Enhanced Graphic Simulator for Dynamic Fuzzy Path Finding Using Potential Fields

Răzvan Tănasie*

**Software Engineering Department, University of Craiova, A.I.Cuza, 13,Craiova, RO-200585, Romania (e-mail: rtanasie@software.ucv.ro)*

Abstract: An enhanced graphics engine was developed using Microsoft C++ and DirectX libraries. This engine is part of a multi-purpose platform used, in this case, to simulate an algorithm for fuzzy dynamic path finding using potential fields. The fuzzy system uses singleton fuzzifier, product inference engine and center average defuzzifier. The developed algorithm is for a dynamic environment with multiple moving obstacles. The inputs are a squared space map, the initial and target position of the wanderer, the initial position and motion parameters of the obstacles. The algorithm computes, if possible, a path from the initial position of the wanderer to its target, using the weighted artificial potential field approach.

Keywords: Path finding, graphics engine, potential fields, fuzzy systems, dynamic systems, simulation.

1. INTRODUCTION

A realistic graphic multimedia platform was developed in order to be able to simulate in a virtual environment both engineering systems and other applications such as games or educational applications (Fig. 1.).

The platform allows a user who is un-familiarized with the advanced graphic techniques to build, at any point, his own engineering applications. He can set their constraints and visualize their graphic simulations in a very pleasant and realistic manner. Also, this platform permits a graphic programmer to properly test his algorithms, without being concerned with the environment development.

As a future development, a standardization method to define the inputs will be developed (for instance, the definition of the kinematic and dynamic models). This will allow an automatization of the use of this platform. Also a unit testing can prove to be very efficient and can be developed in the future.

The platform brings an important innovative and interdisciplinary aspect – there exist simulators for engineering applications (Tarjan, 1987), there exist realist render engines, but there i not a platform that contain both. Also, apart from the scientific impact, there is an impact on the educational environment. The graphic simulations represent tools for educational process quality enhancement. A graphic simulator is much more appealing and easier to understand for the students, thus becoming a stimulant in the learning process. Some of the most important aspects of the platform are: portability, modularity, parallelism, and, very important, extensibility.

The graphics engine is built on top of interfaces that abstract the different graphics API's the engine can use. At the moment it is built around the DirectX API.

The graphics engine handles the rendering API configuration and initialization, and exposes the interfaces used inside the engine for graphics related resource creation, modification and release. It also implements a lighting manager and frustum culling that work closer to the underlying API for increased efficiency and uses a shader system.

It implements the pipeline necessary to draw engine objects on the screen using the materials, lights and effects they specify. It is also used by the interface manager module and the Resource Manager module for graphics resource creation.

Path finding is an important aspect of many domains like robotics, game programming and any kind of movement simulations. The idea of a path finding algorithm is basically simple: given a moving object (that will be called wanderer) in a space, a target that object has to reach, the space map and a set of constraints, the wanderer has to reach the target and, while moving, it has to respect the constraints.

The path computing refers to computing the steps needed to be taken to reach the target. The collision avoidance takes care that the wanderer does not runs into any obstacles. In order to have a good path planning algorithm, these two parts must work well together (Jungnickel, 2004).

A generic fuzzy system consists of four components (Wang, 1997):

- Fuzzy rule base;
- Fuzzy inference engine;
- Fuzzifier;
- Defuzzifier.

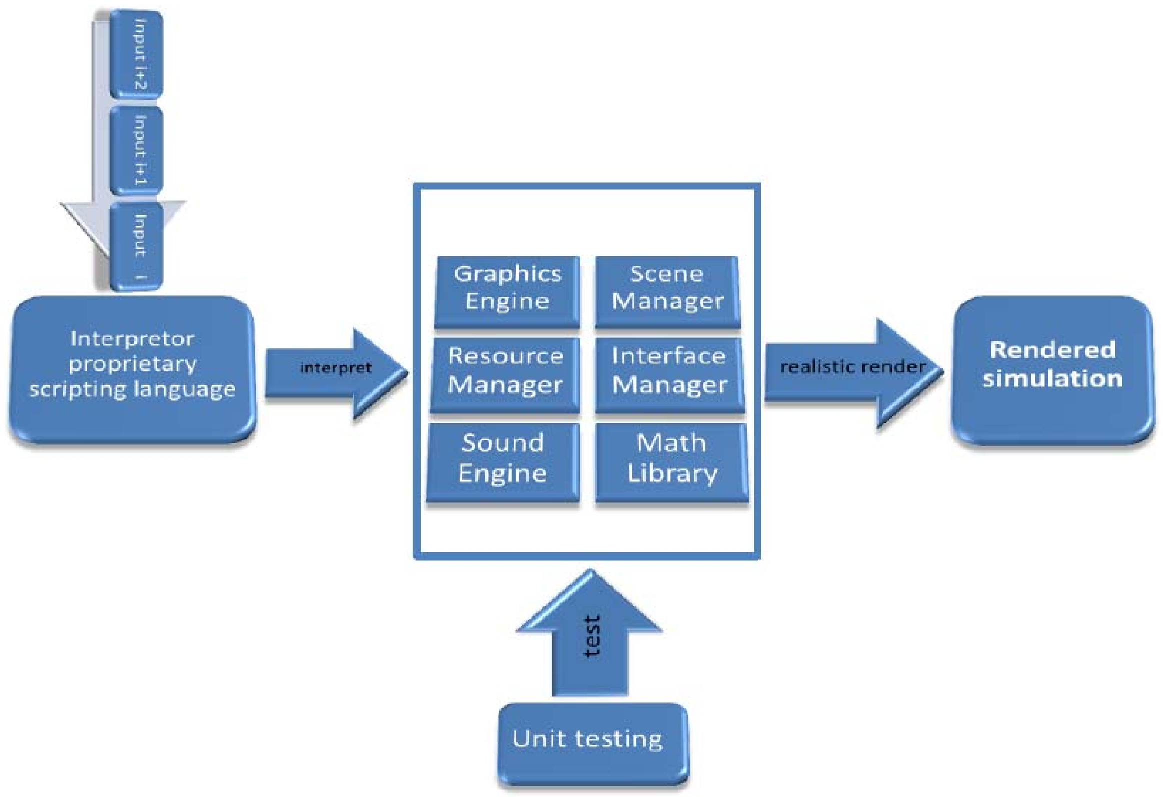

Fig. 1. Graphic simulation platform structure and functionality.

In this application a fuzzy system uses singleton fuzzifier, product inference engine and center average defuzzifier were used to develop an algorithm for a dynamic environment with multiple moving obstacles (Martinez et. all, 1994).

The algorithm proposed is based on the principle of artificial potential fields. This acts as a set of attraction and repulsion forces. The target reach is considered to be the wanderer's scope. It is like an attraction pole for it, while the obstacles are repulsion poles, no matter that they are moving or rigid.

The algorithm computes, if possible, a path from the wanderer's initial position to its target, and uses the artificial potential field approach to compute weights for each of the possible future positions of the wanderer (Khatib, 1986; Kavraki et. All, 1994). The initial computed path is adapted on the way, based on the obstacles motion that might interfere with it.

2. ENGINE STRUCTURE

The engine is made up of several modules that work together in order to handle application needs. These modules are:

- Graphics Engine;
- Scene Manager;
- Resource Manager;
- Interface Manager;
- Sound Engine;
- Math library.

2.1 Graphics Engine

The graphics engine is actually built on top of interfaces that abstract the different graphics API's the engine can use. At the moment it is built around the DirectX 9 API (Sanchez et. all, 2000). The graphics engine handles the rendering API configuration and initialization, and exposes the following interfaces used inside the engine for graphics related resource creation, modification and release:

- void CreateTexture (Texture *texture);
- void ReleaseTexture (Texture *texture);
- void CreateVertexBuffer (CVertexBuffer *pVertexBuffer);
- void LockVertexBuffer (CVertexBuffer *pVertexBuffer, UINT offsetToLock, UINT sizeToLock, void **ppBuff, DWORD flags);
- void UnlockVertexBuffer (CVertexBuffer *pVertexBuffer);
- void ReleaseVertexBuffer (CVertexBuffer *pVertexBuffer);
- void CreateIndexBuffer (CIndexBuffer *ibuff);
- void LockIndexBuffer (CIndexBuffer *pIndexBuffer, UINT offsetToLock, UINT sizeToLock, void **ppBuff, DWORD flags);

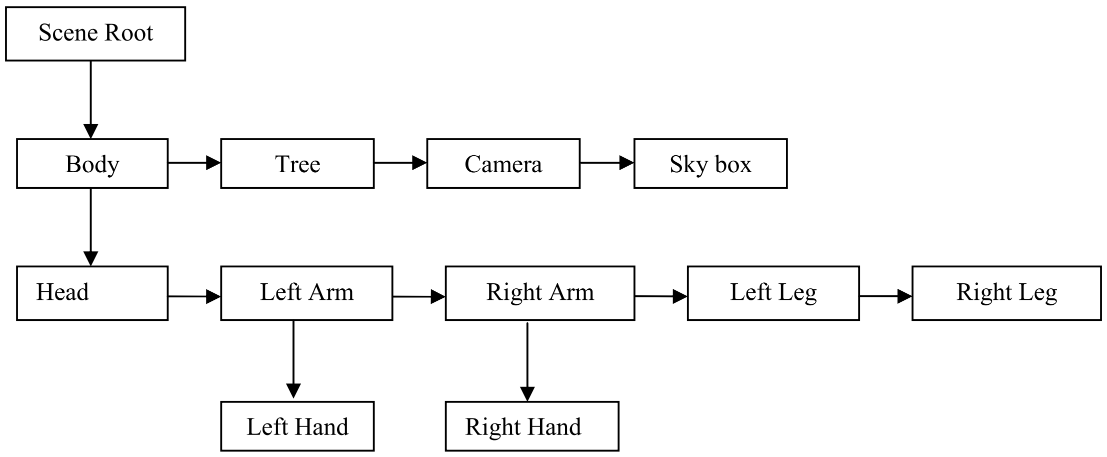

Fig. 2. Object tree created by scene manager.

- void UnlockIndexBuffer (CIndexBuffer *pIndexBuffer);
- void ReleaseIndexBuffer (CIndexBuffer *pIndexBuffer);

It also implements a lighting manager and frustum culling that work closer to the underlying API for increased efficiency.

The graphics engine implements the pipeline necessary to draw engine objects on the screen using the materials, lights and effects they specify. It uses an advanced shader system. It is also used by the interface manager module and the Resource Manager module for graphics resource creation.

2.2 *Scene Manager*

The Scene Manager maintains a tree of objects based upon the hierarchy of the objects in the scene. An example can be seen in Fig. 2.

Each object in the tree has links to its parent, it's next sibling or its first child. A child object is positioned relative to its parent object. If for example the Body object is translated to the right then all of its child objects are also translated to the right, along with their child objects and so on.

In Fig. 2. there are three special objects:

- The scene root object is there only to offer a starting point when traversing the scene tree;
- Camera objects are also present in the scene. If a camera needs to be attached to an object (for example to the head object), it is simply added as a child of that object;
- The Sky box.

The scene manager is also responsible for updating the objects and feeding them to the render pipeline, and for maintaining an acceleration structure for faster spatial object querying implemented through a loose octree.

2.3 *Resource and Interface Managers*

The engine has to deal with several types of resources. It must load, maybe reload, and release these resources as the application progresses. This task is handled by the resource manager.

The resource manager handles the following types of resources:

- Textures;
- Fonts;
- Scripts (Python);
- Sound buffers;
- Shaders;
- Meshes.

Resource Manager operations are executed asynchronously on a separate thread, as to not interfere with engine operation. The resource manager keeps lists of resources to be loaded, resources to be reloaded and resources to be released.

Whenever the application asks for a resource, the manager checks to see if it already exists. If it already exists, a reference to the existing resource is given. If it does not exist, a new resource is created and its id is given to the requester.

If the application wants to load a resource, the resource id is placed in the loading queue, and the resource will be loaded when its turn arrives asynchronously, on the resource manager thread.

The interface manager handles everything related to the GUI initialization, interaction and release. It consists of a hierarchy of widgets and layouts that can be used by the client application to build a graphical user interface.

It also handles the interaction with and between these widgets by the use of menu events. The interface Manager makes use of python scripting for describing widget behavior.

3. FUZZY PATH FINDING ALGORITHM

The algorithm is adapted from one that computes the path for the wanderer in a static environment (the obstacles don not change their initial position) (Barraquand, 1991).

Based on the dimensions of the environment, an *n* x *m* location map is constructed (Tanasie et al., 2007). A location is considered to be a square. In each location can be:

- The wanderer;
- The target;
- Free location;
- An obstacle (either moving or static).

The motion is considered to be a discrete one, and the positions and speeds are multiple of the square dimension (Surmann et. All, 1996). The map can also be implemented as a mesh graph. An example of such a map is presented in Fig. 3., where a black location denotes an obstacle, a red location represents the target, a blue location – the wanderer and a blank location - a vacant space.

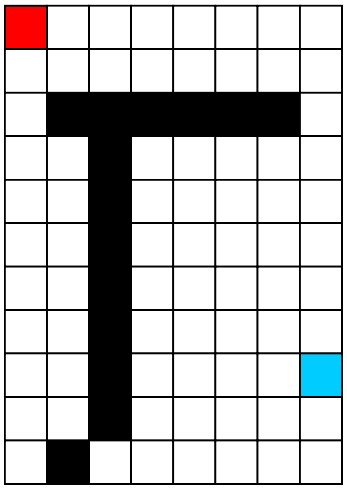

Fig. 3. An example of a map.

As this algorithm considers also moving objects, whose speeds and trajectories can change (with a reduced facto of randomness), it can not guarantee that the target will be reached. It will only end in success if the conditions that will appear allow to there exists at least one way from the initial position of the wanderer to the target.

3.1 Artificial Potential Fields

The algorithm proposed is based on the principle of artificial potential field, more exactly, on attraction and repulsion forces.

The scope of the wanderer is to reach the target. This target is like an attraction pole for it, while the obstacles are repulsion poles. In order to illustrate this in the path finding algorithm, the target is considered to generate an artificial field which induces an attraction force F_a, while the obstacles generate repulsion fields that induce repugnance forces F_r. These forces are strongest near their generator, and grow weaker while moving away from it.

Three main ideas illustrate how these forces work and how they generate each map position weight:

- The main goal is reaching the target;

$$F_r < F_a \tag{1}$$

- The repulsion forces are cumulative (p is the number of obstacles in the operation space);

$$F_r = \sum_{i=0}^{p} F_r^i \tag{2}$$

- Each node (position) weight is given by the composition of all the forces applied to it (W_l represents the weight of node l, F_a^l is the attraction force generated in node l and F_r^l is the summed repulsion force generated in node l).

$$W_l = F_a^l + F_r^l = F_a^l + \sum_{i=0}^{p} F_r^{i,l} \tag{3}$$

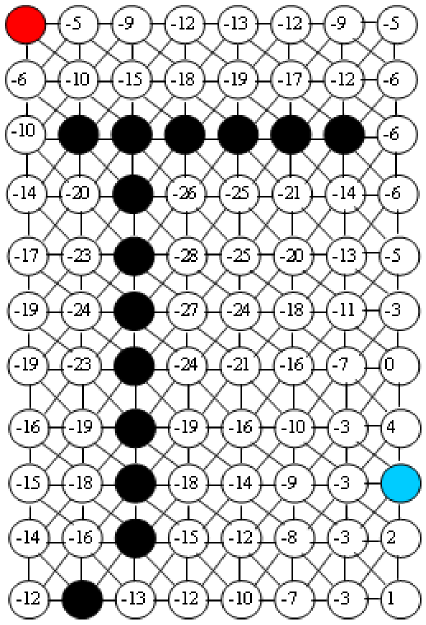

Fig. 4. Weighted graph.

For example, the attraction force can be considered equal to the diagonal of the map minus the distance to the target (in steps), and the repulsion force equal to half of the attraction force. Such a weighted mesh graph for the map in Fig. 3. is shown in Fig. 4.

If such a method can prove to be sufficient for a static environment, a dynamic one, with a degree of randomness in it obstacles motion can be trickier to solve. It order to try to find a suitable solution, the following algorithm is applied:

- First compute the weighted graph for the initial position;
- Compute the foreseen graph for the next step, based on the initial motion information read from the input file;
- Move accordingly;
- For the following steps:
  - Check for collisions and target reach;
  - Compare foreseen positions and speeds and actual ones;
  - If they are the same, keep the same algorithm, otherwise make a new probable next graph based on the history;

Always the shortest path, without collisions and cycles will be tried.

3.2 Fuzzy System

The fuzzy system is basically based on the weights computed using the artificial potential fields adapted with the dynamic rules. That is, the rules composing the fuzzy rule base are related to these weights and determine the wanderer's next move (Buckley, 2005).

The fuzzy system contains six rules, each of them presenting one possible situation that the wanderer may encounter. The fuzzy system is applied after creating an initial path. In fact, only the next step may be computed, the others having no importance.

These instructions would be followed exactly in a crisp manner, using also the above algorithm. In a fuzzy system like this one, each rule adds its influence to the final result (Yen, 2000), specifically it adds (or subtracts) a few (or many depending on the degree of satisfaction of that rule for the current input) degrees to the angle that indicates the next step to be taken.

The membership functions either use the Gaussian bell form, thus probability density functions have been chosen to represent them, or cumulative distribution or partial probability density functions (whichever are more appropriate).

The proposed algorithm uses singleton fuzzifier and product inference engine, which easily eliminates the *sup* part. The fuzzy system outputs a direction angle which will be rounded to the nearest $\pi/4$ multiple.

4. FUZZY PATHFINDING FOR DYNAMIC ENVIRONMENTS USING POTENTIAL FIELDS – ENHANCED SIMULATION

Microsoft Visual C++ language and a proprietary scripting language were used in order to implement the above presented algorithm for fuzzy path finding using artificial potential fields in a dynamic environment with small degree of motion parameters variation.

All the required information for the algorithm simulation is read from an input file. It has the following structure:

- The map dimensions, expressed as a multiple of squares (basic motion unit);
- Initial positions for:
  - Wanderer;
  - Target;
  - Obstacles (both still and moving);
- Wanderer speed;
- Motion parameters of the obstacles:
  - Speeds;
  - Directions;
- Degree of randomness in motion change (recommended less than 10%).

The result of the algorithm is a real time simulation presented in an enhanced 3D visual environment constructed with the use of the developed graphic simulation platform. For this functions from up-to-date graphic libraries, more exactly Microsoft DirectX libraries, were used (Luna, 2003).

The program is modular in order to permit easy modifications of the functions and improvement by adding new modules (for instance, to allow path finding in a 3D dynamic environment).

A simulation example can be shortly seen in Fig. 5-7 (starting with the initial position, an intermediate position, and a final position). As it can be seen, there are both static and moving obstacles in the scene.

6. CONCLUSIONS

A realistic graphic multimedia platform was developed in order to be able to simulate in a virtual environment both engineering systems and other applications such as games or educational applications. Also, a fuzzy path finding using artificial potential fields algorithm for static environments was enhanced to a dynamic environment with multiple obstacles with low randomness in motion change. This was added in order not to allow a pre-computing of the path to the target and impose a real-time simulation (Adams, 2003). This algorithm was used in order to demonstrate the graphic platform capabilities and it computes, if possible, a path from the initial position of the wanderer to its target, using the weighted artificial potential field approach.

The platform allows a user who is un-familiarized with the advanced graphic techniques to build, at any point, his own engineering applications. He can set their constraints and visualize their graphic simulations in a very pleasant and realistic manner. Also, this platform permits a graphic programmer to properly test his algorithms, without being concerned with the environment development.

Also, two significant aspects of the platform are its innovative architecture and purpose and its interdisciplinary character. It is important to point out the educational side, as it can be used as a tool in e-learning or in the regular teaching process, students being more stimulated to lean from a real-time visual simulation.

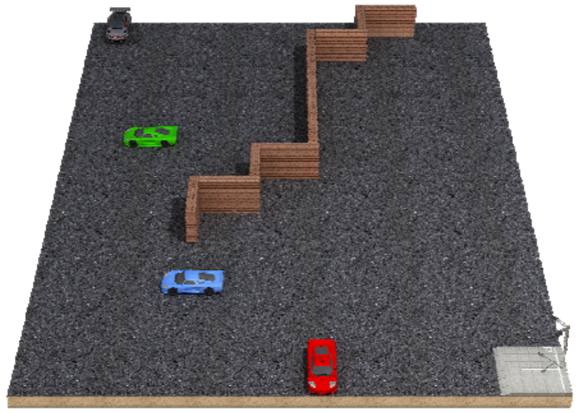

Fig. 5. Initial position.

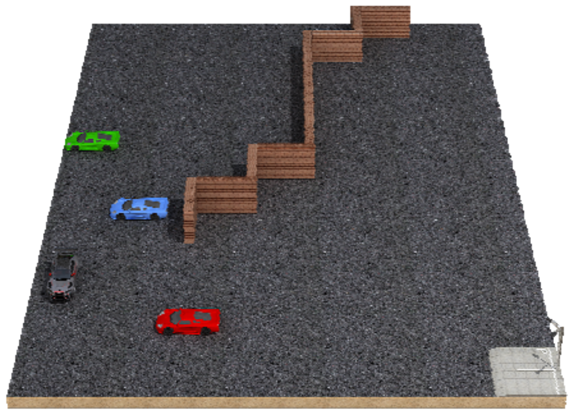

Fig. 6. Intermediate position.

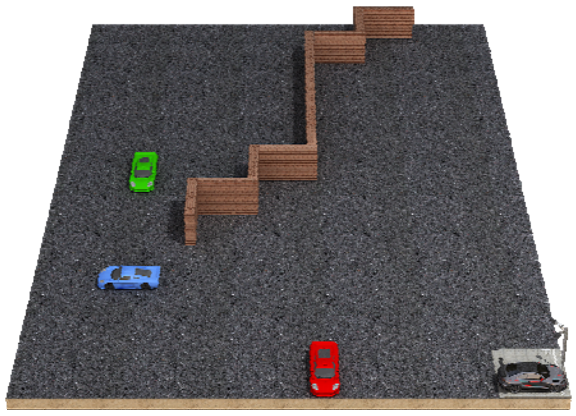

Fig. 7. Final position – target reached.

The tools used for the enhanced graphics engine development were:

- Microsoft Visual C++;
- Microsoft DirectX SDK graphic libraries.

The enhanced graphic engine contains a multi-lighting system, a shader module, frustum culling for increased efficiency, multi-texturing system and, of course, the necessary pipeline to use all of these and more others.

The main design principle of the platform architecture was modularity. Each set of functions with a specific goal has its own module, and modules communicate with each other through interfaces. This leads to the main advantage of such a structure – extensibility.

As a future development, a standardization method to define the inputs will be developed (for instance, the definition of the kinematic and dynamic models). This will allow an automatization of the use of this platform. Also a unit testing can prove to be very efficient and can be developed in the future.

Also, the path finding algorithm enhancement is intended in order to provide solutions for dynamic environments with motion based on behaviors, thus increasing the interdisciplinary character (as a secondary result). A separate branch of the enhancement is developing algorithms for 3D discrete spaces.

ACKNOWLEDGMENT

This work was supported by the strategic grant POSDRU/89/1.5/S/61968, Project ID61968(2009), co-financed by the European Social Fund within the Sectorial Operational Program Human Resources Development 2007-2013

REFERENCES

Adams, A., (2003), *Advanced animation with DirectX*, The Premier Press Game Development Publ.

Barraquand, J., Latombe, J.C., (1991), Robot motion planning: a distributed representation approach, *The International Journal of Robotics Research*.

Buckley, J., (2005), *Studies in Fuzziness and Soft Computing*, Springer, Berlin.

Jungnickel, D., (2004), *Graphs, Networks and Algorithms*, Springer, 2nd edition.

Kavraki, L., Latombe, J.C., (1994), Randomized preprocessing of configuration space for fast path planning, *IEEE Int. Conference on Robotics and Automation*.

Khatib, O., (1986), Real-time Obstacle Avoidance for Manipulators and Mobile Robots, *Int. Journal for Robotics Research*, (5)1, p90-98.

Luna, F., (2003), *Introduction to 3D Game Programming with DirectX 9.0*, Wordware Publishing Inc.

Martinez, A., Tunstel, E., and Jamshidi, M., (1994), *Fuzzy Logic Based Collision Avoidance for a Mobile Robot*, Robotica.

Sanchez, J., Canton, M., (2000), *DirectX 3D Graphics Programming Bible*, IDG Books Wordwide.

Surmann, H., Huser, J., Wehking, J., (1996), Path planning for a fuzzy controlled autonomous mobile robot, *Fifth IEEE International Conference on Fuzzy Systems*, New Orleans.

Tarjan, R. E., (1987), *Data Structures and Network Algorithms*, Society for Industrial & Applied Mathematics

Tunstel, E., Jamshidi, M., (1994), Fuzzy Logic and Behavior Control Strategy for Autonomous Mobile Robot Mapping, *3rd IEEE Int. Conference on Fuzzy Systems*.

Wang, L. X., (1997), *A Course in Fuzzy Systems and Control*, Prentice Hall PTR.

Yen J., Langari R., (2000), *Fuzzy logic, intelligence, control and information*, Prentice Hall, New York.

Tanasie, R.T., Cojocaru, D., Ivanescu, M., (2007), A Real-time Solution for Target Reach in a Static Environment, *16th Int. Conference On Control Systems and Computer Science*.