion planning, object detection and recognition and the control of the vehicle.

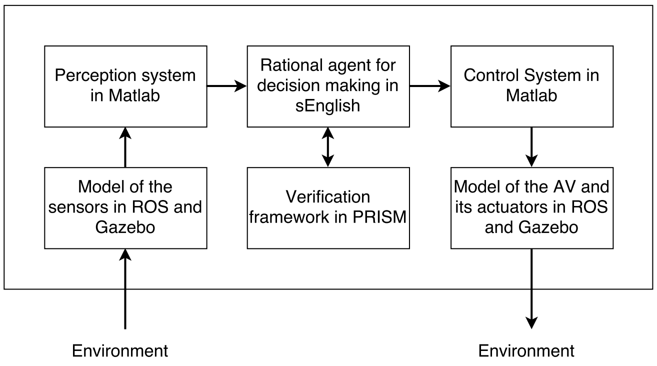
Perception system in Matlab
Rational agent for decision making in sEnglish
Control System in Matlab
Model of the sensors in ROS and Gazebo
Verification framework in PRISM
Model of the AV and its actuators in ROS and Gazebo
Environment
Environment

Fig. 1. General Framework for probabilistic verification of decision making

Fig. 1 shows a simplified overall diagram of the proposed framework. The perception and control system provides information and feedback to agent reasoning for decisions on the applications of control and manoeuvring skills using the actuators. A novelty of this paper is that the decisions under various environmental events are modelled by *Probabilistic timed programs* (PTP) [20] [21] and then verified by the model checker PRISM [22] in order to obtain a set of probabilities for the vehicle's actions to satisfy given requirements.

III. Models of the proposed system

The model of the AV and its sensors have been developed in ROS and Gazebo simulator for autonomous parking scenario as shown in Fig. 2.

ROS is a flexible framework to implement a robot's software. It enables the creation of a set of nodes as processes, which can communicate with each other by service requests and message broadcasting to serve as implementation platform of robotic software in various programming languages.

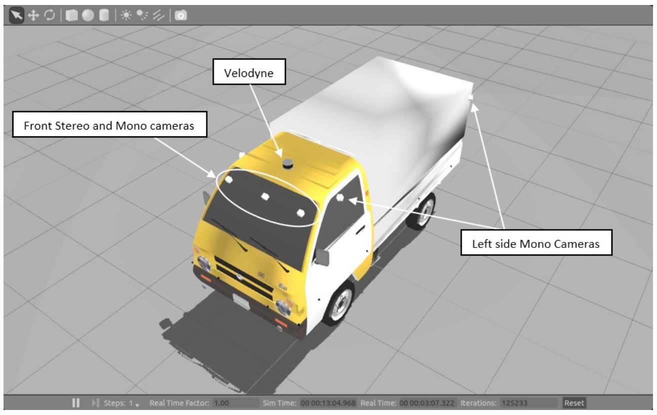

Fig. 2. A VR model of the TATA ACE and its sensors

Creating a dynamical model of the AV is an important step to start with. A faithful-to-reality model makes it possible to test basic physical properties affecting the decision making algorithms, and perform assessment of probabilities of recognition and action execution using realistic scenarios. Gazebo simulator is capable of efficiently and accurately simulate of robots in complex environments. A number of sensors have been used with the AV to percept the environment surrounding the AV, those sensors are:

- LiDAR (VLP-16).
- Cameras (one stereo, and seven mono).
- Low-end GPS.
- IMU.
- Odometer.

Some scenarios have been created to assess quality of perception and the coverage area for the LiDAR and the cameras allocated to the vehicle. The AV model is controlled by the rational agent and control system, the performance of which affects decision making.

IV. Perception and control system

An autonomous vehicle software through agent-oriented programming can provide a verifiable decision-making framework for vehicle autonomy, in which the success of correct sensing, e.g. traffic, obstruction, road and object detection/recognition for environment mapping is carried out correctly with only specific probabilities [4] [23] known.

A. Principles of operation

The cameras have been allocated to the AV to provide 360° coverage. Vehicle surrounding zones can be classified as based on heading direction and ego vehicle speed. The zone that is directly in front of the vehicle is highly critical, zones perpendicular to the longitudinal direction of the travel are low-critical, and all other zones are in the medium-criticality category. In our sensor suite, four cameras are providing the front coverage, which are a stereo pair, one central camera and one side cameras on each side. Physical measurements are provided by a 3D LiDAR which has 360° horizontal rotational coverage and 30° vertical coverage. To enhance the perception and the redundant support of LiDAR, we have designed it in a way such that it could move to both sides of the vehicle with tilting capability. The position of the sensor and possible tilt angle would be regulated based on the number and position of the objects around the AV, and also considering any camera failures, processing time limitations and performance influenced by lighting or traffic density for hardware implementation of the proposed system.

B. *Perception Modules*

Perception merges the data from the sensors and creates semantic knowledge of the environment in which a vehicle is operating. It includes objects detection, classification, dynamic state estimation and also actors intent prediction. Semantic understanding shall be used for error detection, the potential degradation of functionality and sensor management. The perceived environment of the vehicle is represented in passive world model for internal (decision and control, etc.) and external (HMI, V2X, etc.) processes. The localisation of a vehicle with the respect of map is part of this group too.

Our test vehicle has been designed with a focus on parking based applications with a development focus on robust SLAM [24], behaviour confidence prediction, formal verification of algorithms execution time and functionality.

For filtering dynamic objects [25] in the surroundings, raw map data are segmented into static and moving objects, the later ones are used for tracking and the former for grid map building.

Cameras deliver series of images at (24 fps) which are synchronised with an external trigger; these images are grouped in front, sides and rear view for processing them separately. Pre-processing block does image registration, warping and resizing for these images separately.

These conditioned images are used for dense point cloud generation with the fusion of 3D LiDAR as shown in Fig. 3. Fusion enhances the quality of support points which in turn helps in generating accurate disparity map. The dense 3D point cloud is segmented to support in ROI for object classification of landmarks and agents/actors.

When the cameras detect an object, the equivalent LiDAR segment data is received for the distance measurements by comparing the position of the detected object from the image frame with the LiDAR 3D point cloud points belonging to that object. Based on the depth map, the objects positions are calculated from the ROI, those measurements from LiDAR are calculated according to the coordinates transformation. Finally, the position of the detected objects and the distance

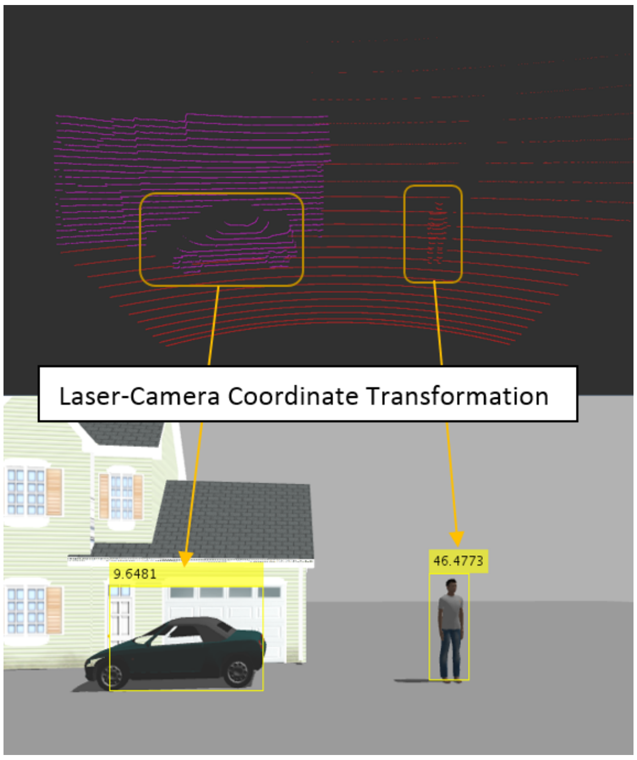

Fig. 3. Sensory cooperation between LiDAR and camera

from them are sent to the rational agent for further processing. The HoG features have been used in this work for its simplicity and low computational cost.

The HoG algorithm used here has been trained using real images; hence we got a lower detection rate from using the algorithm in the simulated environment.

C. *Decision and Control Modules*

Decision and control modules are concerned with vehicle motion and behaviour in the perceived environment. Typically they comprise collision-free trajectory generation, energy and fault management and reactive control for collision risk mitigation. To provide the flexibility of updating and to handle large maps, environment maps are organised in sub-maps. These are integrated and corrected for changes by graph optimisation approach with critical landmarks as nodes.

Maps, which are digitised offline are not always up to date, not sufficiently complete and also have some measurement errors. For achieving robustness in the situational evaluation architecture should have provision to build/update the map, is made by filtered SLAM for metric representation, for ease of data storage/sharing it is parameterised [26], for scalability and lifelong mapping.

Global trajectory planning generates route and path segments from start position to the destination. Route constitutes a sequence of road links and in path segments waypoints. Trajectory generation includes functions for trajectory waypoint selection and trajectory evaluation on screened paths. Finally, trajectory control consists of longitudinal and lateral vehicle motion controllers for motion realisation.

V. Rational Agents in Jason

Jason [9] is an extension of AgentSpeak, a logic-based AOP language based on the BDI agent paradigm. The agent's architecture consists of declarations of initial beliefs and goals, agent logic, perception processes and prewritten plans to execute common tasks the agent needs to do.

The belief base is used to abstract the agent's perception of the environment and predicted behaviours of other traffic participants. It is continuously updated as traffic situations develop on the road. Route-planning-based subgoals of the agent are achieved by the execution of prewritten plans in the Jason program. In each of its reasoning cycle, the agent can trigger some of its plans and reason with logic. The reasoning cycle is repeated ten times per second, each of those cycles will consider the new beliefs, decides the intentions and sub-goals, perform logical implication and start executing the pre-written plans if their context is applicable. Unexpected events can trigger an interrupt of the execution of prewritten plans.

Definition 1 (Rational agent): A formal description of the *Jason* ($\mathcal{R}$) *agent based on rational BDI agent* is a tuple:

$$\mathcal{R} = \{\mathcal{B}, \mathcal{G}, \mathcal{M}, \mathcal{E}, \mathcal{L}, \mathcal{A}\} \quad (1)$$

where

- $\mathcal{B}$ is a total belief set, where $\mathcal{B}_0 \subset \mathcal{B}$ is the set of initial beliefs and $\mathcal{B}_t \subset \mathcal{B}$ is the currently 'true' belief set at time t. Belief b can be extended with internal variables as $b(x, y, \ldots)$ for a richer description. The agent assigns a value of 'false', 'true' or 'unknown' to every predicate $b \in \mathcal{B}$ at the end of each reasoning cycle. This set represents what the agent belief about itself and its surrounding environment both when it starts to run and then later during operation.
- $\mathcal{G}$ is a total goal set, of which $\mathcal{G}_0 \subset \mathcal{G}$ is a set of initial goals and $\mathcal{G}_t \subset \mathcal{G}$ is the current goal set at time t. A goal g can be extended with internal variables as $g(x, y, \ldots)$ for a richer description. During each reasoning cycles, those Goals can be created or erased while working. They simply represent what the agent wants to achieve during its operation.
- $\mathcal{M}$ is a set of messages generated by other agents and human supervisors and as incoming and outgoing communications by the agent. Those messages will be interpreted into beliefs in $\mathcal{B}_t$ during reasoning cycles. In this way, the agent can communicate with the outside world represented by other agents around and/or a supervisor. In case of the AV, the supervisor could be the owner of the vehicle where the AV can inform him of messages such as the vehicle has parked successfully or no free space has been found in the parking lot.
- $\mathcal{E}$ is an ordered list of executable plans $\pi_1, \pi_2, \ldots$ The agent should be equipped with an extended list of plans that can be executed during operation. A plan could affect the agent beliefs, goals or actions.
- $\mathcal{L}$ is a set of logic-based implication rules that the agent should follow while operation, these rules will organise the agent behaviour with its environment in a logical manner.
- $\mathcal{A}$ is a set of executable actions, of which $\mathcal{A}_I \subset \mathcal{A}$ is the set of initial actions. Those actions will determine the agent behaviour; an action is chosen based on the agent beliefs, goals and plans.

The format of each executable plan $\pi_i \in \mathcal{E}$ is:

$$triggering\_event : context \rightarrow body.$$

The plan can be activated by an addition or a deletion of a triggering predicate b_i from the current belief base $\mathcal{B}_t$ ($+b_i$ or $-b_i$) or triggering g_i from the current goal base $\mathcal{G}_t$ ($+!g_i$ or $-!g_i$), only if the $context$ is satisfied, which is a propositional logic formula of predicates from $\mathcal{B}$ or $\mathcal{G}$. $body$ is a sequence of actions, predicates (added or erased from the belief base using '$+$' or '$-$') and goals to be achieved in order to process the event.

The Jason program development has been carried out by an Eclipse plugin application programming interface (API) for sEnglish Publisher [27], including compilation from Jason to PRISM, this compilation is done by a special translator built for this purpose where its capable of auto-generation of the PRISM code represented by PTP modules directly from Jason program. A PTP module describes merely the next action or set of actions for the AV and the possible behaviour for the objects around. This relationship between the agent and the other objects is determined by a set of probabilities, with some formulas, the agent can determine the probability of success, and based on this result provided by PRISM model checker the AV can proceed forward with the current set of actions or modify them to proceed with better action behaviour based on logical rules. The Simulink implementation has been done by use of the Matlab/Simulink based Agent Executive Toolbox (AET) [28]. The code for the skills of the agent, which occurred in the action files of the sEnglish project, were compiled to Matlab functions [29] [19].

VI. Verification of the Agent's actions

The previous section presented a well-defined decision structure for agents. Here we develop our approach in [30] further in order to formally verify agents' decisions.

As an autonomous vehicle runs in a dynamic and uncertain physical environment, we need a modelling formalism that can deal with real-time and uncertainty. In this work, we adopt *Probabilistic timed programs* [20] [21], which have been developed from Markov Decision Processes along with state variables and real-time clocks. A PTP is composed of states of the environment and transitions between those states. A state corresponds to a snapshot of the status of the environment, and a transition corresponds to triggering of predicates through the sensor system of the robotic agent at a certain probability, and feeds back newly generated predicates to the agent's belief base under various states of the environment.

This section illustrates how probability distributions, when combined with the logic based decision making of the agent and the environmental PTP, can be modelled in PRISM [22].

A. Probabilistic timed programs (PTP)

Given a set S, let $\mathcal{P}(S)$ be the set of subsets of S and $\mathcal{D}(S)$ the set of discrete probability distributions over S. Let $\mathcal{V}$ be a set of variables. We define $Asrt(\mathcal{V})$, $Val(\mathcal{V})$ and $Assn(\mathcal{V})$ to be a set of *assertions*, *valuations* and *assignments* over $\mathcal{V}$ respectively. A set $\mathcal{X}$ of *clock* variables denotes the time elapsed since the occurrence of various events. The set of *clock valuations* is $\mathbb{R}^{\mathcal{X}}_{\geq 0} = \{t : \mathcal{X} \to \mathbb{R}_{\geq 0}\}$. Given a clock valuation t and $\delta \geq 0$, we define a *delayed* valuation $t + \delta$ as $(t + \delta)(x) = t(x) + \delta$ for all $x \in \mathcal{X}$. Given a subset $Y \subseteq \mathcal{X}$, we obtain a new valuation $t[Y := 0]$ by setting all clocks in Y to 0, i.e., $t[Y := 0](x)$ is 0 if $x \in Y$, and other wise, $t(x)$. A clock *zone* is the set of clock valuations that satisfy a number of clock difference constraints of the form: $\rho = \{t \in \mathbb{R}^{\mathcal{X}_0}_{\geq 0} \mid t_i - t_j \lesssim b_{ij}\}$. Let $Zones(\mathcal{X})$ be the set of all zones.

Definition 2 (PTP): A PTP is a tuple $P = (L, l_0, \mathcal{V}, v_0, \mathcal{X}, \mathcal{I}, \mathcal{T})$ 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 $Trans(L, \mathcal{V}, \mathcal{X}) = Asrt(\mathcal{V}) \times Zones(\mathcal{X}) \times \mathcal{D}(Assn(\mathcal{V}) \times \mathcal{P}(\mathcal{X}) \times L)$.

The evolution of a PTP from a state (l, v, t) is composed of two steps:

1) An elapse of some time $\delta \in \mathbb{R}_{\geq 0}$;
2) A *transition* $\tau = (\mathcal{G}, \mathcal{E}, \Delta) \in \mathcal{T}(l)$.

The transition is described by a *guard* $\mathcal{G} \in Asrt(\mathcal{V})$, an *enabling condition* $\mathcal{E} \in Zones(\mathcal{X})$, and a probability distribution $\Delta = \lambda_1(f_1, r_1, l_1) + \cdots + \lambda_k(f_k, r_k, l_k))$ over an *update* $f_j \in Assn(\mathcal{V})$, clock *resets* $r_j \subseteq \mathcal{X}$ and a *target location* $l_j \in L$.

The delay δ is chosen in such way that the invariant $\mathcal{I}(l)$ remains satisfied continuously. As $\mathcal{I}(l)$ is a (convex) clock zone, this is equivalent to ensuing that both t and $t + \delta$ satisfy the invariant $\mathcal{I}(l)$. Transition τ must be *enabled*, i.e., the guard $\mathcal{G}$ and the enabling condition $\mathcal{E}$ in τ must be satisfied by v and $t + \delta$, respectively. After τ is executed, an assignment, the set of clocks that are reset, and the successor location are selected according to the distribution Δ in τ.

B. Performance queries

Definition 3 (pctl):

$$\begin{aligned} \phi &::= \ \texttt{true} \mid a \mid \phi \wedge \phi \mid \neg\phi \mid \texttt{P}_{\bowtie p}[\psi] \\ \psi &::= \ \texttt{X}\, \phi \mid \phi\, \texttt{U}^{\leq k}\, \phi \end{aligned}$$

where $p \in [0, 1]$, a is a Boolean expression with no referring to any clocks, and $\bowtie \in \{min,\ max\}$.

The following PCTL queries can be used to verify properties on PTP:

1) $\texttt{P}_{\bowtie=?}[\text{F } a]$,
2) $\texttt{P}_{\bowtie=?}[\text{F}_{\leq T}\ a]$,

where T is an integer expression. The first query questions about the maximum/minimum probability that a can be satisfied, while the second query questions the probability that a is satisfied within the time-bound T. These queries allow us to compute the maximum/minimum probability of reaching all target states satisfying a within a bound T or without time limit. For example, one can ask what is the maximum probability of an autonomous vehicle running to a given location within a specific time limit. A case study to clarify this approach will be given in the next section.

VII. Case Study

In this project, we are interested in testing the system in a parking lot scenario. Fig. 4 illustrates the proposed scenario for the verification of agent's decisions where the AV is moving into a parking bay, at the same time, a pedestrian is walking towards the AV, and there is another vehicle going out of the parking bay near to the AV. Note that this motion plan model is only an example of a possible behave for the three object mentioned above. A different motion model may be constructed based on the proposed scenario, objects, etc.

To avoid the collision with the pedestrian and/or the other vehicle, the agent constructs a PTP module for the two moving objects and the AV itself to estimate the maximum probability of a collision with any of the other moving objects under the current motion plan of the moving objects and the AV. The motion plan contains the trajectory that the AV will follow. For demonstration purposes we discretised the trajectory by roughly one meter apart, we also discretised the possible pedestrian's and vehicle's trajectories.

In order for the rational agent to build a meaningful PTP module while the AV is moving, it has been formerly equipped with the possible behaviour for both the pedestrian and the other vehicle. As we mentioned before that our simulated vehicle is based on TATA ACE electric van for next stage hardware implementation of the proposed system on the real vehicle, hence it is essential to indicate that the electric vehicle is silent while driving and this makes it difficult to notice without visual contact.

Usually, when a pedestrian notice that he is walking towards a moving vehicle, he may slow down with a high probability. The pedestrian may also choose to stop at some point, 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 average speed. Regarding the other vehicle trying to get out from the parking bay, the driver may not notice the AV if it is distracted by other parked vehicles or it is located in the blind-spot of the driver sight. For the purpose of simulating a realistic scenario, and to equip the agent with the possible behaviour of other objects, we used JAAD dataset [31] for pedestrians and drivers reactions to vehicles around them in different scenarios to get a better idea about how the objects may behave in such scenarios.

After the perception system captures the moving objects around through cameras, the AV start to measure the distances from those objects using the LiDAR sensor onboard and keep updating those distances instantly. The initial PTP module for the AV's behaviour is shown in Fig. 6. We use (x_1, y_1) to represent the coordination of the AV while the centre point is $(0, 0)$. We assume that it takes more than one second and less than two seconds for the vehicle to move one-meter distance. We do not take for granted that the vehicle moves at a constant speed.

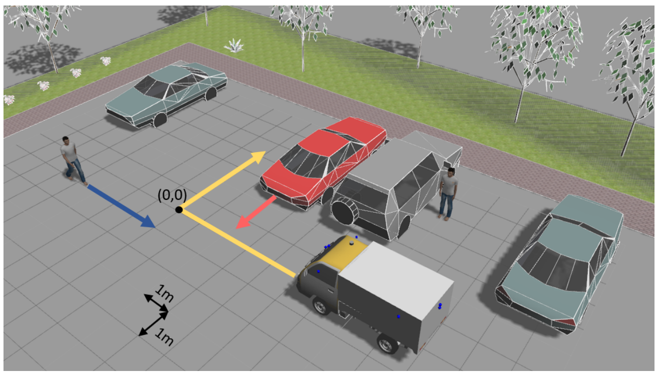

Fig. 4. Parking the AV in a parking lot

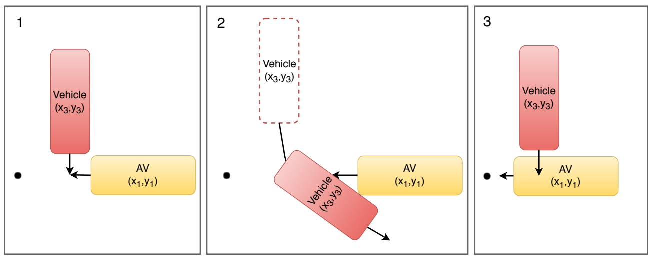

Fig. 5. A simple demonstration for three possible cases for collision

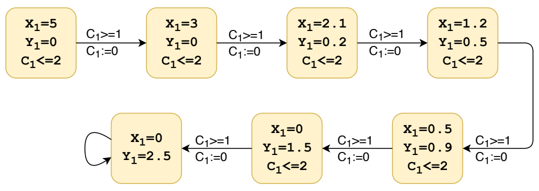

Fig. 6. Initial PTP module for the AV's behaviour

Fig. 7 shows the PTP module for the pedestrian's possible behaviour. We assume that the average speed of the pedestrian is the same as the speed of the AV inside the parking lot. The pedestrian can halve their speed after noticing the moving vehicle. We also assume that the probability of changing from the average speed to the reduced speed is 0.6, from the average speed to stop is 0.1, maintaining the average speed is 0.3, maintaining the reduced speed is 0.7,

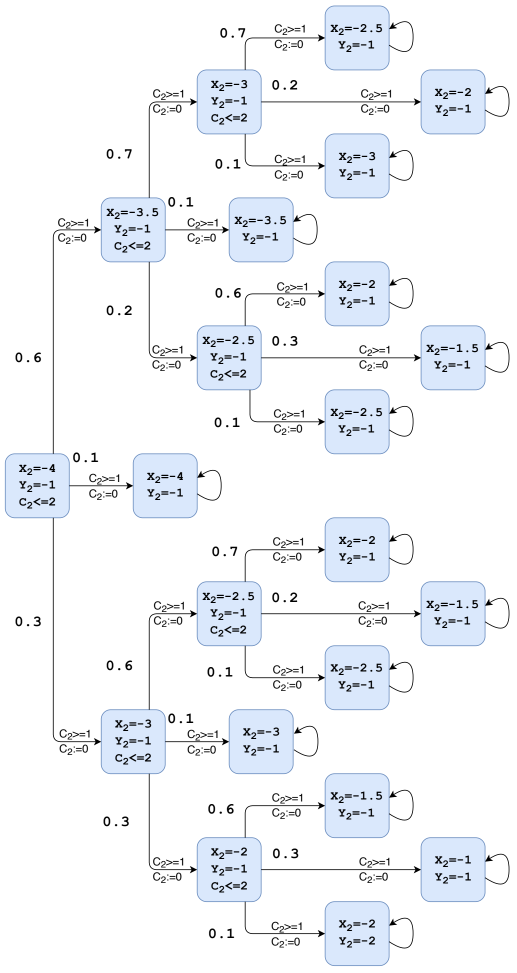

Fig. 7. PTP module for the pedestrian's behaviour

from the reduced speed to stop is 0.1 and from the reduced speed to the average speed is 0.2.

Fig. 8 shows the possible PTP module for the other vehicle trying to get out of the parking bay. We assumed that the driver may notice the AV and/or the pedestrian behind the vehicle with probability 0.5 hence he will stop till the space become empty, with a probability of 0.5 the driver may not notice them so he may move back then when he gets a better visual contact with the other objects around he will either stop with a probability of 0.4 or return to his first place with 0.4 probability or he may think that the other objects are still far and decide to continue his way out with a probability of 0.2, and so on. The agent will then modify the AV's PTP module according to the instant behaviour of the other objects around. Fig 5 demonstrate the possible cases that could lead to a collision between the AV and the other vehicle as explained by the PTP modules of both the AV and the vehicle.

Note that the parameters used in the PTP module, such as the speed and probability, may not reflect the exact behaviour of the AV, the pedestrian and the other vehicle. The agent is building those PTPs based on the instantaneous position of the moving objects, the instant speed of each object will be

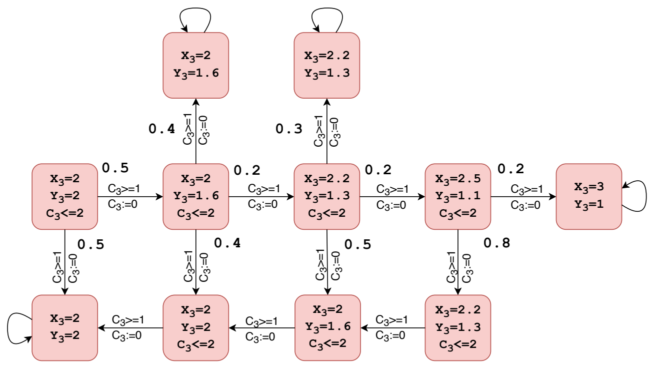

Fig. 8. PTP module for the vehicle's behaviour

TABLE I
TABLE 1: VERIFICATION RESULTS FOR THE PROPOSED SCENARIO

PTP Module	States	Transitions	Choices	Ver. time	Result
Pedestrian	590	1665	1311	0.035s	0.8399
Car	103	273	251	0.007s	0.5

considered in future development of the system. In general, this framework will help to predict a possible behaviour for the different objects around the AV, and this will help in reducing the possibility of collision. More accurate PTP modules could be obtained later after collecting more data through real tests.

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

$$\phi \equiv (x_1 - x_2)^2 + (y_1 - y_2)^2 >= 4. \tag{2}$$

$$\phi \equiv (x_1 - x_3)^2 + (y_1 - y_3)^2 >= 4. \tag{3}$$

Where (x_2, y_2), represent the coordinate of the pedestrian, (x_2, y_2) is the coordinate for the vehicle. Fig. 9 illustrate part of the Jason program used to generate PRISM code. As PRISM cannot deal with real numbers, we multiple the distance by 10. We compute the maximum probability of the violation of Equation (2,3), by the following property:

$$\mathrm{P}_{max=?}[\mathrm{F}\ \neg\phi]. \tag{4}$$

Due to the discretisation of the trajectory, the negation of Equation (2) is translated into the following expression:

$$((x_1 - x_2) \leq 38 \wedge y_1 = 0) \vee ((x_1 - x_2) \leq 38 \wedge y_1 = 2) \vee$$
$$((x_1 - x_2) \leq 37 \wedge y_1 = 5) \vee ((x_1 - x_2) \leq 35 \wedge y_1 = 9) \vee$$
$$((x_1 - x_2) \leq 31 \wedge y_1 = 15) \vee ((x_1 - x_2) \leq 19 \wedge y_1 = 25).$$

While the negation of Equation (3) is translated into:

$$((x_1 \geq x_3 \wedge x_1 - x_3 \leq 6) \wedge y_3 > y_1 \wedge (y_3 - y_1 \leq 7)) \vee$$
$$((x_1 \geq x_3 \wedge x_1 - x_3 \leq 10) \wedge y_1 > y_3 \wedge (y_1 - y_3 \leq 6)) \vee$$
$$((x_1 \leq x_3 \wedge x_3 - x_1 \leq 10) \wedge y_3 > y_1 \wedge (y_3 - y_1 \leq 4)).$$

The verification results is shown in Table I returned from PRISM for Formula (4), 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. The agent has to modify its plan (PTP module) to reduce that probability. The modification can be done immediately by choosing a different motion plan, or at a later stage if the objects are still far because the agent has to monitor the progress of other objects and the PTP module will frequently be updated using new positions of the AV, and the other moving objects to evaluate the situation.

```
    PERCEPTION PROCESS
    Monitor the following booleans:
    //Percepts
    Free parking lot detected. {[],[0,5]}
    Moving object detected.
        {[],[-4,-1],[2,2]}
    I am at global waypoint.
    Parking lot not fully explored.
    Generate PTP for moving object. {[I am at
          global waypoint],[5,0]}
    ...
    EXECUTABLE PLANS
    ...
    //Plan 5
    If ^[pedestrian detected] while ^[
        distance more than 4m] and ^[object
        get closer] then
    [Activate slow mode.]
    [Generate set of waypoints.]
    +^[object PTP generated]
    [Activate drive mode.].
    ...
    //Plan 9
    If ^[vehicle detected] while ^[distance
        less than 4m] then
    [Activate stop mode.]
    [Generate set of waypoints.]
    +^[object PTP generated]
    [Activate drive mode.].
```

Fig. 9. Part of the Jason program used to generate PTP modules.

VIII. CONCLUSION

This paper has presented the design and implementation of a framework for run-time verification of decisions made by a software agent onboard an AV. The structure includes a suitable configuration of sensors to sense the surrounding environment with a number of skills to control the movement of the AV. A rule-based BDI agent has been implemented to guide the AV to determine its moves.

To verify the actions made by the onboard agent controlling the vehicle, and to equip that agent with real-time verification of its plans, the PRISM model checker was used based on PTP modules to make sure that all combination of events will be thought of by the agent (in real-time) and verified before deciding its next move. The relatively small size of PTP modules of the AV and other objects and the limited number of those objects make it possible for run-time verification of AV's actions.

The simulation results for the proposed framework and the verification method shows the possibility of real-time verification for such application. This will be further investigated during future work of a hardware implementation of the proposed system on a TATA ACE electric vehicle. We will also equip the agent with extended, more challenging (yet possible) traffic behaviours for verification.

Acknowledgements

This work was supported by the Higher Committee for Education Development in Iraq (HCED) and SysBrain Ltd, UK. A special thanks to Suresh Arikapudi from TATA Motors for his valuable comments and suggestions on this project.

References

[1] J. Bohren, T. Foote, J. Keller, A. Kushleyev, D. Lee, A. Stewart, P. Vernaza, J. Derenick, J. Spletzer, and B. Satterfield, "Little ben: The ben franklin racing team's entry in the 2007 darpa urban challenge," *Journal of Field Robotics*, vol. 25, no. 9, pp. 598–614, 2008.

[2] T. Luettel, M. Himmelsbach, and H.-J. Wuensche, "Autonomous ground vehiclesconcepts and a path to the future," *Proceedings of the IEEE*, vol. 100, no. Special Centennial Issue, pp. 1831–1839, 2012.

[3] M. Wooldridge and N. R. Jennings, "Intelligent agents: Theory and practice," *The knowledge engineering review*, vol. 10, no. 02, pp. 115–152, 1995.

[4] P. Izzo, H. Qu, and S. M. Veres, "A stochastically verifiable autonomous control architecture with reasoning," in *55th IEEE Conference on Decision and Control, CDC'16*. IEEE, 2016, pp. 4985–4991.

[5] Y. Shoham, "Agent-oriented programming," *Artificial intelligence*, vol. 60, no. 1, pp. 51–92, 1993.

[6] F. Von Hundelshausen, M. Himmelsbach, F. Hecker, A. Mueller, and H.-J. Wuensche, "Driving with tentacles: Integral structures for sensing and motion," *Journal of Field Robotics*, vol. 25, no. 9, pp. 640–673, 2008.

[7] K. B. Isa and A. B. Jantan, "An autonomous vehicle driving control system," *International Journal of Engineering Education*, vol. 21, no. 5, p. 855, 2005.

[8] N. Lincoln, S. M. Veres, L. A. Dennis, M. Fisher, and A. Lisitsa, "An agent based framework for adaptive control and decision making of autonomous vehicles." in *ALCOSP*, 2010, pp. 310–317.

[9] R. H. Bordini, J. F. Hübner, and M. Wooldridge, *Programming multi-agent systems in AgentSpeak using Jason*. John Wiley & Sons, 2007, vol. 8.

[10] F. L. Bellifemine, G. Caire, and D. Greenwood, *Developing multi-agent systems with JADE*. John Wiley & Sons, 2007, vol. 7.

[11] A. S. Rao and M. P. Georgeff, *Formal models and decision procedures for multi-agent systems*. Australian Artificial Intelligence Institute Melbourne, Australia, 1995.

[12] H. V. D. Parunak, "Practical and industrial applications of agent-based systems," *Environmental Research Institute of Michigan (ERIM)*, 1998.

[13] R. Hoffmann, M. Ireland, A. Miller, G. Norman, and S. Veres, "Autonomous agent behaviour modelled in prism–a case study," in *International Symposium on Model Checking Software*. Springer, 2016, pp. 104–110.

[14] S. Kammel, J. Ziegler, B. Pitzer, M. Werling, T. Gindele, D. Jagzent, J. Schröder, M. Thuy, M. Goebl, F. v. Hundelshausen, *et al.*, "Team annieway's autonomous system for the 2007 darpa urban challenge," *Journal of Field Robotics*, vol. 25, no. 9, pp. 615–639, 2008.

[15] P. Falcone, F. Borrelli, H. E. Tseng, J. Asgari, and D. Hrovat, "A hierarchical model predictive control framework for autonomous ground vehicles," in *American Control Conference, 2008*. IEEE, 2008, pp. 3719–3724.

[16] L. E. Fernandes, V. Custodio, G. V. Alves, and M. Fisher, "A rational agent controlling an autonomous vehicle: Implementation and formal verification," *arXiv preprint arXiv:1709.02557*, 2017.

[17] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, vol. 3, 2009.

[18] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3. IEEE, 2004, pp. 2149–2154.

[19] S. M. Veres, "Natural language programming of agents and robotic devices," 2008.

[20] M. Kwiatkowska, G. Norman, and D. Parker, "A framework for verification of software with time and probabilities," in *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 2010, pp. 25–45.

[21] K. Dräger, M. Z. Kwiatkowska, D. Parker, and H. Qu, "Local abstraction refinement for probabilistic timed programs," *Theor. Comput. Sci.*, vol. 538, pp. 37–53, 2014.

[22] M. Kwiatkowska, G. Norman, and D. Parker, "Prism 4.0: Verification of probabilistic real-time systems," in *Computer aided verification*. Springer, 2011, pp. 585–591.

[23] S. Behere, F. Asplund, A. Söderberg, and M. Törngren, "Architecture challenges for intelligent autonomous machines," in *Intelligent Autonomous Systems 13*. Springer, 2016, pp. 1669–1681.

[24] A. R. Khairuddin, M. S. Talib, and H. Haron, "Review on simultaneous localization and mapping (slam)," in *Control System, Computing and Engineering (ICCSCE), 2015 IEEE International Conference on*. IEEE, 2015, pp. 85–90.

[25] T.-N. Nguyen, B. Michaelis, A. Al-Hamadi, M. Tornow, and M.-M. Meinecke, "Stereo-camera-based urban environment perception using occupancy grid and object tracking," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 1, pp. 154–165, 2012.

[26] M. Schreier, *Bayesian environment representation, prediction, and criticality assessment for driver assistance systems*. Technische Universität Darmstadt, 2016.

[27] "senglish publisher for eclipse," 2011, http://www.sysbrain.com/cognitive_agent_toolbox/sEP_User_Manual.pdf.

[28] "Cognitive agents toolbox," 2017, http://www.sysbrain.com/cognitive_agent_toolbox/, 25/09/2017.

[29] S. M. Veres, L. Molnar, and N. K. Lincoln, "The cognitive agents toolbox (cat)-programming autonomous vehicles," 2009.

[30] M. Y. Hazim, H. Qu, and S. M. Veres, "Testing, verification and improvements of timeliness in ros processes," in *Conference Towards Autonomous Robotic Systems*. Springer, 2016, pp. 146–157.

[31] A. Rasouli, I. Kotseruba, and J. K. Tsotsos, "Agreeing to cross: How drivers and pedestrians communicate," in *Intelligent Vehicles Symposium (IV), 2017 IEEE*. IEEE, 2017, pp. 264–269.