Cybernetic Transportation Systems Design and Development: Simulation Software cybercars

Sébastien Boissé, Rodrigo Benenson, Laurent Bouraoui, Michel Null Parent, Ljubo Vlacic

Cybernetic Transportation Systems Design and Development: Simulation Software

Sébastien Boissé, Rodrigo Benenson, Laurent Bouraoui, Michel Parent, Ljubo Vlacic

***Abstract*— The growing number of vehicles saturates cities in terms of congestion and pollution. A Cybernetic Transportation System (CTS) appears to be a way to resolve those problems. Based on a network of clean driverless vehicles (the cybercars), CTS aims to improve safety and organization of urban transport by providing a door-to-door complement to efficient and fast mass transport. The simulation software presented in this paper has a goal to facilitate the development of such a transportation system. It simulates several cybercars in a dynamic virtual 3D environment, and provides sensors information in real vehicles. As it is easy to create scenarios containing various static and moving obstacles, evaluation of control algorithms in several situations is its main feature. The adopted architecture for the simulation tool also enables evaluation of road traffic scenarios constructed on various levels of interaction or cooperation among cybercars.**

***Index Terms*—Cybernetic Transportation System, Driverless Vehicles, Cybercars, Simulation.**

I. Introduction

The common use of personal vehicles all over the world causes safety, congestion and pollution problems, especially in large cities. To solve these problems, an innovative approach, called Cybernetic Transportation System (CTS), has been proposed [10,11]. CTS is a city transportation system based on a fleet of unmanned electrically powered car-like robots. These cybercars, equipped with sensors, have the ability of autonomously transporting people in an urban environment. The highly dynamic nature and unpredictability of such environments makes the development of a cybernetic transport system very complex. Therefore, the use of a simulator appears to be a good solution to facilitate developments of CTS driving capabilities as it allows multiple testing of the same road traffic scenarios and easy evaluation of the overall system's traffic performance.

Lots of simulation software packages have been developed in the past and more recently, some 3D simulators were released, again reducing the gap between simulation and reality. Among all of those simulators, some of them deal with wheeled robots, including various sensors like cameras, range sensors or contact sensors. Most of these tools are built upon Open Dynamic Engine (ODE) [4], an open source library providing high performance for rigid body dynamics simulation.

In this area, two free tools, OpenSim [1] and Gazebo [6] are able to simulate several wheeled vehicles with sensors in a 3 dimensional environment. The OpenSim provides real-time rendering for simple shaped robots while Gazebo can mimic a complex world thanks to simulated vehicle sensors. The Webots [7], a commercial simulation software for mobile robots proposed by Cyberbotics, simulates several models of vehicles and various sensors. This prototyping application permits testing of control algorithms on various simulated robots.

In another area, USARSim [8] has been developed to be the basis for the RoboCup Urban Search And Rescue (USAR) simulation competition. This simulator, based on the Unreal Tournament Game Engine, has the particularity to simulate a camera with accuracy thanks to its high-quality rendering.

Finally, CycabTk [2] has been developed at INRIALPES (France) to simulate a cybercar with its sensors (camera, SICK laser, …), using the same interface as real vehicles. This simulator runs under linux and uses mgEngine as 3D engine [5].

While some simulation tools tend to be as generic as possible in order to simulate several robots and sensors, we have chosen to develop a simulation tool that can accurately mirror the vehicle we are using at INRIA (Lara Research unit, France) and Griffith University (Intelligent Control System Laboratory, Brisbane, Australia), in a 3D virtual environment. In addition, to appropriately simulate CyCabs (name of the cybercars developed by Robosoft) dynamics and sensors, this tool is designed to face CTS issues by providing an opportunity to test the behavior of a fleet of vehicles in the same simulated environment. The latter is composed of a world model (map containing roads, buildings and other static obstacles), moving obstacles and cybercars. The 3D models of both maps and moving obstacles are fully customizable as

This work was supported by European Commission Information Society Technologies under Grant 028262 and Griffith University.

Sébastien Boissé is with IMARA Team, INRIA Rocquencourt, France and with Ecole Centrale Nantes, Nantes, France (e-mail: Sebastien.Boisse@eleves.ec-nantes.fr).

Rodrigo Benenson, Laurent Bouraoui and Michel Parent are with IMARA Team, INRIA Rocquencourt, France (e-mail: Rodrigo.Benenson@inria.fr, Laurent.Bouraoui@inria.fr, Michel.Parent@inria.fr).

Ljubo Vlacic is with ICSL Laboratory, Griffith University, Brisbane, Australia (e-mail: L.Vlacic@griffith.edu.au)

they are created with 3D modelers. In this way, we will be able to deal with interactions between vehicles involved in many different traffic situations.

In section II, we present the architecture of the framework, and then give details about the simulation in sections III and IV. In section V we present the different traffic situations considered.

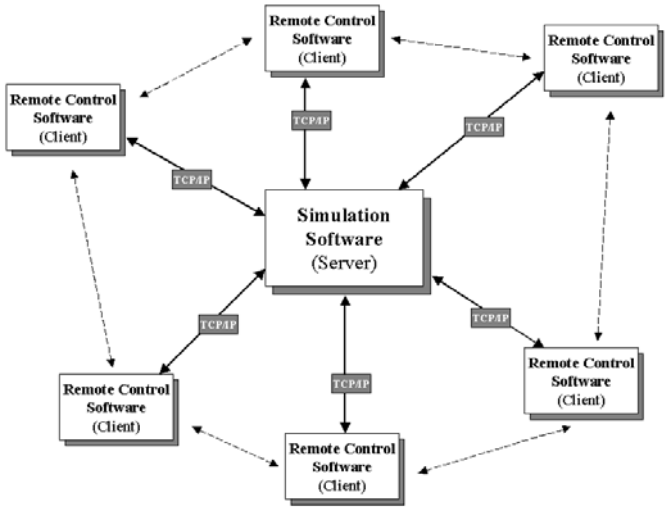


Figure 1: Framework architecture

The first results obtained with this simulator are highlighted in section VI and finally, limitations and future works are discussed in section VII and section VIII.

II. Framework architecture

Due to the high dynamicity of urban environments, the cybercar control software program needs to obtain data from sensors and to analyze them at a high frequency, which is computationally expensive. This often requires the whole CPU of a powerful computer to work properly. Moreover, each cybercar is controlled by its own computer; therefore, we adopted the same architecture when designing the simulation tool. The simulator acts as a server and each control software program (clients), which can be run on distinct computer, is connected to a simulated vehicle inside the simulator through a TCP/IP socket (Figure 1). Basically, the simulator sends the simulated sensors' data to each control software program and receives in return commands from them to make the vehicle move inside the virtual environment (Figure 2).

The design of this framework allows the different remote control software programs to communicate between each other through the network without transmitting by the simulator, using either Ethernet or Wifi links. In this way, the control software program, used to exchange data between cybercars, can also be tested with this simulator.

Remote control software programs can be easily connected to the simulator thanks to Zero Configuration Networking (ZeroConf) standards [9], permitting automatic detection of the server's address (IP + PORT) over the network (Figure 3).

The simulator, running under windows, uses a cross-platform interface allowing control software program developed under windows or linux to connect to it. The interfaces to access sensors' information and to send commands are the same as in real vehicles, so the same application can be run on both real and simulated CyCabs.

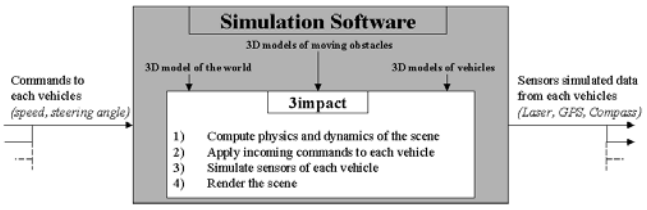


Figure 2: Simulation

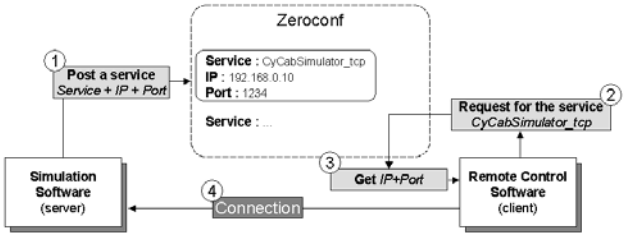


Figure 3: Zero Configuration Networking

III. Dynamic environment simulation

A. Simulation loop

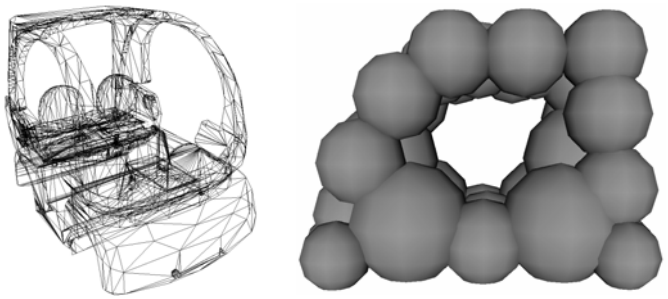
To simulate the dynamics of the environment, we use a physics engine able to compute interactions between all objects introduced into the virtual world. Open Dynamic Engine (ODE), a free library for simulating rigid body dynamics allows us to introduce vehicles easily in a 3D environment.

Each simulator requires an interface to show the results of its computations. Rendering of 3D objects appears to be the best solution since we wish to see the results of the simulations in real-time, and in a format than can be easily interpreted by people with non-scientific backgrounds. Further, it makes it easy to save the results in a video file to display our results during exhibitions.

However, rendering is just a way to see the results and must not affect the simulation itself. To render 3D textured objects is not an easy task; it sometimes requires significant CPU time and some care must be taken to minimize the impact on other tasks.

To fulfill these requirements, the simulator has been built using the 3impact Engine [3], which is a low cost library

integrating an ODE-based physics engine and a rendering engine. We made this choice because the rendering engine operates without affecting the physics engine. The frequency of the physics engine can be set (75Hz is used as it provides sufficient accuracy for computations), and will remain constant during the simulation. The simulation main loop, computing all the motions and checking for collisions, will always be executed at the same frequency and the rendering will be done only if time is available so the frame rate will decrease if the computer is not fast enough.



(a) Rendering model (b) Collision model

Figure 4: CyCab models

B. Models

The 3impact Engine does not require any specific 3D model editor; any modeler able to export to DirectX format can be used (for example 3D Studio Max [15], LightWave 3D [16], MilkShape 3D [17], AC3D [18]). Therefore there are no limitations on the physical structure of the models loaded into the simulator.

1) Vehicle models

The simulated vehicle (CyCab) is defined by three 3D models:

a) The first model, made only with triangles, defines the shape of the vehicle. It also contains texture-mapping information. Created from the CAD model, it is only used for rendering so the number of triangles has been reduced (around 7000) to avoid losing too much time when rendering it (Figure 4a).

b) The second model, made only with sphere primitives is the collision model. All collision tests between the vehicle and something else will be performed using this sphere-based model. Therefore, this model must match closely the first model. This model is built up with 41 spheres (Figure 4b).

c) The third model, a simple box bounding the car, defines the mass distribution of the vehicle.

The combination of the last two models defines the model used by the physics engine.

Once the models of the vehicle have been introduced into the virtual world, four independent wheels are joined to this vehicle with strength and damping suspension parameters. Friction parameters between the wheels and the ground are also defined, allowing us to simulate realistic performance of the vehicles [3].

2) Moving obstacle models

For moving obstacles, we use the 3 models described above to define a vehicle. Collision checks are performed using the sphere-based model, so the accuracy of the simulation is dependant on how this model is close to the mesh model (number and size of spheres).

3) World model

World models are 3D maps representing a real area and are only composed of static obstacles (roads, buildings, parked cars…). These triangle-based models, containing information about textures, are used to compute collision checking. However, in order to avoid time-expensive collision tests, world models should only contain basic shapes made with few triangles. Because collision checks between rays and triangles are slower than collision checks between rays and spheres, more detailed static models should be introduced as moving obstacles with null speed in order to use a sphere-based model for collision tests.

4) The whole virtual environment

The various objects (vehicles, world model, moving obstacles) are introduced separately into the simulator. To process the simulation of the global environment, interactions between those objects have to be defined. These interactions can be defined with collision couples specifying friction parameters between two different objects. Creating collision couples between all objects is not necessary and would necessitate a lot of time; so only couples between vehicles and the map are added (only CyCabs need accurate simulation). There are five couples per vehicle (one for the vehicle body, and one for each wheel).

C. Rendering

A camera is introduced into the virtual environment to get a view of the simulation; the position of the camera can be modified in runtime with the keyboard. We can focus on each of the different simulated vehicles from various directions and also are able to get a global view of the scene.

The DirectX-based rendering engine renders the different object seen by the camera in the 3D environment.

IV. Simulation of Sensors and Actuators

A. Sensors

To get a good simulation, models of the sensors we are using have to be as accurate as possible. Moreover, we need to use the same interface to access the virtual sensors' data and the real sensors; we can then use the same application to drive the real vehicle and the simulated vehicle.

The time available to analyze sensors' data and compute commands is dependent on the frequency with which sensors send data, because for each step in the calculation, we want to take new data into consideration. Therefore, it is essential that the simulator provides data at the same frequency as in reality.

As a starting point, three sensors have been implemented: Laser, GPS and compass.

1) Laser sensor

The laser we are simulating (IBEO) scans the world in 181 directions, each separated by one degree. The laser sensor returns a vector of distances to obstacles, each element corresponding to a given direction and we check whether there is any intersection between the rays and obstacles. The laser's range has been set to mirror the real laser.

The different types of obstacles are treated separately to improve calculation speed. In the first step, moving obstacles and simulated vehicles are considered as single bounding spheres; if a collision is detected between a ray and one of these spheres, the accurate sphere collision model of the corresponding object is used to determine the accurate distance to the obstacle. For each direction, if more than one obstacle is detected, the shortest distance is returned.

By default, the scanning frequency of the laser is 8 Hz but this value can be adjusted.

2) GPS sensor

To simulate the GPS sensors, we approximate the small area of the globe where we are located by a map with a 2D system of coordinates. By default, the reference point of this system is the geometric centre of the virtual world model, but we can translate and rotate this coordinate system to match a virtual map with a real environment.

GPS simulated sensors return (X, Y) coordinates relative to this coordinate system. The default frequency of this sensor is 1 Hz but is also adjustable.

3) Compass sensor

Compass sensors give the orientation of the vehicle, relative to north as defined in the 3D model of the world by the simulator.

4) Noise

To make our modeling more realistic, some noise (with customizable Gaussian distribution) can be added to the sensors' data sent to the client.

B. Actuators

On the real CyCab, each wheel is controlled independently by an electric motor. It is the role of the control software program to determine feasible commands before sending them to the actuators as the CyCab has limits on acceleration and wheel angle. In the simulator, the command (speed in m/s and steering angle in radians) will also be applied to each wheel while keeping in mind the physical constraints of the real CyCab. If a non-feasible command is sent, the highest possible command will be applied.

Each command sent by the client to the simulator is temporarily stored and applied to the intended CyCab until new commands are received by the simulator.

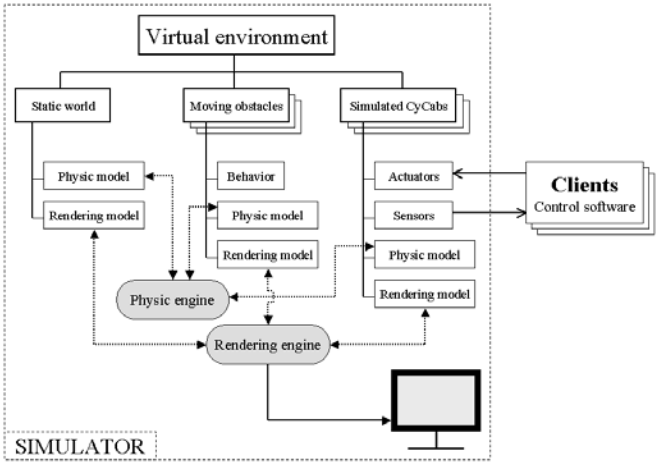


Figure 5: Virtual environment simulation

V. Simulation of Traffic Scenarios

One of the most important features of this software is that it can easily model various traffic scenarios. For instance, in addition to the static environment, it is possible to add moving obstacles like a pedestrian crossing the road or walking on the side of the road.

The trajectory of moving obstacles can be customized with a list of waypoint to reach the map coordinates system and the speed. These obstacles are moved in runtime by the simulator and do not require any external software to pilot them. A simple control system inside the simulator dictates the orientation of the moving obstacle towards its next waypoint. The simulator applies the specified speed and directs the object to its next valid waypoint.

A small library of maps has been modeled, ranging from very simple cases to more complex maps containing buildings

and other static obstacles. They typically represent various traffic conditions that may be encountered in reality like different types of roads (straight or curved) or intersections ("T", "x"). With all of these models, many scenarios can be set by customizing:

1) The number of simulated vehicles
2) Their initial location
3) The number of moving obstacles
4) The type of moving obstacles
5) The predefined motion of these obstacles

Each scenario is defined in a formatted file where all the parameters described above are stored. Figure 5 schematizes the simulation of the virtual environment and Figure 6 shows the view of a scenario involving several simulated vehicles and pedestrians in a city-like crossing situation.

During the simulation, virtual vehicles are driven by remote applications and execute maneuvers defined by control algorithms, taking into consideration the different obstacles detected by the sensors in the virtual world. The camera introduced inside the simulator allows us to analyze the cybercars behavior.

Figure 6: scenario example

The real and simulated CyCabs are not completely isolated from one another and can interact with each other. The level of this interaction depends on the capabilities of the control software program to exchange data. This interaction is possible because the simulation runs in real-time (provides sensors information at the same frequency as real CyCabs), processes data in the same way as the real CyCabs and the world model can be created to mirror the area around the real CyCabs, including GPS location and orientation of the CyCabs. In this way, simulations involving virtual and real vehicles can be run before attempting the same scenario with only real vehicle.

VI. Results

The simulator is relatively new but has already proved its capability to test new control algorithms by simulating CyCabs' driving performance.

The simulator was originally developed to simulate the interaction of cybercars at crossings. The goal was to evaluate whether an exchange of data between two vehicles at a crossing would improve their ability to avoid each other by anticipating the behavior of the other.

Based on a control software program [12] able to drive the car to its final destination while avoiding static and moving obstacles [13], the new control software program now contains communication capabilities to broadcast data through a UDP socket over a wireless network. This communication module allows sending information about various obstacles that have been perceived by the sensors and planned trajectory of the vehicle. This module also receives the broadcasted UDP packets and adds them to the world model previously created with its own perceived obstacles (Figure 7).

The vehicle's world model now contains the obstacles perceived by the sensors, the obstacles perceived by the other cybercars and their last computed trajectories represented as a succession of obstacles in time. Therefore, the planning module will consider all of these obstacles to compute its new trajectory.

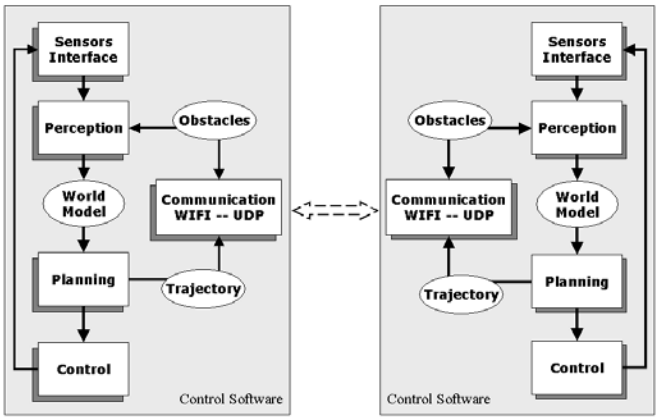


Figure 7: Remote control software program architecture

The simulator was first used to make the existing algorithm from the real CyCab more robust and stable. The first tests of the new algorithm were run in a scenario called "face to face encounter" that involves two vehicles facing each other set on a collision course.

The results of these tests show that the vehicles anticipate and avoid the collision by adapting their trajectory to the new received future obstacles (i.e. the trajectory of the other cybercar). Without the communication, tests had shown that the vehicles can stop in front of each other, unable to move anymore. Then, without any priority rules in the planning algorithm, vehicles were able to pass beside each other.

Now that the algorithm has been validated, two real CyCabs are being equipped with the appropriate sensors to run the same experiments for real.

VII. LIMITATIONS

The purpose of this simulation software is to test functionalities of autonomous driving algorithms in city-like road traffic situations, helping design and development of Cybernetic Transportation System. However, the presented simulation tool cannot be used for the purpose of evaluating the global performance of CTS systems. Another tool will be needed to accurately monitor the traffic (average time to reach a specified location, traffic flow in crossings, etc.) and prove that CTS can resolve congestion problems in cities.

To remove unessential calculations, only collision couples between vehicles and the static world have been defined. This allows more simulated vehicles and more moving obstacles to be added. Therefore, it is not possible to get accurate feedback about collisions between other objects, and the results of the simulation (obstacle avoidance for instance) have to be visually interpreted through the rendering window.

As previously mentioned, the main loop frequency remains constant during the simulation ensuring the accuracy of the simulation. Therefore, the number of CyCabs able to be introduced to any given scenario is heavily dependant on the processing power of the computer running the simulator. The quick collision test capabilities of 3impact engine (thanks to sphere-based models) let us introduce up to six simulated vehicles in a world up to 500*500 m2 containing static and moving obstacles on a Pentium 4 2.8 GHz with 512 MB of RAM. If the computer running the simulator is not fast enough to process the main loop at the set frequency, a black screen is displayed (rendering frequency tends towards zero). Therefore, a more powerful computer is needed to simulate more complex scenarios.

VIII. CONCLUSION AND FUTURE WORK

The simulation software presented in this paper aims to help development of CTS by simulating several cybercars with physics and sensors in a virtual 3D environment. 3D models of worlds and moving obstacles are fully customizable as most 3D modelers can be used to create them. Therefore, it is possible to build very specific environments (traffic situation, crossings, pedestrian…).

The real vehicles are equipped with more sensors (camera, odometers, …) than are currently simulated. Therefore, to keep the simulation software up-to-date, further development will include the addition of new sensor simulation models.

In the meantime, in order to simulate a complete traffic scenario including communication between cybercars, some features may be added to make communication more realistic. The current simulator setup permits communication between all vehicles being simulated, regardless of whether they would be able to communicate in real life (maximum wifi range), as they are all connected via ethernet or WiFi. New modules will be added to emulate the WiFi structure regarding of the relative location of the vehicles in the virtual world and to filter packets that actually can not reach their target.

Moreover, first experiments have shown that cybercars which use the Optimize Link State Routing (OLSR) protocol [14] allow for an ad-hoc network to be created between the vehicles, which dynamically updates itself with respect to the relative position of the vehicles and the radio links present between them. As communication between cybercars seems to be a very important research topic for the next few years in CTS development, a simulation of this protocol based on information available in the simulator could be interesting as well.

REFERENCES

[1] http://opensimulator.sourceforge.net/
[2] http://cycabtk.gforge.inria.fr/
[3] http://www.3impact.com
[4] http://www.ode.org/
[5] http://mgengine.sourceforge.net/v1/
[6] Nathan Koenig, Andrew Howard, “Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator” presented at the 2004 IEEE/RSJ International Conference Intelligent Robots and Systems, September 28 - October 2, 2004, Sendai, Japan
[7] Olivier Michel for Cyberbotics Ltd, “WebotsTM: Professional Mobile Robot Simulation”, pp. 39-42, International Journal of Advanced Robotic Systems, Volume 1 Number 1 (2004), ISSN 1729-8806
[8] Mary Koes, Jijun Wang, Michael Lewis, Stephen Hughes and Stefano Carpin, “Validating USARsim for use in HRI Research”, presented at the 2005 Human factors and ergonomics society meeting.
[9] Eric Guttman, Autoconfiguration for IP Networking: “Enabling Local Communication”, IEEE INTERNET COMPUTING, May-June 2001, p81-p86
[10] M. Parent, “Automated public vehicles: A first step towards the automated highway”, In 4th World Congress on Intelligent Transport Systems, October 1997.
[11] Michel Parent, Arnaud de La Fortelle, “Cybercars: Past, Present and Future of the Technology” in ITS World Congress 2005 ITS World Congress 2005, San Francisco, USA, November 2005
[12] Rodrigo Benenson, Stephane Petti, Thierry Fraichard and Michel Parent, “Integrating Perception and Planning for Autonomous Navigation of Urban Vehicles”, IROS 2006
[13] S. Petti and Th. Fraichard. “Safe motion planning in dynamic environments”. In IEEE-RSJ Int. Conf. on Intelligent Robots and Systems, Edmonton, AB (CA), August 2005.
[14] Laurent Bouraoui, Arnaud de La Fortelle, Anis Laouiti, “OLSR improvement for distributed traffic applications”, Mediterranean Ad Hoc Networks, 2005
[15] http://www.autodesk.com/3dsmax/
[16] http://www.newtek.com/lightwave/
[17] http://chumbalum.swissquake.ch/
[18] http://www.ac3d.org/