Virtual Intelligent Vehicle Urban Simulator: Application to Vehicle Platoon Evaluation

Franck GECHTER, Jean-Michel CONTET, Stéphane GALLAND, Olivier LAMOTTE, Abderrafiaa KOUKAM

Laboratoire Systèmes et Transports,
Université de Technologie de Belfort-Montbéliard, Belfort, France
franck.gechter@utbm.fr, jean-michel.contet@utbm.fr, stephane.galland@utbm.fr,
olivier.lamotte@utbm.fr, abder.koukam@utbm.fr
http://www.multiagent.fr

Abstract

Testing algorithms with real cars is a mandatory step in developing new intelligent abilities for future transportation devices. However, this step is sometimes hard to accomplish especially due to technical problems,... It is also difficult to reproduce the same scenario several times. Besides, some critical and/or forbidden scenarios cannot be tested in real. Thus, the comparison of several algorithms using the same experimental conditions is hard to realize. Considering that, it seems important to use simulation tools to perform scenarios with near reality conditions. The main problem with these tools is their distance with real conditions, since they deeply simplify the real world. This paper presents the architecture of the simulation/prototyping tool named Virtual Intelligent Vehicle Urban Simulator (VIVUS). The goal of VIVUS is thus to overcome the general drawbacks of classical solutions by providing the possibility of designing a vehicle virtual prototype with well simulated embedded sensors and physical properties. Part experiments made on linear platoon algorithms are exposed in this paper in order to illustrate the similarity between simulated results and those obtained in reality.

Keywords: Sensors simulation, Physics simulation, Autonomous vehicle, Platoon systems, 3D modelling

1. Introduction

Testing algorithms with real cars is a mandatory step in developing new intelligent abilities for future transportation devices. However, this step is sometimes hard to accomplish especially due to hardware problem, vehicle disposability problem, etc. Moreover, it is also difficult to reproduce the same scenario several times since some experimental parameters are variable and uncontrollable such as perception conditions, for instance. Thus, it is hard to compare

several algorithms using the same experimental conditions. Besides, some critical and/or forbidden scenarios (i.e. scenario that implies the collision and/or destruction of a part of the physical device) are difficult to test in real.

Considering that, it seems important to find a less restrictive way to perform scenarios with near reality conditions. That's why many laboratories are testing their algorithms in simulation. The main problem of standard simulation tools is their distance with real conditions, since they generally simplify vehicle/sensors physical models and road topology. Virtual cameras are basically reduced to a simple pinhole model without distortion simulation and vehicle models do not take into account any dynamical characteristics and tyre-road contact excepted in specific automotive industry tools.

The participation of the SeT Laboratory to CRISTAL project [1] puts in evidence the fact that a new simulation/prototyping tool is required to perform a comparison between several solutions in the context of urban intelligent vehicles. For instance, in the context of the CRISTAL project, a comparison between linear platoon solutions has had to be performed. In order to obtain results near to reality, this tool must be as precise as possible in terms of sensors simulation and vehicle dynamical model.

This paper presents the global architecture of the simulation/prototyping tool named Virtual Intelligent Vehicle Urban Simulator[2] (VIVUS) developed by the SeT Laboratory. This simulator is aimed at simulating vehicles and sensors, taking into account their physical properties and prototyping artificial intelligence algorithms such as platoon solutions [1] and obstacle avoidance devices [2]. The goal of VIVUS is thus to overcome the general drawbacks of classical solutions by providing the possibility of designing a vehiclevirtual prototype with well simulated embedded sensors. The retained solution has several interestssuch as:

- Prototyping artificial intelligence algorithms before the construction of the first vehicle prototype. In this case, they can be developed, tested and tuned with a virtual prototype of a real vehicle.
- Testing critical and/or banned use cases, i.e., cases that imply partial or total destruction of vehicles, to draw the limits of the retained solutions.
- Testing and comparing several algorithms/solutions for embedded features with a low development cost. It can thus help to choose the future embedded devices related to retained solutions requirements (processing power needs, connectivity, etc.).
- Testing and comparing the sensor solutions before integrating them into the vehicle.

[1] Industrial Research Cell for Autonomous Light Weight Transportation Systems. In French: "Cellule de Recherche Industrielle en Systèmes de Transports Automatisés Légers"

[2] `http://www.multiagent.fr/Vivus_Platform`

- Integrating tests and evaluations results into the vehicle design process.
- Using informed and documented virtual reality to access the attribute and state values of the vehicle, its components, and the environmental objects around.

This tool has been especially used during the CRISTAL project to compare several linear platoon algorithms to the results obtained with a real vehicle. Parts of these experiments are exposed in this paper in order to illustrate the similarities between the simulation's and real car's results.
The paper is structured as follows. Next section presents a state of the art of the simulation applications with a comparison of them according to several criteria. Section 3 gives a description of VIVUS architecture focusing on physics model and virtual sensors description. Then Section 4 gives a short presentation of the target application, linear platoon evaluation, on which the developed simulator has been tested. Section 5 exposes results obtained comparing simulator and real vehicle experiments. Finally, the paper finishes with a conclusion and a short description of future work.

2. Simulation tools comparison

Computer simulation and their extension into robot and vehicle simulations are an emerging topic. The main motivation for using simulation tools is the potential to validate new technology before its deployment in real devices. The validation aims at assessing the compliance of a system according to the design objectives. Generally, this evaluation is done by submitting the system (or a model that represents it more or less accurately) to a series of tests. The quality of test cases determines the confidence in the validation results. When the focus is put on a model, the validation is then linked to simulation. In the domain of vehicle systems, the simulation must take into account the kinematic/dynamical models of the vehicle and their connection to the ground (tyres on road) and model-board sensors. The representation model of the real world in which sensors are taking their measures and where vehicles evolve must be precise enough from a physics point of view. Various proposals currently exist to simulate robots and vehicles in virtual environments, to model their dynamical behaviors and their sensors. Among the existing simulators, the following can be mentioned:

- *Open source*: MissionLab [3], Player/Stage/Gazebo [4], SimRobot [5], USARSim [6], Simbad [7].

[3] http://www.cc.gatech.edu/ai/robot-lab/research/MissionLab/
[4] http://playerstage.sourceforge.net/
[5] http://www.informatik.uni-bremen.de/simrobot/
[6] http://usarsim.sourceforge.net/
[7] http://simbad.sourceforge.net/

- *Commercial devote to robots*: Webots [8], Microsoft Robotics Studio [9].
- *Commercial devote to vehicle*: Carsim [10], CarMaker [11], PreScan [12], Pro-SiVIC [13].

As expressed in [3], simulation tools can be classified according to the following criteria:

- Physics accuracy: vehicle dynamical model, vehicle kinetic model, road topology...
- Sensors accuracy: frequency, data, noise...
- Functional accuracy: intelligent behavior, ...

According to these criteria, a comparison of simulation tools can be made (Table 1).

Simulator	**Physics Accuracy**	**Sensors Accuracy**	**Functional Accuracy**
MissionLab, Player, SimRobot, USARSim, Simbad	Medium	Low	Medium
Microsoft Robotics, Matlab, Unity, Webots	Medium	Medium	High
Carsim, CarMaker	High	Medium	High
Pro-SiVIC	Medium/High	High	High

Figure 1: Simulators comparison

Most open source simulation tools have a medium accuracy, thus they can be considered as early prototyping tools, which cannot be used for precise simulation. Robot simulation tools are not well adapted to a vehicle because they have more physical constraints. However, they still provide functional results closed to the real car. Car simulation tools are perfect for physics consideration and from the functional point-of-view. Unfortunately, except for Pro-Sivic, most of them don't simulate precise sensors. The major weakness of Pro-Sivic is the physics engine used (ODE), which suffers from drawbacks.

One of the major goals of VIVUS is the validation in a virtual world of automatic control algorithms of vehicles with physics and functional accuracy. Considering this goal, it is required to recreate a vehicle with its precise dynamical physical behavior inside a virtual environment closed to the Reality as possible. Moreover, vehicle sensors should be reproduced in order to give to the control algorithm the same formatted data as given by the real sensors.

[8] http://www.cyberbotics.com/
[9] http://www.microsoft.com/Robotics/
[10] http://www.carsim.com/
[11] http://www.ipg.del
[12] http://www.tno.nl/
[13] http://www.civitec.net

Additionally, VIVUS must run under real time constraints because the control algorithms are the one used with the real vehicles. Physics simulation is handled by the PHYSX software, which allows to apply forces to the different vehicle components (dampers, motors...). All these components are also 3D objects on which physic-based relationship are applied.

3. Simulator Model

This part presents the simulator model. After a global overview of both the simulator architecture and running process, this section will focus on specific components such as the physics model and the sensor models.

3.1. Global Overview

The following subsections discuss the simulator architecture, execution and its implementation.

3.1.1. Simulator Architecture

VIVUS is a 3D-based simulator, which supports physics simulation and realistic 3D rendering. According to game and serious game literature, virtual simulators are mainly composed by the following modules and related data structures: physics simulator, artificial intelligence simulator, and 3D rendering engine. Unfortunately, these modules cannot use the same data structures for efficiency concerns [4]. The VIVUS simulator model follows this approach, and has one module for the physics engine, one for the 3D rendering and one for control algorithms. Figure 2 illustrates the overall architecture. Control

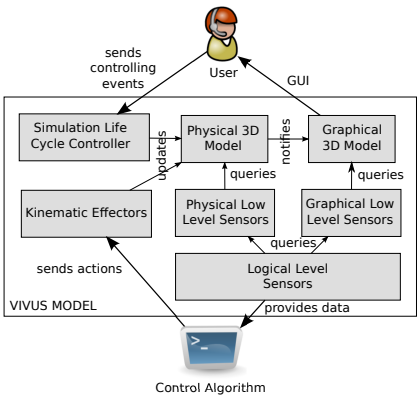

Figure 2: VIVUS global architecture

algorithms are external software, which retrieve sensor information from VIVUS and send back the control orders. These orders are received and applied by the physics engine. Platoon control [1] and obstacle avoidance [2] algorithms have been successfully applied conjointly with the VIVUS platform. During each

simulation step, the physics model notifies the graphical 3D model about each change. This last model then applies the newly received position and orientation on the displayed objects. According to our model-separation assumption, the graphical-model data structure is based on classical 3D scenegraph.

3.1.2. Simulation Execution

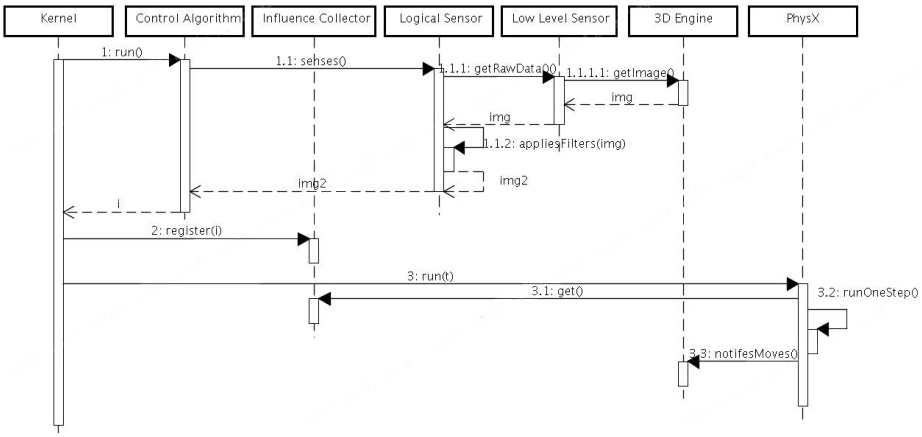

Figure 3: Sequence Diagram for one Simulation Step

Figure 3 illustrates the sequence of actions that are executed during one simulation step. The kernel schedules all the components of VIVUS. It runs the control algorithms, registers the influences given by these algorithms, and then runs the Nvidia PHYSX engine[14]. A control algorithm needs to retrieve information from the simulated objects. In this way, it senses the world model via a set of high-level — logical — sensors. In the given example (Figure 3), only the graphical sensors[15] are represented. Details on the differences between the high-level and the low-level sensors are explained in Section 3.3. With the sensor-output data, a control algorithm can decide an action to do. this action is sent back to the Kernel. This control-algorithm invocation is repeated for each available algorithm in the system. As soon as all the control algorithms have been performed, the Kernel launches the physics simulation, which will retrieve control-algorithm influences, solves them, and applies the resulting reactions [4, 5]. For each moving object inside the physics model, a notification is sent to the 3D rendering engine to update its internal data structures.

[14] `http://www.nvidia.fr/object/nvidia_physx.html`

[15] Graphical sensors are sensors, which only need graphical information, that is retrieved from the 3D world, to build their output

3.2. Physics Model

Physics model is based on the PhysX engine, which can be considered to be one of the best as for accuracy and obtained realistic behavior. The implementation of PhysX is not public and copyrighted. Basically, it is defined by $\langle E, L\rangle$ where E and L are the sets of simulated objects and of physics laws, respectively. Each object supported by the physics-simulation engine is defined by $\langle G, p, o, m, S_l, S_a, A_l, A_a\rangle$; m is the mass of the object; S_l and A_l are the current linear velocity and acceleration of the object; S_a and A_a are the angular velocity and acceleration. p and o are the position and orientation. Finally, G is the geometrical shape associated to the object (basically a box, a cylinder, or a sphere). Equation 1 is the transformation applied by the physics simulator at each simulation step t to obtain the model state at step $t+1$. Let δ_t and f be respectively the current state of the simulated world and the function that is mapping an object to a motion request. Operator $\prod$ permits to detect and to solve conflicts between motion requests from any object ω , according to the physics laws. Operator $\oplus$ computes a new world state from a given one and a set of motion actions.

$$\begin{array}{rl} \delta_{t+1} : \langle E, L\rangle \times (E \to \mathbb{R}^3) \to & \langle E, L\rangle \\ (\delta_t, f) \mapsto & \delta_t \bigoplus \prod f \end{array} \tag{1}$$

According to [4], both operations $\prod$ and $\oplus$ are key features of the environment model in a 3D simulation. In the vivus model, these operators are implemented with the PhysX libraries. In opposition, other models such as JaSim[16] [4] have software procedures for these operators. However, both approaches use similar algorithms to solve the problems of conflict detection, and of model state updating.

The operator $\prod$ is defined by Equation 2. It takes as parameters the function that associates each simulated object to a force to apply on this object. First, the operator $\prod$ computes the new positions and orientations of any object ω after the application of the force m on it. With these informations, geometrical intersection between the meshes of objects of each pair is tested (denoted i in Equation 2). If no collision nor intersection is detected, the forces are applicable on the object without change on their values, and they are replied by the operator $\prod$. If two objects are under collision, the operator computes the penetration points and the physic reaction of the both objects to the collision, according to the Newton laws. The result of the $\prod$ operator is a function that maps each object to its valid position and its valid orientation after the application of the forces (denoted by the operator $\lfloor\rfloor$). In PhysX, the implementation of this algorithm is based on a tree data-structure to speed up the computation of i. Use of trees is classical and standard in computer-graphics and simulation. This use of trees is out of scope of this paper.

[16] http://www.multiagent.fr/Jasim_Platform

$$\begin{array}{rl} \prod : (E \to \mathbb{R}^3) \to & (E \to \langle \mathbb{R}^3, \mathbb{H} \rangle) \\ (f := \omega \mapsto m) \mapsto & \begin{cases} \omega \mapsto (\sigma_p(\omega) + m, \sigma_o(\omega) + m) & \text{if } i = \emptyset \\ \prod (f \cap \{e \mapsto \lfloor f(e) \rfloor | (e, b) \in i\}) & \text{else} \end{cases} \\ i := & \{a \in E | a \neq \omega \wedge \sigma_G(a) + f(a) \cap \sigma_G(\omega) + m\} \end{array} \tag{2}$$

The operator $\oplus$ is defined by Equation 3. It loops on all the objects in the world state δ_t, and it applies the motion vectors and quaternions to them. This operation is linear in time complexity. In PhysX, this operator is directly invoked on all the objects after the collision reactions were computed.

$$\begin{array}{rl} \oplus : \langle E, L \rangle \times (E \to \langle \mathbb{R}^3, \mathbb{H} \rangle) \to & \langle E, L \rangle \\ (\delta_t, f) \mapsto & \langle \{e + f(e) | e \in \sigma_1(\delta_t)\}, \sigma_2(\delta_t) \rangle \end{array} \tag{3}$$

Equations 2 and 3 are coded in Algorithm 1, which illustrates the physics simulation loop. First, the algorithm computes the new positions and orientations $G'(o)$ of all the objects at line 3. Function $applyForce(s, t, f)$ applies the Newton forces to the shape given by $G(o)$. The second step of Algorithm 1 is the collision detection. The penetration points between objects of each pair are computed at line 9. If these points exist, the forces that led to the collision are clipped to avoid it. Finally, after all the collisions have been discarded, Algorithm 1 moves the objects and computes their kinematic attributes at line 18.

3.2.1. Implementation Notes

Modules illustrated by Figure 2 are implemented in C++; except the 3D rendering engine and its associated model, and the graphical low-level sensors, which are written in Java. These two parts of the code are connected through sockets or a Java Native Interface. The control algorithm and the user are external actors of the system. They are connected with a network interface to the simulator. The rendering engine is based on Java3D and its scenegraph. The low-level graphical sensors are Java3D cameras inside this scenegraph. They are executed inside renderer threads at fixed rate to capture the pictures of the scene. These pictures are replied to the logical-level sensors for applying some noises. The physics 3D model is written in PhysX. The kinematic effectors are functions that change the positions of the physics-simulated objects. The physics level-level sensors are based on the query functions of PhysX (ray-cast, box intersection, etc.)

3.3. Sensor Architecture

Real vehicles are using a set of sensors for immediate environment perception: laser rangefinder, Lidar, sonar, GPS, mono or stereo video camera, etc. One of the first steps for designing VIVUS is to identify the different types of sensors to simulate. They are classified according to the type of data they produce and to the data they need from the environment.

Algorithm 1 Physics Simulation Algorithm

procedure SIMULATEPHYSICS($objects, t, forces$)
  repeat ▷ Equivalent to $\prod$ operator
    for $\{o \in objects\}$ **do**
      $G'(o) \leftarrow \text{applyForce}(G(o), t, forces(o))$
    end for
    $hasCollision \leftarrow$ **false**
    for $\{o_1 \in objects\}$ **do**
      for $\{o_2 \in objects | o_2 \neq o_1\}$ **do**
        $p \leftarrow \text{computeCollisionPoint}(G'(o_1), G'(o_2))$
        if $p \neq \emptyset$ **then**
          $forces(o_1) \leftarrow \min(\text{clipForce}(G(o_1), p, forces(o_1)), forces(o_1))$
          $forces(o_2) \leftarrow \min(\text{clipForce}(G(o_2), p, forces(o_1)), forces(o_2))$
          $hasCollision \leftarrow$ **true**
        end if
      end for
    end for
  until $hasCollision =$ **false**
  for $\{o \in objects\}$ **do** ▷ Equivalent to $\oplus$ operator
    $G(o) \leftarrow G'(o)$
    $v \leftarrow \frac{G'(o) - G(o)}{t}$
    $acceleration(o) \leftarrow \frac{v - velocity(o)}{t}$
    $velocity(o) \leftarrow v$
  end for
end procedure

- *Image sensors* produce a bitmap; needs the 3D world.
- *Video sensors* produce a sequence of bitmaps; needs the 3D world.
- *Geometry sensors* produce intersection information on a predefined set of rays, or between objects; needs the physics world.
- *Location sensors* produce the vehicle positions and orientations; needs the physics world.
- *State sensors* produce the states of the vehicles or one of their components (communication, engine, etc.), or the states of the simulated environment components such as weather reports; needs both the 3D and the physics world.

These sensors handle the recovery of data from the universe. However, sensor data is directly used by the control algorithms. It has been decided, for development issues, that control algorithms have to be the same in the virtual vehicle and in the physical vehicle. Then, sensor algorithms must provide the same output data as the real sensors: same data format, frequency and data quality. For avoiding the creation of several almost identical sensors in each category, each virtual sensor is decomposed into two parts: the low-level sensor to collect data from the virtual universe (3D, physics or both), and the high-level sensor to organize data from the lower level so that it meets the exact specifications of the real sensor. According to this structure, it is possible to connect a low-level sensor to many high-level sensors.

3.3.1. Low-Level Sensors

For creating various low level sensors, the universe has been decomposed into three parts: the physics part, the graphical part, and the logical part. This decomposition is the basis of the overall architecture implementation of the simulation tool. According to the type of the required data, the sensor is connected to one of the universe parts. VIVUS does not reproduce the internal physics of the sensors, in opposite with specific commercial offers (Pro-CIVIC, etc.) that already support physics and accurate simulation of such sensors. Sensor algorithms implement simple models, which have the same output properties as the real sensors. This choice of simplicity is due to the real-time constraint, which is required by the global simulator. The physics part of the environment is supported by the PHYSX engine. The graphical environment is supported by the rendering engine. The logical part is supported by dedicated modules of the VIVUS simulator. The following items describe all the low-level sensors defined in VIVUS.

Geometry and Location Sensor. The geometry and location sensor uses the physics and/or graphical parts of the environment to compute the possible collisions between objects and to measure the distance between them. Physics environment depends on the physical attributes of simulated vehicles (shock, acceleration, braking, etc.) on one hand. On the other hand, it also depends on

computations based on environment 3D geometry. Given the laser rangefinder, for instance, the geometric sensor needs that PhysX computes the intersections between the 3D objects and the line segments representing the rays, which are spreading out by the laser rangefinder.

Image Sensor. The image sensor and video sensor use graphic images from the 3D projections, computed from significant points of view, with a specific resolution and frequency. For a stereoscopic camera, two bitmaps from the points of view of the two cameras are generated using the 3D rendering engine. For performance concern, the computed images are not directly displayed but transmitted to the high-level sensors.

State Sensor. State sensor takes global information, like the weather, from the vivus state or from one or more of the low-level sensors. Then, data is formatted and filtered by the state sensor. For example, GPS sensor takes global position and orientation of the vehicle from the PhysX object, and translates them into longitude-latitude pairs.

3.3.2. High-Level Sensors

The main goal of the high-level sensors is to format data, acquired from the low-level sensors, in order to fit the real sensor output format, and to integrate some noises at the same time. For example, let consider virtual cameras, the 3D rendering engine produces very precise bitmaps without visual default. The high-level camera introduces some defaults (radial distortion, chromatic aberration, low-pass filter, etc.) Organization of sensors allows to apply different filters on perfect initial information to match the real sensor's characteristics. Another important aspect is the ability of vivus for simulating specific failures of the sensors, such as partial data sending, GPS accuracy discontinuity (real-time or following a specific scenario), etc. Let take an example of a ray-based sensor: the Laser Range Finder (LFR) LMS200 from SICK. This sensor is based on a collection of 180 horizontal infrared rays fired from the front of the vehicle. Each time a ray is cut by an obstacle; the LRF sensor replies the distance to the intersection point. The LRF sensor implementation is based on the ray-object intersection algorithm from PhysX. Unfortunately, the PhysX's intersection algorithm provides precise values computed from the virtual world. A real-life LRF sensor is mostly influenced by environmental noises such as the position of the Sun, material on which the infrared rays are reflected, etc. To reproduce as accurate as possible the behavior of the real LRF sensors; the results from the intersection algorithm should be noised. Let take a second example: the General Position System (GPS) sensor. It returns the global position of the vehicle on Earth. The low-level GPS sensor is simple: from the Euclidian coordinates given by the PhysX engine, it computes the equivalent longitude-latitude position.

As for the LRF sensor, the GPS position is perfect. Real GPS receivers have an error from 4 to 1 meters [6, 7], according to several environmental constraints around the vehicle, such as the ionosphere or buildings. GPS perturbation is introduced with equations on the GPS error estimation, that could be found in literature.

4. Comparison Methodology and Application Presentation

4.1. Comparison Methodology

As expressed in the introduction the main goal of our project is to develop a reliable simulation tool able to test new intelligent vehicle algorithm and to perform impossible scenario (crash, extreme tyre/road contact conditions,...). In order to validate the choices made in this simulator, we compare the results obtained in simulation with those obtained with real car. To that way, we take as an example an application well-known in our Laboratory and for which results have already been published in [1], [8], and [**?**].

Since the 3D models are geo-referenced, it is possible to record trajectories with centimeter precision. These trajectories may be replayed. Moreover, the interface between the control algorithms and the simulator has been developed to fit the same specifications as for the interface between these control algorithms and the physical vehicle.

The comparison protocol is then the following: (i) perform the tests with real vehicles at a specific place recording each vehicle trajectory; (ii) perform the same test in simulation at the same virtual place (geo-localized reference) and record each virtual vehicle trajectory; (iii) compare results obtained.

Before exposing the results obtained in Section 5, next paragraph presents an overview of the chosen application.

4.2. Application Presentation

Platoon systems can be defined as sets of vehicles that navigate according to a trajectory, while maintaining a predefined configuration [9]. Many applications, such as transportation, land farming, search-and-rescue and military surveillance, can benefit from platoon system capabilities. The platoon control model used in this paper is based on a local approach (i.e. each vehicle is only able to perceive its preceding). Thus, each vehicle determines its trajectory relatively to a preceding vehicle. The behavior of each vehicle bases exclusively on local perceptions. Other approaches [10] base on centralized services such as geo-localization or communication with an infra-structure. Local and decentralized approaches are simpler, because a locally determined behaviors abstract from global details. Local approaches also possess a greater reliability, as operation does not depend on central components with specific roles. They are less expensive, as they do not require any road infra-structure. In this model, controls are enabled by forces, which are computed from a virtual physical link. The virtual link is made by a classical spring-damper model as shown in Figure 4. Let consider the point of view of the platoon vehicle V_i. In this approach, operation of V_i bases on its perception of the preceding vehicle, V_{i-1} in the train. The local model is based on forces. They are not materially exerted but computed from a virtual physical interaction device relating the vehicles V_i and V_{i-1}. This virtual link connection is made by a bending spring damper (Figure 4). The spring force attracts the vehicle V_i to the vehicle V_{i-1}. The damping force, related to the spring force, avoids oscillations. The torsion force bends the spring damper system (Figure 4). Acceleration value

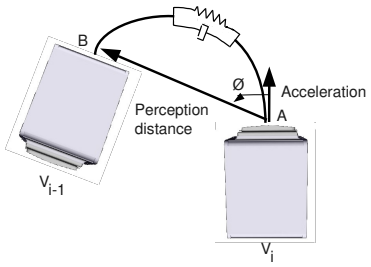

Figure 4: Virtual link between vehicle

can be computed using Newton's second law. By discrete integration, speed and vehicle state (position and orientation) and then the command law can be determined. In this case, command law consists in vehicle direction and speed. The choice of a command law takes into account the characteristics of the test vehicle used in our Laboratory.

The use of Physics inspired forces in the local platoon model enables an easier tuning of the interaction model parameters and the adaptation to any kind of vehicle. Besides, the physics model has been used to prove platoon stability, by using classic physical proof method: energy analysis. By the same way, another stability proof has been realized following a transfer-function approach [1]. A verification has been realized to validate a condition for safe operation and passengers integrity, i.e., the impossibility of inter-vehicular collision during train operation. This verification is performed using a compositional verification method [8] dedicated to this application. This verification exhibits the validity of safety properties during platoon evolution.

5. Experiments

On a linear platoon application, this section presents a comparison between a vehicle simulation and a real experimentation.

5.1. Simulation and Experimental Protocol

Based on the algorithm described in the previous section, simulation and experimentation scenarios are designed and performed to check platoon evolution during the lateral displacement situations and a set of safety conditions.

Simulations are realized with VIVUS simulator (Figure 5 (left)) presented in this paper. The second platform is composed of GEM electrical vehicles modified by the "Systèmes et Transports" Laboratory (Figure 5 (right)). These vehicles have been automated and can be controlled by an onboard system.

Experiments were conducted on the Belfort's Techno-park site. The simulations were performed on a 3D geo-localized model of the same site built from Geographical Information Sources and topological data.

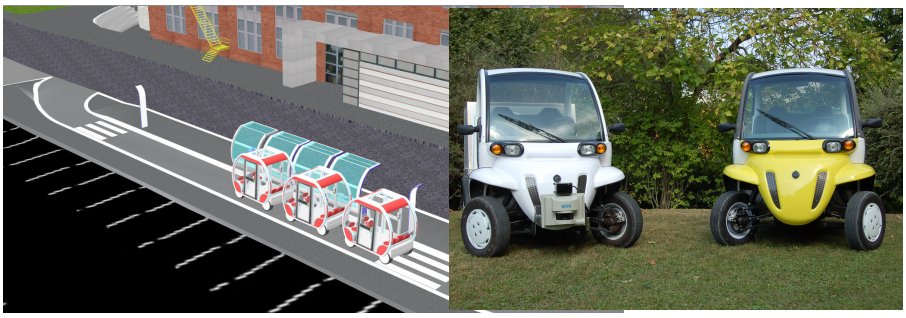

Figure 5: Simulation of a platoon vehicle Station (left) and SeT Laboratory electrical vehicle (right)

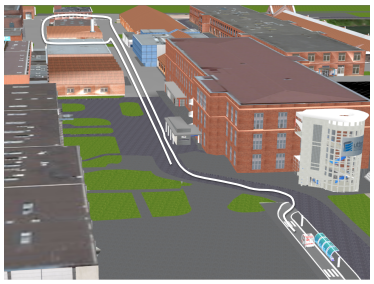

Figure 6: Simulation et experimentation path

Figure 6 shows the path (white curve) used for the simulation and experimentation. This path allows to move the train on a long distance in an urban environment using a trajectory with different curve radius. To compare the simulation and experimentation results, parameters were the same on simulation and real experiments. Thus, the perception of each vehicle is made by a simulated laser range finder having the same characteristics (range, angle, error rate, etc.) as the vehicle real sensor. The distance and angle from one vehicle to the preceding are computed thanks to this sensor. The algorithm used is the same for both simulations and real experiments. Moreover, the program runs on the same computer. Indeed, a great attention has been paid on the fact that simulated vehicles should have the same communication interface as the real ones. Thus, passing from the simulation to the real vehicle relies only on unplugging the artificial intelligence computer from the simulator and plugging it on the real vehicle. However, experiments were performed with a more important regular distance in order to avoid collision that can lead to irreversible damage for vehicles. The distance has been established to 4 meters and the safety distance to 1.5 meters.

5.2. Vehicle Physics Model

In order to obtain simulation results as near as possible from the reality, a complete physics model of vehicles has been made. This section presents the dynamical model of the SeT Laboratory vehicle platform.

This model has been designed to suit the PhysX engine requirements. Models designed for it are based on composition of PhysX elementary objects. New components can also be defined using these PhysX requirements. Vehicle dynamical modeling is a common task in simulation. The SeT-car vehicle is then considered as a rectangular chassis with four engine/wheel components. This choice can be considered to be realistic, the chassis being made as a rectangular un-deformable shape. The following parts describe all the parameters determined and/or computed for the vehicle.

5.2.1. Chassis Model

Chassis is modeled as a rectangular shape with a size denoted C, a mass denoted M_c and a gravity center G_c. Considering vehicle design, the following propositions can be exposed:

$$C = \begin{pmatrix} L_c \\ l_c \\ h_c \end{pmatrix} \quad G_c = \begin{pmatrix} 0 \\ 0 \\ h_{zs} \end{pmatrix} \tag{4}$$

Gravity center position G_c of the chassis has been computed taking into account gravity center of the body, gravity center of each component of the chassis, i.e., battery, embedded electronic card and components, etc. Wheel/engine components are not included in this computation.

5.2.2. Tyre and Shock Absorber Models

Wheels are modeled as dynamical objects. Each wheel is considered to have diameter R and a mass M_{wheel}, and a specific position. Since the last version of PhysX, wheels can be modeled with a specific collider class: wheelcollider[17]. With this specific model, tyre grip computation function takes tyre sliding as an input. Lateral and longitudinal tyre sliding are computed separately. The output of this function is the tyre grip. This value can then be interpreted depending on the tyre model used. PhysX tyre model computes tyre friction constraints from a Hermit spline. Model parameters have been defined from standard data for 130/70-10 Michelin tyre. Then extremum A, asymptote B and rigidity coefficient, P_x for longitudinal and P_y for lateral, have been defined thanks to several experiments made on the real vehicle. Shock absorber model is defined by PhysX with several parameters such as damping constant A_a, stiffness A_r and free length A_l. All these constants were defined from real vehicle experiments.

5.2.3. Engine Model

Engine model proposed by PhysX corresponds to a standard engine with a starting torque C_d and a braking torque C_f. Real vehicle engines are electrical engines with permanent magnet allowing it to make both acceleration and braking. After experiments with a standalone engine, C_d and C_f values have been determined.

5.2.4. Summary of the Vehicle Physics Model

- $L_c = 1.95\ m$ chassis length
- $l_c = 1.195\ m$ chassis width
- $h_c = 2.3\ m$ chassis height
- $M_c = 350\ kg$ suspending mass
- $h_{zs} = 52\ cm$ gravity center height
- $R = 42\ cm$ wheel diameter
- $M_{wheel} = 7\ kg$ wheel mass
- $L = 1.2\ m$ semi-length
- $e = 1.1\ m$ semi-width
- $h_{center\ wheel} = 19\ cm$ height of wheel axis
- $A_r = 1000$ stiffness constant
- $A_a = 330$ damping constant
- $A_l = 12\ cm$ shock absorber free length
- $C_d = 500\ MKG$ starting torque
- $C_f = 500\ MKG$ braking torque
- $A = (1.0; 0.02)$ Extremum coordinates
- $B = (2.0; 0.01)$ Asymptote coordinates
- $P_x = 15$ tyre longitudinal sliding rigidity
- $P_y = 15$ tyre lateral sliding rigidity

5.3. Comparison between Experimentation and Simulation

This subsection presents tests performed both in simulation and with real vehicles to assess the quality of platooning. The following cases were discussed:

[17] http://unity3d.com/support/documentation/Components/class-WheelCollider.html

- Evaluation of inter-vehicle distance: measuring the distance between two following vehicles, compared to the desired regular inter-vehicle distance during platoon evolution (Figure 7).
- Evaluation of lateral deviation: measuring the distance between the trajectories of the geometric center of a vehicle relative to the same path of his predecessor (Figure 7). For the measurement, points on the first vehicle trajectory were selected. Then, the normal trace of these points is drawn and a measure of the distance between the selected point and the point of intersection with the trajectory of its predecessor is made.

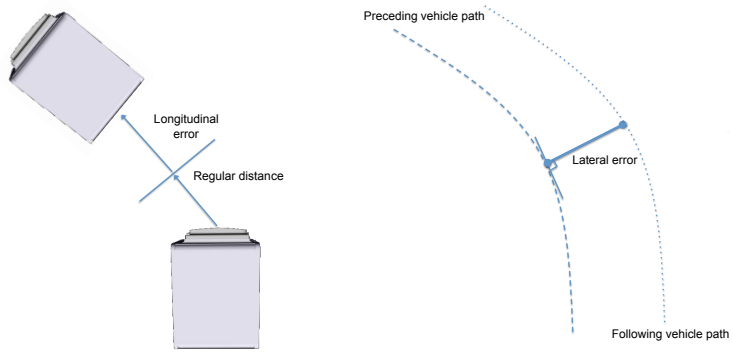

Figure 7: Longitudinal error (left) Lateral error (right)

5.3.1. Evaluation of Inter-Vehicle Distance

To evaluate the inter-vehicle distance, the train is submitted to critical operations such as starting, emergency braking and quick change of speed of the first vehicle. Figure 8 (top) shows the distance variations between vehicles in relation to quick changes of first vehicle speed. Figure 8 (bottom) shows the distance evolution between two electric vehicles during the experiment. Figure 9 gives an overview of the inter-vehicle distance variation. One can observe that despite the very sudden changes in first vehicle speed, this value is above the safety distance and stabilizes rapidly to the regular distance. Figure 8 (top and bottom) present the same inter-vehicle variation.

5.3.2. Evaluation of Lateral Deviation

The distance between the trajectories of the vehicle geometric center relatively to the same path of its predecessor has also been evaluated. Lateral deviation may cause problems in curves such as collision with vehicles in the opposite direction. Simulations were performed with a train of four vehicles. Figure 10 presents the tracks of vehicles to measure the lateral error between each vehicle. To see the maximum error when the train evolves, lateral errors in relation to the wheel rotation are plotted (Figure 11 (left)). Figure 11 (left)

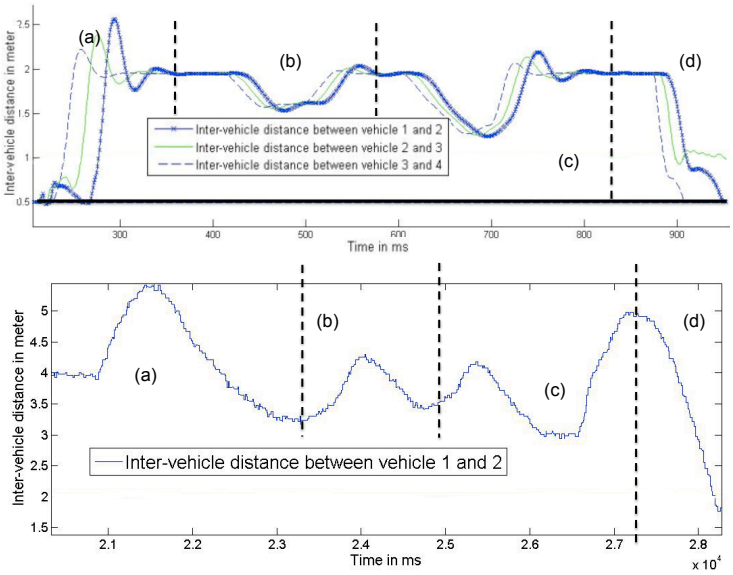

Figure 8: Inter-vehicle distance in simulation (top) and in real experiments (bottom)

Critical case	Inter-vehicle distance
Simulation	
Starting from 0 to maximal speed	Overrun of 30% compared to the regular distance
Speed variation of 30 et 70 %	Inter-vehicle distance variation 20 et 50 %
Safety stop from maximal speed to 0	Above the value of the safety distance
Experimentation	
Starting from 0 to maximal speed	Overrun of 55% compared to the regular distance
Speed variation of 30 et 70 %	Inter-vehicle distance variation 30 et 50 %
Safety stop from maximal speed to 0	Above the value of the safety distance

Figure 9: Evaluation of inter-vehicle distance

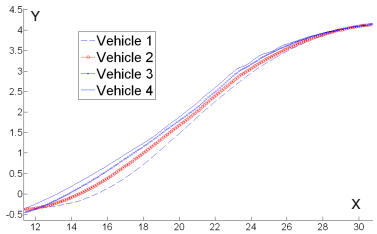

Figure 10: Simulation: lateral error during exit station

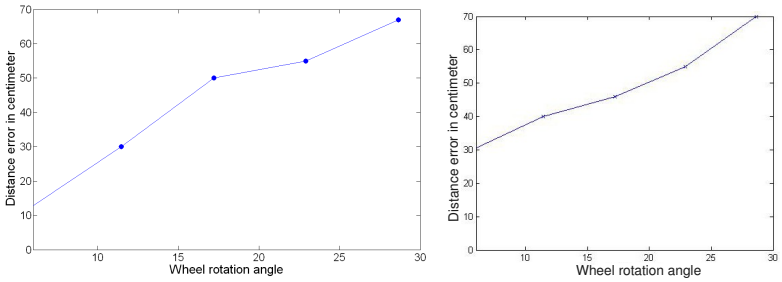

Figure 11: Lateral error in relation to the wheel rotation in Simulation (left) and with real vehicle (right)

shows the maximum lateral error in relation to the radius of curvature of the lead vehicle. In normal operation case (wheel rotation below 10 degrees), it denotes the lateral error is less than 20 centimeters. Experiments were realized with two electric vehicles from the SeT Laboratory. The measure has been made thanks to the GPS RTK installed on the vehicles to know their positions in centimeters. To see the maximum error when the train evolves, lateral errors in relation to the wheel rotation are plotted (Figure 11 (right)). Figure 12 shows the maximum and average lateral error in relation to the radius of curvature from the lead vehicle. Results presented in the table above illustrate that the

Wheel rotation (degree)	Curve	Medium error in simulation	Medium error in experimentation
5.73	18 m	12 cm	30 cm
11.46	9 m	30 cm	40 cm
17.2	6 m	50 cm	46 cm
22.9	4.5 m	55 cm	55 cm
28.65	3.6 m	67 cm	70 cm

Figure 12: Trajectory error in curve

tracking error of the vehicle simulation is close to the experiment. Indeed, results representing the average error between each car of the train have same order values.

6. Conclusion and Perspectives

This paper presents an urban vehicle simulator based on two main engines: one for physics simulation, and one for 3D immersion in a topological environment. The use of these engines allows a precise simulation of vehicle dynamics and a wide range of physical sensors. This sensor's simulation is split into two parts: the low-level sensors that extract information from physics and 3D models; and high-level sensors, which are formatting data coming from the low-level sensors to fit the physical sensor specifications. These design choices enable the

test of control algorithms, as if they were run on a real vehicle. Indeed, these algorithms obtain the same sensor inputs with the simulator and the physical vehicle. Thus, going from simulation to the real car relies only on an unplugging and replugging operations. This simulator has been successfully used as a prototyping tool for sensor designing and positioning. It is also a test bed for the developers of artificial intelligence algorithms. Experiments exposed in this paper shows simulation results closed to the Reality. The differences with the real conditions are essentially due to the precision of the world topology, and the model choose for the wheel/road contact. From this day forwards, we aimed at improving this simulator on the 3D level by changing the 3D engine. Indeed, the actual engine based on Java3d doesn't allow to easily deal with climatic simulation, shadows, lens flare, etc. The change of this engine will provide better simulation for image based sensors. We work on the vehicle physics model to refine several elements, and introduce new relevant components, which influence global vehicle behavior. Moreover, we are also designing a new physics model for partner vehicles in the context of Safe-platoon project[18].

References

[1] J.-M. Contet, F. Gechter, P. Gruer, A. Koukam, Bending virtual spring-damper : a solution to improve local platoon control, Lecture Notes in Computer Science 5544.

[2] F. Gechter, J.-M. Contet, P. Gruer, A. Koukam, Car-driving assistance using organization measurement of reactive multi-agent system, in: International Conference on Computational Science 2010 (ICCS 2010), Amsterdam, 2010.

[3] J. Craighead, R. Murphy, J. Burke, B. Goldiez, A survey of commercial and open source unmanned vehicle simulators, IEEE International Conference on Robotics and Automation (2007) 852–857.

[4] S. Galland, N. Gaud, J. Demange, A. Koukam, Environment Model for Multiagent-Based Simulation of 3D Urban Systems, in: the 7th European Workshop on Multi-Agent Systems (EUMAS09), Ayia Napa, Cyprus, 2009.

[5] F. Michel, The IRM4S model: the influence/reaction principle for multiagent based simulation, in: Sixth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS07), ACM, 2007. doi:http://doi.acm.org/10.1145/1329125.1329289.

[6] D. A. Grejner-Brzezinska, R. Da, C. Toth, Gps error modeling and otf ambiguity resolution for high-accuracy gps/ins integrated system, Journal

[18] Safe-platoon is a project labeled by the National French Research Agency (ANR) `http://safeplatoon.utbm.fr`

of Geodesy 72 (1998) 626–638, 10.1007/s001900050202.
URL http://dx.doi.org/10.1007/s001900050202

[7] B. W. Parkinson, GPS Error Analysis, Global Positioning System: Theory and applications 1 (A96-20837 04-17) (1996) 469–483.

[8] J.-M. Contet, F. Gechter, P. Gruer, A. Koukam, An approach to compositional verification of reactive multiagent systems, Working Notes of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI), Workshop on Model Checking and Artificial Intelligence, Atlanta, Georgia, USA, July 11-12, 2010.

[9] J. Hedrick, M. Tomizuka, P. Varaiya, Control issues in automated highway systems, IEEE Control Systems Magazine 14 (6) (1994) 21 – 32.

[10] P. Martinet, B. Thuilot, J. Bom, Autonomous navigation and platooning using a sensory memory, International IEEE Conference on Intelligent Robots and Systems, IROS'06, Beijing, China, 0ctober 2006.