# An Approach to Simulate Autonomous Vehicles in Urban Traffic Scenarios

Miguel C. Figueiredo, Rosaldo J. F. Rossetti, *Member*, *IEEE*, Rodrigo A. M. Braga, Luis Paulo Reis
Department of Informatics Engineering
Artificial Intelligence and Computer Science Lab.
Faculty of Engineering, University of Porto
Rua Dr. Roberto Frias S/N, 4200-465 Porto, Portugal
{ee01193, rossetti, rodrigo.braga, lpreis}@fe.up.pt

*Abstract*—**The most common cause of traffic accidents is arguably the driver error due to lack of attention. And it is very unlikely this is going to change soon thanks to increasingly cell-phone usage, in-car entertainment systems, and naturally the more frequent traffic jams in highly populated areas. Autonomous vehicles, such as driverless cars, are a promising approach to decrease traffic accidents, as well as congestions. To test this approach, simulations are a safer, more efficient, and cheaper way than live testing. This paper presents an approach to implement a simulator to test such vehicles. It includes a study of the state of the art in driverless car simulation and discusses on the specific objectives that this particular simulator aims to achieve in order to aid testing the interactions of multiple driverless cars in urban networks.**

***Microscopic traffic simulation; autonomous driving; autonomous vehicles; autonomous agents; multi-agent systems.***

## I. Introduction

Autonomous vehicles are one of the possible solutions to our most common cause of traffic accidents: drivers' error due to lack of attention. According to several different studies, such as the one by L. C. Davis in 2004 [1], driverless cars will substantially decrease traffic accidents and traffic jams even if there is just a few of them driving among regular cars.

In the latest years, with computer technology advancing fast, simulation began to be used more regularly for this kind of projects. Simulations are safer, more efficient, and cheaper than live testing on real vehicles. They also allow testing more scenarios than those that would be possible with real world testing, as well as testing dangerous situations to involve humans. Therefore, testing complex systems in virtual worlds is the ideal solution to validate code quickly, with more possibilities, cheaply and with minimum risk.

Our main purpose with the present work is to specify a realistic simulator to aid the analysis of both autonomous and semi-assisted driving on networks of vehicular traffic, preferably in urban areas. We intend to include in the conceptualization of our platform as much functionality as possible, such as definition and calibration of sensors and actuators in vehicles, realistic representation of vehicle kinetics, 2D/3D visualization of simulation models, parameters and outputs, interaction between vehicles such as car-following, lane-changing, ramp-metering, and intersection-crossing models, integration with an agent-based microscopic traffic simulator, and support tools for editing and visualizing simulation models.

The remaining part of this paper is organized as follows. As existing robotic simulators are not fully appropriate when it comes to simulating driverless cars in scenarios with intense traffic, some further assumptions must be taken into consideration, as we will discuss in the following section. The approach herein presented aims to enhance an existing platform to overcome such drawbacks. Section three will explain the details of the software architecture for the proposed approach. Finally, in section four we will draw conclusions on this preliminary work and discuss on further developments and potentials for future research.

## II. State of the Art

### A. *Types of Simulators*

Simulators that are related to the study of traffic and autonomous vehicles are currently distinguished into two types: large scale traffic simulators, and small scale robotic simulators. In the former case, large scale traffic simulators are used to represent traffic flow over very large and/or complex road networks. In these simulators the movement of each car is so simplified that it is not possible to distinguish between human drivers and robot drivers. This is also due to the behavior of drivers, which is simplified as well, in order to allow simulations of thousands of vehicles to be run in a computer with reasonable resources, and within a reasonable timeframe. One example of such simulators is the MAS-T$^2$er Lab's microscopic traffic simulator [2].

However, it should be noted that these simulators are not detailed enough to simulate autonomous cars when it comes to their kinematic behavior, sensors and actuators operation, interaction with surroundings, and so on. The physics in these simulators are over-simplified to the point that the movement of cars that are changing lanes is not continuous. In other words, a car is either in lane *x* or lane *y*. There is no state where a car is partially occupying both lanes, as if it were moving along the horizontal marks dividing adjacent lanes.

In order to simulate and test the performance of a driverless car, a more detailed simulator is required though, such as a small scale robotic simulator. A lot of sensor and actuator details is needed if we want to know how will a car throttle, break, and steer while reacting to various stimuli and obstacles ranging from walls and other cars, to people, animals, sidewalks, and so forth. Collision detection is also required, so as it is possible to verify whether the driverless car is malfunctioning to the point of causing an accident. We might want to know which of the sensors are detecting (or not) each obstacle, and try to identify errors in its design (both hardware and software) from the tests. A large scale traffic simulator cannot provide us with such details.

A look has been taken at the DARPA Urban Challenge teams and what tools they used to simulate their cars before live-testing [3, 4, 5, 6, 7, 8, 9, 10]. To be successful in such a competition, the use of a simulator is basically unavoidable. Overcoming safety concerns and strict time constraints is a must here then. And testing in the virtual world is the ideal solution to validate code quickly and with minimum risk. The teams were also able to test situations too dangerous to involve humans and test more scenarios than it would have been possible in the real world. Simulation was used to find obvious problems with their software, but this was always followed by testing on the vehicle, as well.

## *B. Characterization of Robotics Simulators*

These are some features used as a means to compare the different robotic simulators. Some of which are more important than others, depending on the field and interest of study. Below we comment on the ones that are more or less useful when it comes to testing driverless cars.

3D visual simulation is important. Some simulators run full 3D physical simulations. The calculations become much more complex and resource-consuming, but this results in a more realistic interpretation of the model. Simulators featuring 3D visualization allow the user to observe and better understand the events happening in the simulation by animating detailed 3D graphics and models to represent the different elements of the simulation. On the other hand, handling large scale traffic volumes is also a requirement. These simulators are able to manage large amounts of elements with very simple behaviors and physics in order to reproduce large-scale phenomena in a reasonable timeframe and using reasonable computer resources. Multi-agent simulation then has been adopted, allowing some simulators to have multiple and independent autonomous agents interacting in the same simulation. The reason for these simulations can be simply to test how agents react to one another, but they also enable testing either cooperative or competitive behaviors.

Vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communications have equally become an imperative technology in today's transportation systems. Simulators that can simulate communications between agents will allow for messages to be exchanged between agents, and may simulate physical restrictions like the broadcasting radius of a certain robot. Also, collision detection is a very basic feature that is implemented in almost every simulator available. Because the point is to test for anomalies and undesirable events, and a collision is the most common of such events, independently of the field of study, this is also an imperative feature.

As for sensors, simulators that are able to calculate random noise at the outputs of sensors will allow for more realistic testing of the decision making mechanisms that need to deal with the flawed nature of real sensors. Failure simulation is another feature that will help testing whether robots are fail-safe or fault-tolerant. This is the case when the simulator has the ability to corrupt, or completely suppress, the information coming from sensors to the agents, or from the agents to actuators. Also, sensors can be affected by the environment. Indeed, harsh weather conditions and hazardous terrains can affect sensors in various ways, for example, fog or darkness affecting the visibility of an optical camera, or intense weather causing echoes in laser scanners. Simulators might include these factors in the calculation of sensor values then. Similarly, the environment can also affect vehicles' physical behavior. Weather and ground conditions can affect the performance and control of vehicles in various ways, for example, loose gravel, rain, or snow making roads more slippery.

It is also important to consider that sensors can behave differently and be used for a wide range of purposes. We discuss on a few of them next. For instance, a GPS receiver is able to determine vehicles' current location, time, and velocity, thanks to the precise microwave signals transmitted by a constellation of between 24 and 32 Medium Earth Orbit satellites. Luminosity sensors used to simply detect the amount of light, being useful for knowing whether a car needs to turn the lights on (e.g. due to the nightfall or tunnels). Optical cameras, of many different types, send a stream of images for the robot to analyze things like movement flow, colors, and so on. Infra-red cameras are essentially the same as a normal optical camera, except that it detects light in the infra-red spectrum instead of the human visible spectrum, and thus are also known as night-vision cameras. Laser scanners (LIDAR), on the other hand, emit a laser that is constantly changing its angle, while listening to the laser reflections. This results in an array of points where the laser hit a target and got reflected. This happens as fast as 12.5 times per second in a typical one. If used correctly, it can measure the distance, size, shape and speed of multiple obstacles several times per second. Ultrasound sensors are typically used to measure short distances in a wide angle, whereas infrared ones are used to measure short distances in a sharp angle. An inertial measurement sensor is the main component of inertial guidance systems used in air-, space-, and watercrafts. It works by sensing motion (including the type, rate, and direction of that motion) using a combination of accelerometers and gyroscopes. The data collected from these sensors allows a computer to track a craft's position, using a method known as dead reckoning. Radars emit either microwaves or radio waves that are reflected by the target and detected by a receiver, typically in the same location as the transmitter. Although the signal returned is usually very weak, the signal can be amplified. This enables radars to detect objects at ranges where other emissions, such as sound or visible light, would be too weak to detect. Finally, the simple speed sensors that exist in every car, connected to the speedometer, let drivers know the instant speed. They are usually based on watching how many times the wheels of a car complete a revolution in a given period of time.

### *C. Today's Robotics Simulators*

We now compare the features of different robotic simulators, such as those presented in [11, 12, 13]. Some teams that participated in the DARPA Urban Challenge 2007 will also be compared, both in terms of their simulators' features and sensors their cars had been equipped with. A generic game engine with typical features will be compared as well, since with some modifications game engines are functional enough to allow real-time simulation of real applications. USARSim is an example of a simulator that was implemented by modifying a game engine, namely Unreal Engine, developed by Epic Games enterprise [14].

Thus, we want to compare game engines with generic robotic simulators and with simulators used by DARPA Urban Challenge's teams, as presented in Table 1. Also, a concise comparison of sensors used by teams' driverless cars is presented in Table 2. To the best of our effort, detailed information about the different simulators that each team used was hard to find, and so some of these features are marked as not determined, at the time of writing this paper.

TABLE I. FEATURES OF TODAY'S ROBOTIC SIMULATORS

**Table Legend:**
**x** – Yes
**m** – Yes (considering modifications known to have been already made)
**x?** – Could not be verified, and the information was not found, but assumed as being a positive
**?** – Could not be determined at the time of writing this paper
**blank** – No

| | Simulator features | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 3D simulation | 3D visualization | Large scale traffic | Multi Agent simulation | V2V / V2I communication | Collision detection | Sensor noise | Failure simulation | Environment affecting sensors | Environment affecting physics |
| Game engines | | | | | | | | | | |
| Generic | x | x | x | | | x | | | x | x |
| DARPA Teams' simulators | | | | | | | | | | |
| 1st place - "Boss" | x? | x? | | | | x? | | | | |
| 2nd place - "Junior" | x? | x? | | | | x? | | | | |
| 3rd place - "Odin" | x? | x? | | | | x? | | | | |
| "Talos" | x? | x? | | | | x? | | | | |
| "Little Ben" | x? | x? | | | | x? | | | | |
| "Skynet" | | | | | | x? | | | | |
| PAVE (Princeton's team) | x? | x? | | | | x? | | | | |
| Generic robot simulators | | | | | | | | | | |
| Ms Robotics Studio | x | x | | x | ? | x | | | | |
| Webots | x | x | | x | ? | x | | | | |
| Ciber-Rato | | | | x | m | x | x | | | |
| USARSim | x | x | | | | | x | | | |
| ÜberSim (Robot Soccer) | | x | | | | x | | | | |
| EyeSim (Robot Soccer) | | x | | x | | x | x | | | |

It is important to remember, at this point, that large scale traffic simulators are insufficient when it comes to simulation detail. They lack all features mentioned in the previous section, except for that of *large scale traffic* (and, in some cases, *3D visualization*), so they were not included in Table 1. And since they do not simulate sensors at all, they were not included in Table 2 either.

The observation of Table 1 reveals that most simulators nowadays lack important features for the simulation of autonomous vehicles. Robotic simulators do not seem to simulate large scale traffic or pedestrians. However, we want to simulate how a driverless car responds to crowded areas, or traffic jams. As V2V/V2I communications are important, we want to simulate how effective these communications can be between vehicles and the road network infrastructure, as well as how can they improve operations. Different sensors, as well as their behavior and associated noise characteristics must be also accounted for. Similarly to sensors, actuators can fail and for some reason stop working properly. Therefore, sensors and actuators tolerance to faults must be modeled as well.

Some of the teams that participated in the DARPA Urban Challenge implemented their own simulators, while others used already existing ones. Some of their own simulators had very practical features that are worth mentioning, as summarized in [3]. For instance, the simulator used by Princeton's team allowed the user to test their code in the simulator and then transfer it to the vehicle without the need of recompiling it. MIT's simulator could play back data recorded in real life test runs, but the simulated obstacles reflected perfect data during those recorded runs, something that actual sensors did not obtain during real life test runs. CarOLO also used their simulator to test new software implementations before adding them to the vehicle, as well as confirming bugs found during real world tests. That is something that previous teams also did. On the other hand, further development of this simulator has yielded a version in which multiple instances of their autonomous vehicle could be operated. In doing this, their software could learn efficient driving behavior in an environment in which multiple traffic vehicles may exist. Additionally, different versions of code could be run from the same starting point, running the same mission file, in order to compare their performances. Tartan's simulator also had the ability to add virtual obstacles to a real world environment during testing. Therefore, the vehicle was led to think there were obstacles within its path, causing avoidance strategies to be executed, even though there were none actually. All the aforementioned features are very time-saving when simulating and testing, and most of them were not easy to implement. They were developed for these simulators because time was the most important factor during the DARPA Urban Challenge, since the teams only had a few hours to update and validate their code between events.

In Table 2, we compare which sensors are simulated by the generic robot simulators, as well as the sensors equipped in DARPA's driverless cars. After a quick observation, it is possible to notice that generic car game engines are not good enough because they are simply not sensor-based, and they also lack some important simulation features such as the ability to use multiple autonomous agents, V2V/V2I communication, and represent sensors noise. Without profound modifications they lack exactly what real projects, such as DARPA Teams', need to simulate how cars will behave before testing them in real scenarios.

TABLE II. SENSORS SIMULATED IN TODAY'S ROBOTIC SIMULATORS

**Table Legend:**
**x** – Yes
**m** – Yes (considering modifications known to have been already made)
**x?** – Could not be verified, and the information was not found, but assumed as being a positive
**?** – Could not be determined at the time of writing this paper
**blank** – No

| | Simulator Sensors | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | GPS | Luminosity | Optical camara | Infra-red camera | Laserscanner (LIDAR) | Ultrasound | Infrared | Inertial Measurement | Radar | Speed |
| DARPA Teams' simulators | | | | | | | | | | |
| 1st place - "Boss" | x | | | | x | | | x | x | x |
| 2nd place - "Junior" | x | | x | | x | | | x | x | x |
| 3rd place - "Odin" | x | | x | | x | | | x | | x |
| "Talos" | x | | x | | x | | | | | x |
| "Little Ben" | x | | x | | x | | | | | x |
| "Skynet" | x | | x | | x | | | | | x |
| PAVE (Princeton's team) | x | | x | | | | | | | x |
| Generic robot simulators | | | | | | | | | | |
| Ms Robotics Studio | | | | | x | | | | | ? |
| Webots | x | x | x | | | x | x | x | | ? |
| Ciber-Rato | x | | | | | x | | | | ? |
| USARSim | | | | | x | x | | | | ? |
| ÜberSim (Robot Soccer) | | | | | | | | | | ? |
| EyeSim (Robot Soccer) | | | x | | | | x | | | ? |

## D. *Limits and Specialization of Simulators*

Even after so much effort in the development of simulation environments, we are still far away from having an ideal simulator. The more features they tend to have and the more realistic they tend to be, the more resources they need to perform their calculations in a reasonable period of time. But simulation will not help much if it is oversimplified. In other words, we need a trade-off between the realism of a simulation, and the simplicity of its calculations. There are lots of things that we would love to be able to simulate, but the sheer amount of processing power those would require are simply too exacerbated. Therefore, it is also important to take into consideration limits and handicaps of simulators and modeling methodologies.

Care must be taken with quantities. When we are talking about large amounts of characteristics and elements within a simulation, they better be simple. Very frequently, it is necessary to perform both complex and simple calculations repetitively. However, complex calculations tend to be avoided most of the time. For such an example, let us look at a large scale traffic simulator. The behavior of those thousands of vehicles is simplified to the point the simulator is able to make thousands of calculations almost instantly. That is because the decisions are simple, and their physics and movement are kept very simple as well. On the other hand, imagine if each one of these cars were aware of its environment through individual sensors, affected by generated noise, and all these cars went through complex decision-making algorithms in order to send commands to their actuators, as well. At this point then, the simulation should have to calculate the next step based on what the actuators were ordered to do. If one multiplies all these details by a thousand cars, it would take days to simulate just a few seconds (if not less) of such a large scale scenario.

Nonetheless, it is very easy for a normal computer to simulate a few of these driverless cars in real time. So, the most obvious solution is to try and have more detail where it matters and less detail where it does not, depending on the specialization of the simulator. This means, in the case of this project, that we might try and have some detailed cars, with detailed decision-making mechanisms and a more detailed physics simulation to model the driverless entities, and surround them with a crowd of less detailed objects, with their very own simple simulation model.

As for 3D simulation, it becomes a bit of a problem here, specially if one is adding a third dimension to the terrain. That is because in this case, there are a whole lot more calculations involving those thousands of cars, to account for the extra dimension in their dynamics. For highway systems and most road networks, the changes in elevation are relatively small. Therefore, the simulated results are similar even without a 3D simulation. But with some cities and road networks, there are very drastic elevation changes. A 3D simulation is very important in these cases because it must be taken into account that sensors cannot always look around corners, or down hills. The vehicle performance is affected by the third dimension as well. It needs more power to climb uphill and, more importantly, it needs to break sooner and harder if it is going downhill. Weather conditions such as rain and snow also affect the way the vehicle performs. Dirt, loose gravel, and other different ground surfaces can also be simulated to test the vehicle's control system. Again, a normal computer can simulate effects like these for a few cars, but not for thousands.

A good example of something that would be very hard to realistically implement in a simulation is GPS signal loss. The satellite signal can be occluded by pretty much anything that is big enough and in the way, such as buildings or trees. Realistic simulation of such a blockage would require simulation of the satellites and their orbits. The line of sight between the GPS receiver and the satellites would need to be tested to see whether there was a signal loss or not. One can go as far as considering signal reflections off large buildings. But all these details would take a tremendous amount of processing power, and a lot of effort to implement. A trade-off between the effort and the results is essential then. We can try and simulate GPS signal loss in various ways, from simple random time intervals, to specify areas in the map where the GPS signal would be lost. As long as the result is that the GPS receiver looses the signal every once in a while, we get a simulator that can test the vehicle's ability to roughly predict its position even if it temporarily looses the GPS signal.

There is also a good example of something that was very hard to simulate a few years back and now, with some developments in computer technology, has become easy on processing needs. That is the example of vision based sensors. Vision algorithms used to be tested by simulating the markings on the roads, but the results differed a lot from real life, where there are shadows from objects that might be out of the picture, atmospheric conditions such as fog, storms, or even direct sunlight. Vision algorithms can now be tested more realistically because 3D rendering has evolved a lot lately. Now it is very common to have applications (any common video-game) that has a 3D engine implementing all these features, and can render them all in real time with a reasonably cheap computer. It is a matter of streaming the result video output into a virtual camera instead of streaming it into a monitor screen. This way it would be possible to emulate a camera receiving a video that features photorealistic weather effects [14].

Then, an ideal simulator would have to be indistinguishable from the real world. That might only be accomplished in a utopian future. But we can take what exists and adapt it to our necessities when it comes to simulation, by trying and re-balancing the capabilities of the simulator.

## III. The Proposed Approach

### A. *Software Architecture*

Our approach is based on the simulator architecture depicted in Fig. 1, where the different modules of the project are represented. The two grayed out blocks in the center of the figure are the simulation modules which will be the main focus and the point of start, namely the simulator and the simulation viewer.

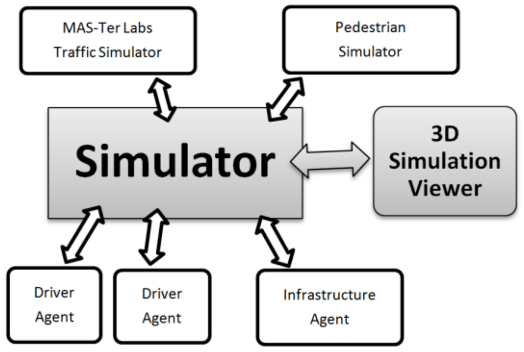

Fig. 1. Architecture of the simulator and its bordering systems.

The details of the connections between the different software pieces are as follows. The 3D Simulation Viewer will connect to the simulator, and receive information from it to render the 3D representation of the simulation. It should also have the functionality to send information of rendered images back to the simulator for optical sensors such as cameras (to test vision based algorithms). The simulator connects to the MAS-T$^2$er Lab's microscopic traffic simulator and to the pedestrian simulator. The main simulator will receive traffic information from both the simulators mentioned above, and will include that information in its calculations. It should also be able to send information of its agents back to those simulators, so that they, in turn, can take those agents into consideration when calculating the outcomes of their own simulations. Agents will connect to the simulator and send all the information it takes for the simulation to know the necessary characteristics about the agents. This includes component positioning and type of agent. It can be a driver agent, in which case it will have a car assigned to it, or it can also be an infrastructure agent, such as traffic lights.

### B. *Specific Features*

Below, the specific features are briefly described, grouped according to their role within the framework. These objectives were reached after comparing the identified problems as we had discussed in the beginning of this paper, as well as the general review of the state of the art to figure out what is missing in the simulators studied.

#### 1) *Simulator*

Collision detection – it is the simulator ability to detect collision between objects; Map reformatting – it is related to modifying the road network map to add more details for the simulation; V2V/V2I communication controlled by the simulator – it is the simulator ability to authenticate and forward messages between agents, to simulate V2V and V2I communications; Environment affecting physics and/or sensors – this is the simulator ability to take the environment into consideration when calculating sensor values and object movements; Failure simulation – it is the simulator ability to simulate failures (e.g. not sending information to a sensor, or sending corrupt data, ignoring actuator commands, etc.); Sensor noise – it is the simulator ability to add noise to the sensor values; Communications degradation – it is the simulator ability to limit communications between the agents depending on the environment.

#### 2) *Agents*

Agents interface with the simulator – it is the simulator ability to let agents to be connected to it; Agents-actuator interfaces – it is the simulator ability to receive actuator values from the agents; Agents-sensor interfaces – it is the simulator ability to send sensor values to agents; Game type agent – this is related to the ability of implementing a simple agent that does not necessarily have decision-making abilities, in other words is not autonomous. This agent will have an interface with the user so that he/she can directly control a vehicle within the simulator. This will come handy for testing and debugging the simulator; Agents-traits interfaces – it is the simulator ability to receive the traits information from an agent, namely information regarding their sensors, actuators and physical traits; Real vehicle type agent – This is the functionality that allows a real vehicle to connect their sensors and reporting as an agent; Real vehicle reacting to virtual sensor responses – basically, it is the functionality of having a real vehicle avoiding virtual obstacles; Infrastructure agents – ability for an agent to connect itself to the simulator, functioning as a simple road network infrastructure (e.g. a smart traffic light); Obstacles agent – ability for an obstacle agent to connect to the simulator, functioning as a simple obstacle such as a broken vehicle or emergency traffic signs.

#### 3) *3D Viewer*

Implementation of a 3D Simulation viewer – it is related to the 3D graphics rendering library to implement the 3D viewer that will be used to render the simulation scenes (e.g. OpenGL); 3D simulation viewer interface with the simulator

– this is basically the ability of the 3D viewer to connect to the simulator and receive information from it to render the 3D scene; Range indicators for sensors and communications – this is a feature that would render shapes to inform the user of various things such as sensor and communication ranges; 3D simulation viewer streaming images back to the optic sensors – it is the ability of the viewer to render images from a given point (an optical sensor) to stream back as information for the agents to test their vision based algorithms; Environment affecting visualization – it is related to rendering some environment features such as daylight, fog and darkness.

#### 4) *Traffic simulators*

Interface with microscopic traffic simulators – it is the simulator ability to connect to external microscopic traffic simulator, such as MAS-T$^2$er Lab's microscopic simulator; Import/export road network maps – it is the simulator ability to load a road network map from an external traffic simulator or any geographic information system (GIS) database; Traffic information interface – it is the simulator ability to receive traffic information from an external running microscopic traffic simulator; Agent information interface – it is the simulator ability to send agent information to an external traffic simulator so that it can account for the driverless car as a vehicle in the traffic flow; Pedestrian simulators interface – it is the simulator ability to connect to a pedestrian simulator; Pedestrian traffic information interface – it is the simulator ability to receive pedestrian flow information from a pedestrian simulator.

## IV. Conclusions

In this work we have proposed an architecture mixing different simulators to support autonomous vehicles simulation in a mixed environment. From previous experiences with robotics simulators and envisaging their potential to more realistically represent a driverless vehicle, we have defined an architecture that balances between simulating a few very realistic autonomous vehicles and too many simple cars within a microscopic traffic model.

Of course, starting a simulator from scratch is a daunting task. At least, when compared to the task of modifying an existing one to adapt it to our needs. And after careful analysis of the available simulators, Ciber-Rato [15] was selected to be the basis for our platform. It is a simulator that already implements many desired features for our simulator, as identified from section two. Besides, we are already somewhat familiar with it which, together with the fact of being an open source tool, turns it into a reasonable option. Additionally, its programming is structured very well, which makes it easy for us to adapt the environment to our own needs and implement the aforementioned features. Some extensions to its original framework already include interesting and useful features, such as those implemented in [16]. This version implements a modification that basically allows agents to communicate between each other, similar to what we intend to enable V2V and V2I communications in our own framework. Another modification of this simulator was implemented within the project IntellWheels, by members of our group, including an enormous amount of new features, compared to the original Ciber-Rato, to allow autonomous and cooperative wheelchairs to interact within a hospital environment. Besides different sensors and actuators, as well as a whole bunch of dynamics models, this enhancement includes a 3D viewer of their own, allowing for different wheelchairs to be connected to the simulator, with different sensors and characteristics [17].

Next steps include the adaptation of the MAS-T$^2$er Lab's microscopic traffic simulator [2] to support the interaction of its agents with external environments, in this case the simulator of our proposed framework.

## Acknowledgment

Authors gratefully thank Pedro Malheiro, Paulo Ferreira and Edgar Esteves for their help and support with the simulators they worked on and effort to integrate them within the proposed framework.

## References

[1] L. C. Davis, "Effect of adaptive cruise control systems on traffic flow," Physical Review E, vol. 69, no. 6, 2004.

[2] P. Ferreira, "Specification and Implementation of an Artificial Transport System," Master's Dissertation. Porto, Portugal: Faculty of Engineering, University of Porto, 2008.

[3] T. Alberi, "A Proposed Standardized Testing Procedure for Autonomous Ground Vehicles," Master's Dissertation, Blacksburg, VA: Virginia Polytechnic Institute and State University, 2008.

[4] "Stanford Racing Team," [Online]. Available: http://cs.stanford.edu/group/roadrunner/, last accessed Aug, 2009.

[5] "Ben Franklin Racing Team," [Online]. Available: http://www.benfranklinracingteam.org/, last accessed Feb, 2009.

[6] "MIT DARPA Grand Challenge Team," [Online]. Available: http://grandchallenge.mit.edu/, last accessed Aug, 2009.

[7] I. Miller and M. Campbell, "Team Cornell's Skynet: Robust Perception and Planning in an Urban Environment," Journal of Field Robotics, vol. 25, no. 8, pp. 493–527, 2008.

[8] "Cornell DARPA Urban Challenge," [Online]. Available: http://www.cornellracing.com/, last accessed Aug, 2009.

[9] "Carnegie Mellon Tartan Racing," [Online]. Available: http://www.tartanracing.org/, last accessed Aug, 2009.

[10] "VictorTango Urban Challenge," [Online]. Available: http://www.me.vt.edu/urbanchallenge/, last accessed Aug, 2009.

[11] "Microsoft Robotics Developer Center," [Online]. Available: http://msdn.microsoft.com/en-us/robotics/, last accessed Aug, 2009.

[12] B. Browning and E. Tryzelaar, "ÜberSim: A Multi-Robot Simulator for Robot Soccer," in 2nd International Joint Conference on Autonomous Agents and Multiagent Systems, 2003, pp. 948–949.

[13] T. Bräunl, "The EyeSim Mobile Robot Simulator," Report CITR-TR-58. Auckland, New Zealand: Computer Science Department, University of Auckland, 2000.

[14] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper, "USARSim: a robot simulator for research and education," in IEEE International Conference on Robotics and Automation, 2007, pp. 1400-1405.

[15] N. Lau, A. Pereira, A. Melo, A. Neves and J. Figueiredo, "Ciber-Rato: um ambiente de simulação de robots móveis e autónomos," Electronics and Telecommunication, DETUA Journal, vol. 3, no. 7, pp. 647-650, 2002. (in Portuguese)

[16] "Micro-Rato Competition," [Online]. Available: http://microrato.ua.pt/, last accessed Aug, 2009.

[17] P. Malheiro, "Intelligent Wheelchair Simulation," Master's Dissertation. Porto, Portugal: Faculty of Engineering, University of Porto, 2008.