An Autonomous Vehicle Controller Using Agent Networks

Daniel Ojeda[1], Lizet Liñero[2] and Ricardo Gudwin[3]

[1]**Universidade Estadual de Campinas**
UNICAMP – FEEC – DCA
Caixa Postal 6101, CEP 13083-970
Campinas – SP – Brasil
danof@dca.fee.unicamp.br

[2]**Universidade Estadual de Campinas**
UNICAMP – FEEC – DCA
Caixa Postal 6101, CEP 13083-970
Campinas – SP – Brasil
llinero@dca.fee.unicamp.br

[3]**Universidade Estadual de Campinas**
UNICAMP – FEEC – DCA
Caixa Postal 6101, CEP 13083-970
Campinas – SP – Brasil
gudwin@dca.fee.unicamp.br

***Abstract* — This work presents the modeling and implementation of an autonomous-vehicle intelligent control system using *agent networks*. We build the whole model over a computational tool called Object Networks toolkit (Ontoolkit – developed by the Computational Semiotics Group, at the DCA/FEEC/UNICAMP) which is used to make the design, implementation and tests of the controller. We further present some results obtained from simulations in order to demonstrate the working of the system.**

I. INTRODUCTION

The problem considered here is a well known problem in literature, that is, the control of an autonomous vehicle moving across an unknown enviroment characterized by obstacles and targets. The main objective here is to navigate the vehicle across the enviroment, reaching the target and avoiding crashes against the obstacles.

Many control metodologies were aplied to solve this kind of problem [1, 2, 3, 4, 5, 8, 9]. Three main approaches are used in solving this problem. The first one is based in a previous knowledge about the enviroment (an *a priori* description of the obstacles and the target). In this case, the problem is usually solved by searching the map of the world. A second way of approaching the problem is by using a reflexive architecture, which doesn't use a world model, but only local sensory information. A third way of approaching the problem is by using a hybrid architecture, i.e. mixing the previous two approaches. In this case, we have a world map, but this is not known a priori, but built using the local sensory information. The system them uses this map to base its actions. In [5], an approach like that is used. The vehicle creates, step by step, a world model using its sensor system. Based on this model, an heuristic search is used to determinate the vehicle's control actions. This mechanism of control is implemented by means of object networks [5], a general mechanism for implementing adaptive discrete event dynamical systems.

Agent networks are a specialization of object networks specially suited for distributed computational systems where there is no centralized control system guiding the coordination between the software components involved. It can be used, then, as a tool for modelling, analysis, design and simulation of intelligent systems.

In this work, we follow the same basic approach given in [5], but now using agent networks instead of object networks. The main gain here is that, when using object networks, each implementation needed to be hardcoded, because the model for object networks is so open that it is difficult to abstract a generic toolkit to use them. Agent networks, instead, can be easily embedded into a generic toolkit, and so are preferrable as a tool of design and development. To implement the system presented here, we used a dedicated computational tool called Object Networks toolkit (ONtoolkit), which is covered in details in [6, 7].

In the next sections, we present the main concepts regarding the vehicle model and its dynamical model. Further, we present the agent network we propose to control the vehicle, detailing on how it coordinates its actions. Finally we present the results, conclusions and the references used here.

II. VEHICLE MODEL

The enviroment for the vehicle is composed by obstacles made of different materials, modeled as rectangles with some physical properties. These properties are: hardness, taste and energy flow. Hardness is related to the difficulty of trespassing the obstacles. A hardness of 1.0 is equivalent to solid (no-trespassing) obstacles. Taste is related to the sensory feeling the vehicle gets when its contact sensors touches the obstacles. They can be positive or negative feelings. And finally, energy flow is the property of an obstacle to be a source or sink of energy for the vehicle. The vehicle has two kinds of sensors (see Fig. 1):

- remote information sensors - RIS (a simplification for a kind of vision mechanism) and
- contact sensors – CS (which are a source of feelings when the vehicle is put in contact to obstacles).

Actuators are of two kinds:

- RIS Position Actuators (determinates a position for the RIS attention region, relative from a point within the vehicle – an angle and length) and

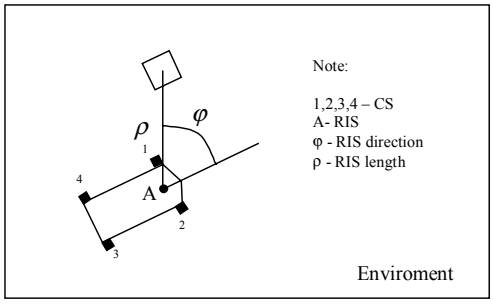


Fig. 1. Autonomous-vehicle sensors.

- Vehicle Movement Actuators (given by a wheel steer angle and a velocity for the vehicle).

The variables of interest for vehicle control are the position (x, y, ψ) inside the enviroment, and the wheel steer angle with regard to vehicle longitudinal axis (θ), (see Fig. 2). Other important parameters are the vehicle nominal velocity (v), the friction constant (μ) and the length between the vehicle's axles (D).

Vehicle dynamics can be expressed using two equation systems. The first, when θ is zero. The second when θ is diferent from zero.

For θ equals to zero, we have:

$$x(k+1) = x(k) + (1-\mu).v.\cos\left(\psi + \pi/2\right) \quad (1)$$

$$y(k+1) = y(k) + (1-\mu).v.sen\left(\psi + \pi/2\right) \quad (2)$$

$$\psi(k+1) = \psi(k) \quad (3)$$

For θ different from zero, we have:

$$x(k+1) = x(k) + \frac{D.sen(\Delta\psi)}{sen(\theta)}.\cos\left(\psi + \pi/2\right) - \frac{D.(1-\cos(\Delta\psi))}{sen(\theta)}.sen\left(\psi + \pi/2\right) \quad (4)$$

$$y(k+1) = y(k) + \frac{D.sen(\Delta\psi)}{sen(\theta)}.sen\left(\psi + \pi/2\right) + \frac{D.(1-\cos(\Delta\psi))}{sen(\theta)}.\cos\left(\psi + \pi/2\right) \quad (5)$$

$$\psi(k+1) = \psi(k) + \Delta\psi \quad (6)$$

Where:

$$\Delta\psi = \frac{(1-\mu).v.sen(\theta)}{D} \quad (7)$$

Analysing equations, we see that besides variables, there are parameters and inputs for the system. The parameter D is always constant. The parameter μ is derived from the hardness of obstacles touching the vehicle. If the vehicle is moving forward (positive nominal velocity), we determine the hardness of the objects touching the front (left and right) points of the vehicle, taking the maximum value of them. We then assure that, if the hardness of one of them is equal to1, the vehicle will not move in this direction (which indicates a collision with an unbridgeable object).

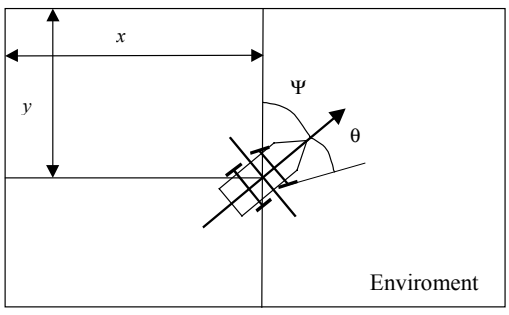


Figura 2. Interest variables of the autonomous vehicle.

The only possible move, in this case will be to move backward, leaving the collision situation. If the vehicle is moving backward, the procedutre is the same, but considering the hardness of the objects located on the back (left and rigth) points of the vehicle. This allows also the detection of collisions when moving backwards, forcing the vehicle to move forward in order to leave the collision condition. Finally, v and θ are related to the nominal velocity and the wheel steer angle - the vehicle's motor control actions.

III. INTELLIGENT CONTROL SYSTEM

The control system implemented through the agent networks is ilustrated in Fig. 3. The main system goal is to generate control actions for the vehicle in order to let it navigate safelly on the environment, avoiding obstacles and reaching the target.

The sensed inputs are:

X, y, ψ	Vehicle position
Ax,ay,aψ	Vehicle position at previous instant
Mx,my	Desired target coordinate
Av	Vehicle nominal velocity at previous instant
Aθ	Wheel steer angle at previous instant
Aφ	Angle between the RIS and the vehicle at previous instant
Aρ	Length between the vehicle and the RIS at previous instant
C	CS values
S	RIS values

The control outputs are:

V	Vehicle nominal velocity to be applied
θ	Wheel steer angle to be applied
ρ	Length to be applied between the vehicle and the RIS
φ	Angle to be applied between the RIS and the vehicle

We now further proceed with a general description of the network operation. The system is composed by 7 modules, divided into two main activities. The first activity is the sensory perception and world modelling activity, which comprises the *IIS – Input Interface Module*, the *PEMM – Perception and Environment Modelling Module* and the *VCM – Visual Control Module*. The second activity is the Behavior Generation activity, which comprises the *PAGM – Points and Arcs Generation Module,* the *PGOM – Plan Generation and Optimization Module,* the *MCM - Motor Control Module* and the *OIM – Output Interface Module*. These modules run independently to each other, being coordinated by the main agent network procedure (the BMSA algorithm, explained in detail in [Guerrero 2000]). We may, although, visually inspect the agent network in order to understand its behavior. At the beginning of each control cycle, the *IIS* is activated to get information from the enviroment, making it available for the whole system operation. After the sensed information get into the system, the *VCM* is activated. This module is responsible for performing a visual search and detecting views with

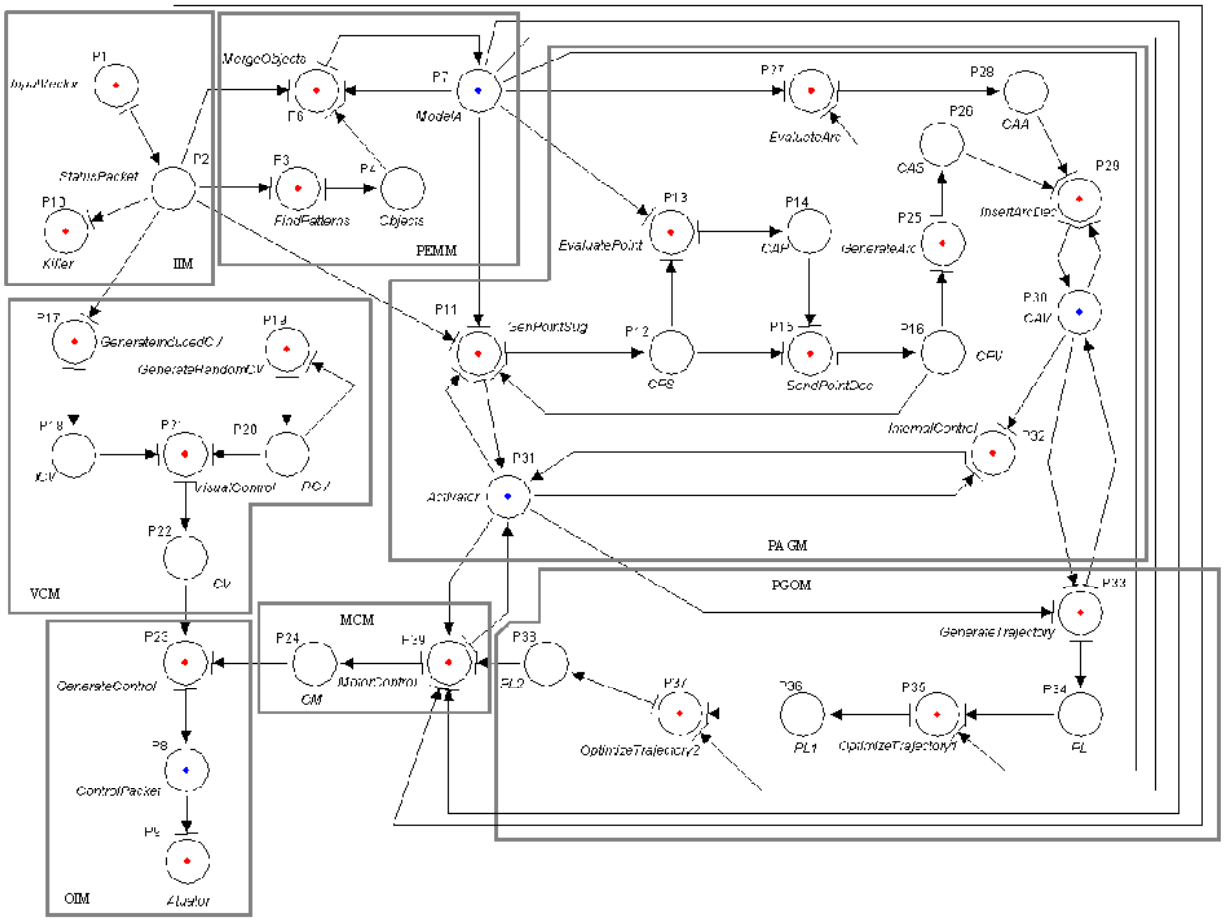


Fig. 3. Intelligent control system for the autonomous vehicle implementd like an agent network.

parts of objects from the world. Once some visual input is detected, the *PEMM* updates the general world model, making a fusion between the new evidence collected and the existing world model. Starting the Behavior Generation activities, the *PAGM* prepares the creation of plans, by generating feasible points and arcs of movement, based on sensorial input and the world model. Using these points and arcs, the *PGOM* builds up and optimize a trajectory to be followed by the vehicle in it's way to the target, resulting in a movement plan. Once the movement plan is obtained, the *MCM* puts itself to track the plan. It is important to notice, however, that the plan can be modified and/or updated at every time. This work is also a responsibility of the *MCM*. Finally, the *OIM* merges the outputs from the *MCM* and the *VCM*, generating the control outputs that are sent to the actuators.

We now proceed with a detailed explanation of each of the modules:

1) Input Interface Module (IIM): This module can be viewed in Fig. 4. It is responsible for taking sensorial information from the enviroment and making it available to the rest of the system. The *InputVector* class stablishes a socket comunication channel between the agent network and the simulation or real enviroment, and starts receiving objects of the *StatusPacket* class. The *StatusPacket* class formats in a convenient way, the data collected from enviroment by means of sensors.

The auxiliary class *Killer* is responsible for destroying the *StatusPacket* objects after they are used by the other network modules, at every control cycle.

2) Perception and Enviroment Modelling Module (PEMM): This module is responsible for processing the information supplied by the *IIM* and updating the enviroment model. It is shown in details in Fig. 5.

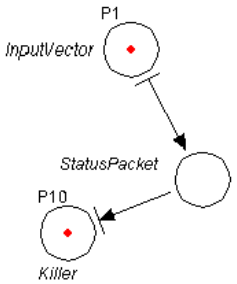


Fig. 4. Input Interface Module

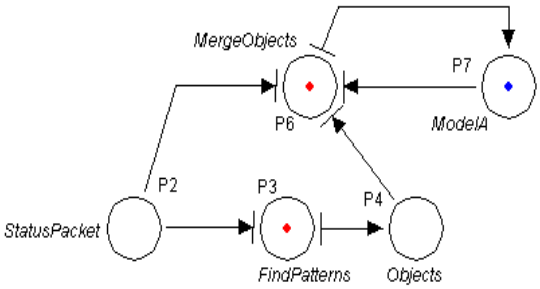


Fig. 5. Perception and Enviroment Modelling Module

The *FindPatterns* class observes the visual sensory information encoded in *StatusPacket*, detecting the presence of objects in the visual field of the RSI sensors. This information is provided in terms of an 8x8 matrix of viewed colors from where it performs a very simple pattern recognition procedure. The objects found by *FindPaterns* are stored into the *Objects* place. The *ModelA* class contains a collection of objects which constitutes the world model The *MergeObjects* class manages the fusion of objects in the *Objects* place with the objects within *ModelA*, performing an update of the world model, that is stored back in *ModelA*. This fusion is performed by means of three simple rules which considers vertical alignment, horizontal alignment and inclusion.

3) Points and Arcs Generation Module (PAGM): This module is presented in Fig 6. It prepares the room for building up plans, by generating a collection of points and arcs that will be used further to integrate feasible plans of motion for the vehicle. It has a somewhat complex behavior. Given one point as a startup, a cloud of points is generated around the given point. This process is perfomed by the *GenPointSug* class. The startup point may have 4 different origins. It may be the current location of the target, or the current vehicle position, or a valid points or a point belonging to the enviroment model. The *EvaluatePoint* class determinates the validity of every point using the following equation:

$$eval = \min_{i}(g_i.e^{-0.4dto(i)}) \qquad (8)$$

Where:
g_i: object(i) taste
dto(i): length to object i.

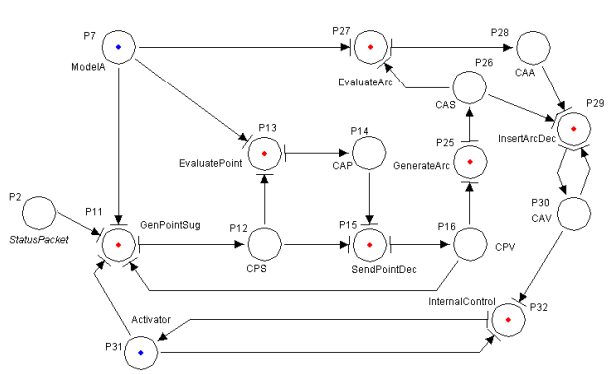


Fig 6. Points and Arcs Generation Module

The valid points are further sent to the *CPV* place. Arcs are then build up from valid points and then evaluated in order to verify:

- if the line linking the source point to the target point intersects with any object of the world model, or
- if there is an intersection with another valid arc already stored in *CAV*, or
- if its length is below a minimum value considered to be irrelevant.

If an arc does not satify any of these conditions, then it is considered valid, and it is sent to the *CAV* place.

4) Plans Generation and Optimization Module (PGOM): This module is shown in Fig. 7. In this module, the valid arcs stored in *CAV* are integrated into plans, which are first generated and further optimized. The *GenerateTrajectory* agent tries to build a plan made by a trivial arc containing the current vehicle location and the target. If this arc is not possible, it takes the arcs where the first point is the current vehicle location, building a kind of a tree, having as root the current vehicle location. It then tries to connect each of the leaves extremities to the target. If it is not possible, it considers extending the tree by adding up new arcs that can be connected to the tree. The procedure do now allow the intercross of arcs. After expanding the tree, its extremities are checked again on its way to the target. This procedure continues over and over, increasing the tree size, which grows along the free space, until a feasible path can be built to the target. When this happens, the procedure stops and a plan is generated. The plan is a sequence of arcs connecting points from the current vehicle position to the target. It is sent to the *PL* place. The trajectory optimization is done in two steps. In the first step, the *OptimizeTrajectory* agent excludes trajectory redundant points. Redundant points are those that, if excluded, still maintain a valid plan. This partially optimized trajectory is sent to *PL1* place.

After that, the OptimizeTrajectory2 agent tries to make a new optimization. In this step a cloud of new random points is generated around each point of the plan, except the vehicle position and the target. A collection of new plans is made out of these points, and they are tested in order to generate the final plan, possibly with some improvements over the prior plan (e.g. a better trajectory). Finally, the optimized trajectory is send to *PL2* place.

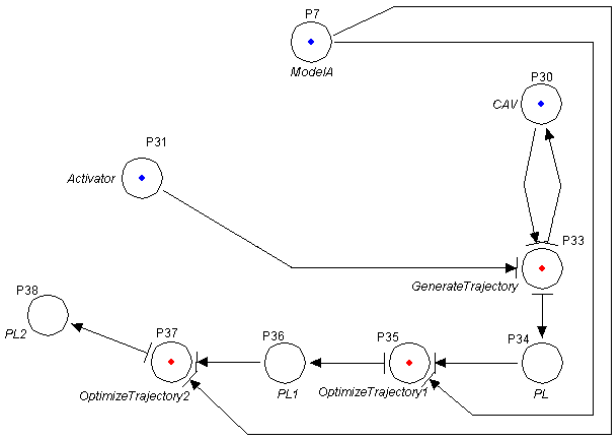


Fig 7. Plan Generation and Optimization Module

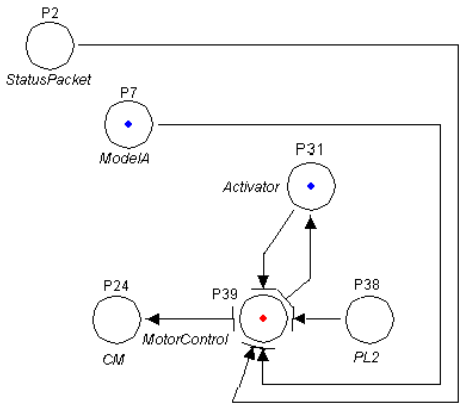


Fig 8. Motor Control Module.

5) Motor Control Module (MCM): This module can be viewed in Fig. 8. Its responsibility is to effectivelly track the plan obtained in the previous stage, generating the control signals Wheel Steer Angle and Velocity. To do that, it calculates the polar coordinates of the next point in the plan (based on the vehicle position), and then turn the wheel angle pointing to the point, arbitrating a positive velocity for the vehicle. In some situations, it may be necesssary to make some maneuvering in order to reach the point. In this case, the *MotorControl* agent should perform such maneuver. When the vehicle position is close to a point on the plan, this point is taken out of the plan and the vehicle is pushed to the next point. The new plan is updated then in *PL2*. The final Vehicle Velocity and Wheel Steer Angle is pushed to the *CM* place.

6) Visual Control Module (VCM): This module is detailed in Fig. 9. It is responsible for the motion of the RIS sensor. Its action is fundamental for the correct functioning of the *PEMM* module. The *GenerateInducedCV* class calculates the center of mass of the visual sensor image and then suggests a motion in that direction, realizing a positional adjustment (compensation) regarding the motion of the vehicle. After the visual control coordinates (ρ, φ) are calculated, they are sent to *ICV* place. The *GenerateRandomCV* agent generates the visual control coordinates (ρ, φ) in a random way, between some previously defined values and sends them to the *RCV* place.

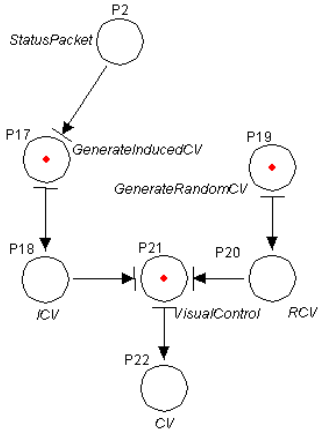


Fig. 9. Visual Control Module.

The *VisualControl* agent sends then the control action to *CV*. If the system is focusing over some object, the control action in ICV is chosen. Otherwise, it will take the random control in RCV.

7) Output Interface Module (OIM): This module is viewed in Fig. 10.

It is responsible by integrating the motor control and the visual control signals, generating the final control action going to the vehicle's actuators. *CV* and CM are the places containing the visual control and motor control values, respectively. The *GenerateControl* agent integrates both controls, obtaining the control action that is sent to the *ControlPacket* place. The *Actuator* agent then builds up a socket channel that sends the *ControlPacket* object to the environment.

All the modules above described maintain a constant coordination wich allows the correct system operation. Some of them operate in a sequencial way and others at the same time (in parallel). This coordination is ensured by the BMSA algorithm ruling out the whole agent network. The final behavior is tuned by the matching functions, which evaluate which objects are going to be used at each time.

The autonomous vehicle controller was implemented as an agents network, using for it, the computational tool ONtoolkit [Guerrero 1999, Guerrero 2000]. To a better and more detailed explanation of "what " and "how" an agent network works, we refer the reader to [Guerrero 2000].

IV. RESULTS

To validate and verify the operation of this controller, we used a virtual enviroment simulator, connected to the Ontoolkit by means of TCP/IP network sockets. Some of the simulations are summarized in Figures 11 and 12. In the majority of cases, the autonomous vehicle reaches the target in a satisfactory way, without crashing with the enviroment obstacles. In some minor cases, some maneuvering was necessary, but performed smoothly.

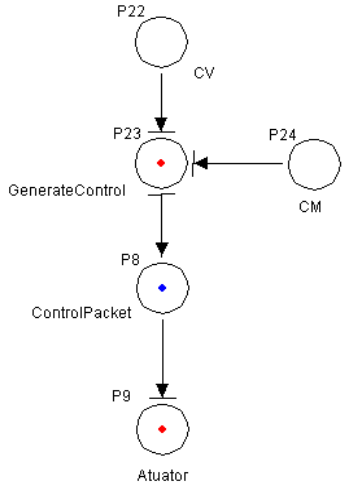
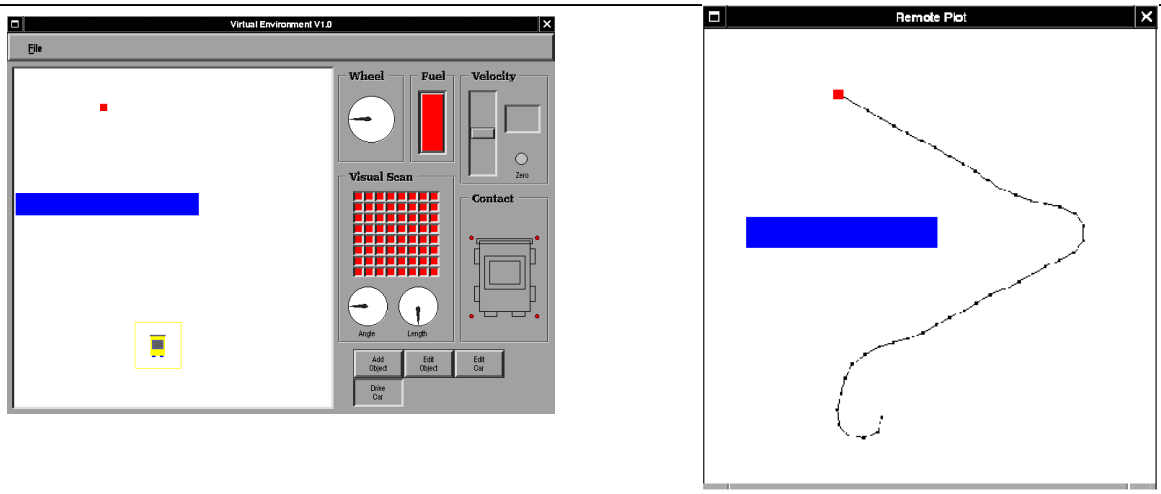
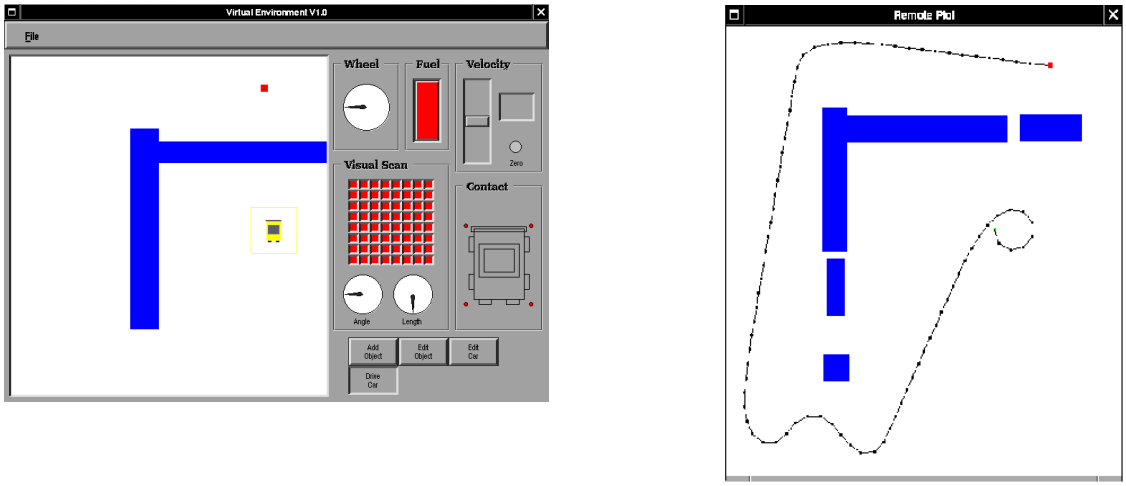


Fig. 10. Output Interface Module.



(a) (b)

Fig. 11. Simulation – case A. (a) Virtual enviroment definition. (b) Simulation result



(a) (b)

Fig. 12. Simulation – case B. (a) Virtual enviroment definition. (b) Simulation result

V. CONCLUSIONS

The main contribution of this work is to validade the ONtoolkit as a viable tool for the developmen of intelligent systems. In this particular case, we developed an intelligent control system for an autonomous vehicle navigating in an unknown environment. The success obtained here shows the potential of the tool and gave us some feedback on how to improve it as a tool, since some minor requirements for future versions were annotated.

VI. REFERENCES

[1] Beom, H.R.; Cho, H.S. “A Sensor-Based Navigation for a Mobile Robot Using Fuzzy Logic and Reinforcement Learning”, IEEE Transactions on Systems, Man and Cybernetics, vol. 25, n. 3, March 1995.

[2] Brooks, R.A. “Intelligence Without Reason” Proceedings of the Twelfth International Conference on Artificial Intelligence, Vol. 1, 1991, pp. 569-595.

[3] Chen, C.X.; Trivedi, M.M. “Task Planning and Action Coordination in Integrated Sensor-Based Robots”, IEEE Transactions on Systems, Man and Cybernetics, vol. 25, n.4, April 1995.

[4] Fan, K.C.; Lui, P.C. “Solving Find Path Problem in Mapped Environments Using Modified A* Algorithm” - IEEE Transactions on Systems, Man and Cybernetics - vol. 24, n. 9, September 1994.

[5] Gudwin, R.R. “Contribuições ao Estudo Matemático de Sistemas Inteligentes”, Tese de Doutorado, DCA-FEE-UNICAMP, Maio 1996.

[6] Guerrero, J.A.S; Gomes, A.S.R; Gudwin, R.R. “A Computational Tool to Model Intelligent Systems”, Anais do 4º SBAI – Simpósio Brasileiro de Automação Inteligente, Setembro 1999.

[7] Guerrero, J.A.S. “Rede de Agentes: Uma Ferramenta Para o Projeto de Sistemas Inteligentes”, Tese de Mestrado, DCA-FEE-UNICAMP, Fevereiro 2000.

[8] Oliveira, M; Figueiredo, M.; Gomide, F. “A Neurofuzzy Aproach to Autonomous Control”, Proceedings of the erd International Conference on Fuzzy Logic, Neural Nets and Soft Computing, August 1994, pp. 597-598.

[9] Taylor, C.J.; Kriegman, D. “Vision-Based Motion Planning and Exploration algorithms for Mobile Robots”,IEEE Transactions on Robotics and Automation, Vol. 14, No 3, June 1998.