Expert Systems With Applications 62 (2016) 104–115

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Neural networks based reinforcement learning for mobile robots obstacle avoidance

Mihai Duguleana[a,*], Gheorghe Mogan[b]

[a] Department of Automotive and Transport Engineering, Faculty of Mechanical Engineering, University Transilvania of Brasov, Str. Universității nr. 1, Brasov 500036, Romania.
[b] Department of Automotive and Transport Engineering, Faculty of Mechanical Engineering, University Transilvania of Brasov, Brasov 500036, Romania.

ARTICLE INFO

Article history:
Received 29 October 2015
Revised 9 June 2016
Accepted 9 June 2016
Available online 11 June 2016

Keywords:
Obstacle avoidance
Neural networks
Q-learning
Virtual reality

ABSTRACT

This study proposes a new approach for solving the problem of autonomous movement of robots in environments that contain both static and dynamic obstacles. The purpose of this research is to provide mobile robots a collision-free trajectory within an uncertain workspace which contains both stationary and moving entities. The developed solution uses Q-learning and a neural network planner to solve path planning problems. The algorithm presented proves to be effective in navigation scenarios where global information is available. The speed of the robot can be set prior to the computation of the trajectory, which provides a great advantage in time-constrained applications. The solution is deployed in both Virtual Reality (VR) for easier visualization and safer testing activities, and on a real mobile robot for experimental validation. The algorithm is compared with Powerbot's ARNL proprietary navigation algorithm. Results show that the proposed solution has a good conversion rate computed at a satisfying speed.

1. Introduction

Path planning is one of the key elements of autonomous mobile robots. Since the introduction of mobile robotic platforms back in the '50s, the main desiderate sought by most researchers in motion planning is the development an algorithm capable of providing collision-free trajectories. The subject was divided into two separate research areas, based on the type of environment information which is used by the mobile robot (de Berg, van Kreveld, Overmars, & Schwarzkopf, 2000).

The first approach uses global knowledge of the environment, meaning that at each moment, the robot has complete information about its location, movement capabilities, obstacles and target. This raises additional problems related to localization. Based on a good localization technique, the robot can determine precisely its position with respect to the changing environment (usually, a configuration space C is employed to describe all possible configurations of robot; presuming the navigation takes place in a 2D workspace, C is divided in 2: the obstacles space – C_{obs}, and the free space – C_{free}). Navigating in C_{free} can be achieved through a wide variety of algorithms (such as SLAM – Simultaneous Localization And Mapping (Leonard & Durrant-Whyte, 1991), Wireless Localization based on RSSI (Stoep, 2009), particle filter localization (Dellaert, Fox, Burgard, & Thrun, 1999) and others) and sensorial systems (GPS (Montiel & Sepúlveda, 2014), camera networks, environment markers and so on). As one can infer, it is possible to know in advance if the goal is reachable, which makes this a perfect candidate for artificial neural networks (ANNs).

The second approach uses local information retrieved by range sensors (sonar (Kim & Kim, 2011), laser (Surmann, Nüchter, & Hertzberg, 2003)), infrared sensors (Alwan, Wagner, Wasson, & Sheth, 2005) or video cameras (Seder & Petrovic, 2007). Aside from the fact that there is no guarantee of convergence, one of the main issues which needs to be solved by scientists is the identification and avoidance of local minima. In most cases, this approach doesn't guarantee convergence. Thus, we settle to use global information within this study.

Over the last decades, many researches dealt with the path planning problem. Various types of solutions were proposed: grid-based, potential fields, geometrical or based on artificial intelligence (AI). Grid-based methods involve the overlay of a grid over the C space. In order to obtain a valid path, all grid cells (or grid points) must therefore be included in C_{free}. One of the most common grid-based algorithms in motion planning is VFH (with its variants, VFH+ and VFH*) (Borenstein & Koren, 1991; Ulrich & Borenstein, 1998). Potential fields model C after a potential function: obstacles are seen as repulsive entities while the target is seen as an attractive center. The workspace is regarded as an

* Corresponding author. Fax: +40 268 418967.
E-mail addresses: mihai.duguleana@unitbv.ro (M. Duguleana), mogan@unitbv.ro (G. Mogan).

http://dx.doi.org/10.1016/j.eswa.2016.06.021
0957-4174/

isolated universe, which works towards minimizing its potential energy, thus pushing the moving entity (the mobile robot) to the goal (Borenstein & Koren, 1991). Among geometrical methods, the most used are cell decomposition and the visibility graph (Barraquand & Latombe, 1991). The visibility graph is constructed based on clusters of inter-visible points within C_{free}. Using a traversing algorithm such as Dijkstra, A* (Dechter & Pearl, 1985) or D* (Stentz, 1994), the shortest or optimal paths are computed (Lozano-Pérez & Wesley, 1979). Latest research in motion planning employs the use of artificial learning techniques. Q-learning has been used in Jaradat, Al-Rousan, and Quadan (2011) to achieve motion planning in dynamic environments. The authors limit the number of states from the states space, thus reducing the size of Q-table and indirectly, the computation time; however, convergence is not guaranteed. Inspired by bird flocking, particle swarm optimization (PSO) is also widely used in motion planning (Qin, 2004). Each particle from the space of solution candidates tries to achieve the goal optimally, and improves its "experience" after every new iteration, based on its trajectory history and on the "experience" of other neighboring particles. Another widely exploited motion planning method is fuzzy logic (Reignier, 1994). Last but not least, neural networks were used to achieve obstacle-free trajectories (Dezfoulian, Wu, & Ahmad, 2013; Fierro & Lewis, 1998). Although many of these AI methods show promising results, just a few actually target the avoidance of dynamic obstacles, and even less fewer implement the proposed motion planning solution in real testing environments for experimental validation.

A multi-layer neural network is able to map non-linear functions (Hecht-Nielsen, 1987). This feature can be used in conjunction with reinforcement learning in order to solve the path planning problem, given prior knowledge of the environment. For this specific case, Q-learning (Russell & Norvig, 2002) was used with the following function that quantifies the quality of a state-action:

$$Q : S \times A \rightarrow \mathbb{R} \tag{1}$$

where Q is the set of solutions, S is the set of states and A is the set of actions. The cost, or better said, the reward for a collision-free trajectory is given if the mobile robot reaches the goal. In other words, the proposed solution samples each state, action and result from the workspace as an underlying probability distribution which helps in calculating the reward parameter. For fast convergence, the solution makes further use of a feed-forward neural network. Thus the proposed, solution usually find a collision-free trajectory from the first few epochs.

The motion planner is implemented in VR for initial testing and efficient visualization, and after achieving satisfying results, on a real mobile robot: PowerBot from Mobile Robots ("PowerBot website", 2015).

2. Literature overview

This study encompasses aspects from multiple research areas. A brief resume of current achievements in these areas is proposed bellow.

2.1. Obstacle avoidance of mobile robots in dynamic environments

Avoiding collisions with moving obstacles is a challenging task. In order to solve this problem, a large number of algorithms using both local and global knowledge were proposed by researchers.

Knowing only local information about the working environment presumes, in most cases, the usage of a reactive approach, such as directional or velocity-based methods. Directional methods calculate geometrically the robot's trajectory (Khatib, 1986; Minguez & Montano, 2004). Knowing the exact coordinates of the robot and of the obstacles, the path planner can simply calculate the Euclidian distance at each time instance, and by setting a lower limit to this variable, the robot can move on collision-free trajectories (Asano, Guibas, Hershberger, & Imai, 1985). Velocity-based methods consider the kinetic energy of the robot and of the closest recognized moving obstacles, and use this data in trajectory generation (Large, Laugier, & Shiller, 2005). The most used velocity-based method is Dynamic Time Window, introduced back in 1997 (Fox, Burgard, & Thrun, 1997). One of the biggest issues with reactive methods is that they need a good sensorial system which can produce accurate position coordinates for any local obstacles. Latest studies use video cameras to get environment information and to estimate the dynamics of the scene. For example, a single camera can be used to either detect landmarks and environment cues, or based on an algorithm such as the Block-Based Motion Estimation (Kim & Do, 2012), to detect and classify moving obstacles. Multiple cameras provide stereoscopic vision, making depth perception much easier (Chilian & Hirschmüller, 2009). Another type of sensor introduced on the market in the last decade is the time-of-flight (TOF) camera (May & Werner, 2006), a hybrid between laser range sensors and classic video cameras.

Usually, obstacle avoidance algorithms based on local information of an environment with a fairly large amount of obstacles rely on selecting the obstacle which is most likely to collide with the robot. This strategy is however hard to implement on real mobile robots, since the selection process itself is subject to many questions such as:

1. Are all the obstacles properly sensed?
2. Is this the closest obstacle?
3. Is this the most dangerous obstacle?
4. What if there are 2 or multiple obstacles closing at the same time?

Considering a global representation of the dynamic environment is available, some of the most used navigation algorithms rely on variations of the potential field method. Cases include specific situations when for example both the target and the robot are moving (Ge & Cui, 2002; Huang, 2009), or the use of harmonic functions in order to completely eliminate the local minima (Kim & Khosla, 1992). Others use an integrated representation of the workspace (Savkin & Wang, 2014) or analytical approaches (Qu, Wang, & Plaisted, 2004) to achieve collision-free trajectories.

One of the main issues that drifted researches away towards unconventional motion planning algorithms is the computation time. Lately, several studies employ the use of AI, since many methods converge faster, are easier to implement and produce in some cases more satisfying results.

2.2. Path planning with artificial intelligence techniques

There are many artificial intelligence techniques used for solving path planning. Among these, fuzzy logic was the first to be used (Reignier, 1994; Saffiotti, 1997; Yen, 1995). Fuzzy logic is great for static workspaces, but produces weak results in dynamic environments. Also, fuzzy-computed trajectories are not optimal. Genetic algorithms (GAs) followed shortly (Sugihara & Smith, 1997). Due to their specific, GAs are great at finding global optimal trajectories. However, they do not scale well with highly complex environments, and finding a good fitness function for the motion planning problem is rather difficult. Hybrid methods emerged, which used classic algorithms such the potential fields, together with AI techniques (such as GAs), for improving the solution (Vadakkepat, 2000). However, the path planning performance is still weak due to the limitations imposed by the potential field model used in the study. PSO also started to be used for achieving collision-free robot navigation (Kennedy, 2010; Nasrollahy, 2009). However, the results

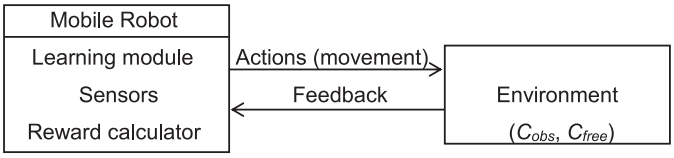

Fig. 1. Paradigm of reinforcement learning for mobile robots.

are computed slowly and are highly dependent on the shape of the obstacles.

ANNs have been used to solve the path planning problem for mobile robots. Several types of neural network planners were proposed in the last 2 decades (Pomerleau, 2012). Latest studies investigate using ANNs together with additional techniques such as fuzzy logic or genetic algorithms for improving both the reliability, training times and the convergence rates of the proposed path planners.

Neuro-fuzzy controllers were also used to achieve a collision-free navigation (Rusu, Petriu, Whalen, Cornell, & Spoelder, 2003). In this study, the fuzzy logic module is based on a set of primitive. The ANN is used if a turning primitive ("Go-Tangent") is activated, and computes the turning angle. However, the obstacles from this study are considered always normal to the robot's main axis center, and like all fuzzy logic approaches, the planner is unable to reach optimal trajectories.

One of the most interesting approaches (which also makes the subject of this study) is reinforcement learning (RL). RL allows a mobile robot to adapt its trajectory, no matter how complex and cluttered its working environment is. The goal of any robot engaged in a RL algorithm is to maximize its reward. Thus, by repeated interactions, the robot can learn about its trajectory in real time, just like any living creature, the key desiderate for all autonomous artificial entities. The terminology was in fact borrowed from the study of animal learning (Mahadevan & Connell, 1992), and was applied to robotics based on the paradigm presented in Fig. 1.

This study uses a relatively new RL algorithm called Q-learning, which was introduced back in 1989 (Watkins, 1989). One of the strengths of Q-learning is that is doesn't require any previous information about the environment. Moreover, after defining a reward function, the robot finds the optimal path by itself (finds the maximum achievable reward). These advantages lured researchers, which started to systematically study this type of unsupervised learning. One of the first studies that has implemented Q-learning in robot navigation used a relatively simple reward function (Eq. 1). 1 was assigned to the goal state, -1 for collision state and 0 for any other states (Smart & Kaelbling, 2002). Although the results were promising (the robot eventually converged to the solution), the computation times were high. Q-learning based robot navigation was greatly improved in Jaradat et al. (2011), but still over 100 training iterations are required to train the Q-table (the matrix holding Q-values).

In real environments, the workspace of a mobile robot may contain many different obstacles. This increases the amount of Q-values needed to be stored, which in turn translates into high computation times. If the environment is excessively large, the state space of rewards may be so sparse, it would also take long periods of time to explore it thoroughly. That is why some studies tried to combine Q-learning with other means of AI, in order to increase the computation performance.

Fuzzy logic was used together with Q-learning in Boubertakh, Tadjine, and Glorennec (2010) and Khriji, Touati, Benhmed, and Al-Yahmedi (2011). In both studies, although complex environments are targeted, the algorithms were not tested for moving obstacles. In all fuzzy logic based path planners, the fuzzy variables that are assigned to the state and possible actions of the robot take a finite number of values. This property makes fuzzy logic suitable to be used in combination with both delayed reinforcement learning techniques as well as ANNs (neuro-fuzzy). On the other hand, handling large sets of state-action pairs is manageable with neural networks, which can be successfully used to store and compute Q-values (Huang, Cao, & Guo, 2005). However, this study uses some empirical values for the reward function (0.02 for forward movement, - 0.01 for rotation and -1 for collision) and although it shows improved performance with respect to simple Q-learning, it doesn't explore the possibility of fine tuning its parameters for even better solutions. The neural Q-leaning approach is suitable to be used in controlling i.e. robotic arms (Duguleana, Barbuceanu, Teirelbar, & Mogan, 2011), but also in other research areas such as economics (Tesauro, 2001).

2.3. Mobile robots in VR

Maneuvering robots, whether they are fixed or mobile, has always posed a challenge, especially to new, uninitiated persons. Modeling these robots in VR may compensate for additional requirements of space, time and money. In the latest 2 decades, helped by the emergence of better personal computers, various initiatives brought the audience closer to robots. One of them is the Player/Stage/Gazebo project (Collett, MacDonald, & Gerkey, 2005), a free open-source simulation environment. The idea of a robotic framework was also explored by both independent researchers (such as Olivier Michel which developed Webots in 1996, a licensed software widely used in robotic competitions such as RoboCup (Michel, 1998), or Louis Hugues which developed Sinbad in 2005, an open source software written in Java (Hugues & Bredeche, 2006)) and commercial entities such as Microsoft, with its Robotics Developer Studio which has now reached the 4th version (Johns & Taylor, 2009).

Robotic simulation has been used in education, virtual testing, industry and entertainment. There are both advantages and downsizes of using virtual mobile robots. As positive points, one can list the easiness of the implementation (no more time spent on setting additional safety systems), the answer to the concurrent use problem (multiple researchers can perform experiments on the same simulated robot) and the increased intellectual flexibility (uninitiated personnel can now be actively involved in the testing phase, without the fear of accidents) (Duguleana & Barbuceanu, 2010; Haton & Mogan, 2008). However, there are also some drawbacks which need to be considered. The inexact reality representation triggers secondary problems such as the absence of noise in recorded data sets. Various factors interfere with the robot in real environments, such as the variable friction coefficient, the air pressure in robot's tires caused by parameters which are hard to quantify (such as air pressure, temperature and others), power level, sensor readings and so on. These introduce differences between the simulated scenario and the real world. Another factor which also brings inconstancies is the time taken to compute the path planning algorithms. In dynamic environments, everything happens fairly fast, thus the motion planning algorithms must converge rapidly. Last but not least, simulating mobile robots presumes spending additional time in order to develop and improve the realism of the VR scenarios, which should be taken into account, especially when talking about the benefits of any simulation software (Sheridan, 1992). Aside from 2D MATLAB representations which are widely used in the literature, just a few researchers actually implement mobile robots and test their path planning algorithms in VR (Chen and Chen, 2014). In this study, VR is used for effectively visualize the iteration process and the final trajectory of the modeled robot. The VR module doesn't interfere in any way with the performance of the trajectory planner.

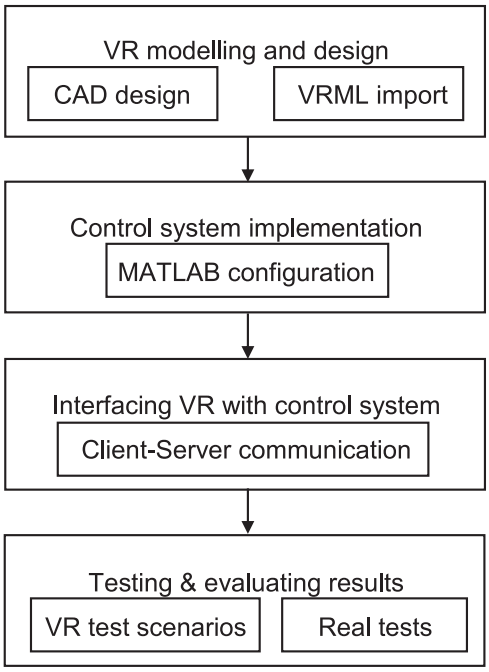

Fig. 2. Experimental methodology.

3. Hardware and software prerequisites

The work involved in the experiments presented in this paper is based on several principles:

- *Performance.* The software written needs to work as fast as possible, so authors have taken all the measures (to the best of their knowledge) to enhance as much as possible the algorithm code.
- *Flexibility.* The software needs to be as scalable as possible, thus an object oriented programming paradigm was used.
- *Robustness.* The software should be easily executed on various operating systems.

The experiments were conducted according to the paradigm presented in Fig. 2.

In the first step, VR modeling and design is performed in C++/VRML. The robot is first modeled in CATIA, then exported in VRML. VRML (Virtual Reality Modeling Language) is a standard file format commonly used in VR applications. The control system is implemented in MATLAB, a widely used software with specific features for neural networks calculus. Considering the study requirements, the 2 development applications need to be integrated into a single program. This is achieved through the controller presented in Fig. 3. A configuration file is used to control the movement of all robots from the scene. The data is sent wireless to the mobile robots, after the control system calculates the trajectories. The results are quantified based on the readings received from both programming environments.

3.1. Physical structure of the mobile robot

Although the purpose of this article is to present a new obstacle avoidance algorithm suitable for navigation tasks in unknown dynamic environments, the aim of the experiments presented in this paper is to control a specific mobile robot – PowerBot. PowerBot is a non-holonomic robotic platform produced by an US company named Mobile Robots. It has the physical structure presented in Fig. 4. It features 2 powered front wheels and 2 pivoting real wheels.

3.2. Modeling the work environment

Real working environments contain obstacles of various shapes and sizes. For simplicity, the 2D virtual environment created for this experiment contains 2 classes of obstacles: circles and spheres. These however can be extended to any type of geometrical figures. The workspace is set up in MATLAB by providing a configuration text file. The configuration is set up as a hierarchy of two levels: the objects on the first level and the parameters on the second. The objects are marked with [<type>] and are followed by their parameters. The possible types of objects are (Fig. 5):

- Workspace – The workspace itself can be configured here. It has as parameters its time resolution (res), and the length/width of the space.
- Obstacle – The obstacles are defined by the following attributes: motion (type of motion: *velmove* – a simple motion in some direction defined by the velocity parameter, *rndmove* – a random motion based on the magnitude of the velocity parameter, *sinemove* - a sinusoid motion based on magnitude of velocity parameter), position (in Cartesian coordinates [x y]), velocity (in Cartesian coordinates [v_x v_y]), type (circle or square) and size.
- Target – The target, which is seen as a regular obstacle, sharing the same attributes.
- Robot – Aside from obstacle parameters which are inherited by the robot entity, a robot object has additional attributes described in Section 3.3.3.

Fig. 5 presents the scene plotted in MATLAB (left) and the real testing environment used in the experiments (right), at different time instances.

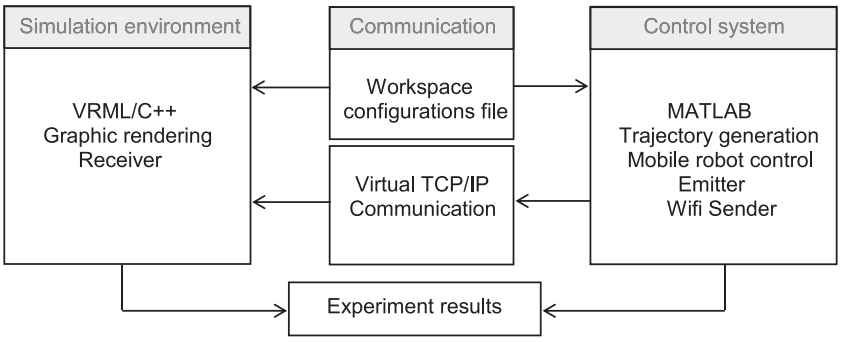

Fig. 3. Application architecture.

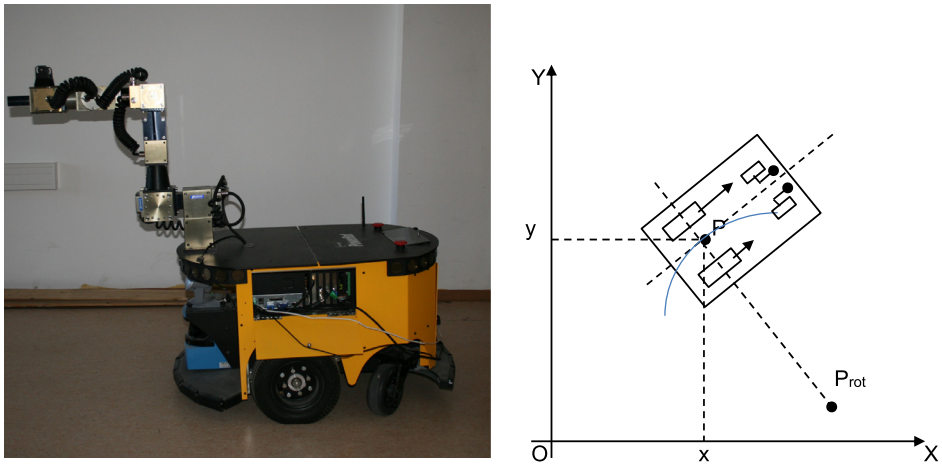

Fig. 4. Powerbot (left) and its cinematic structure in XOY coordinates (right).

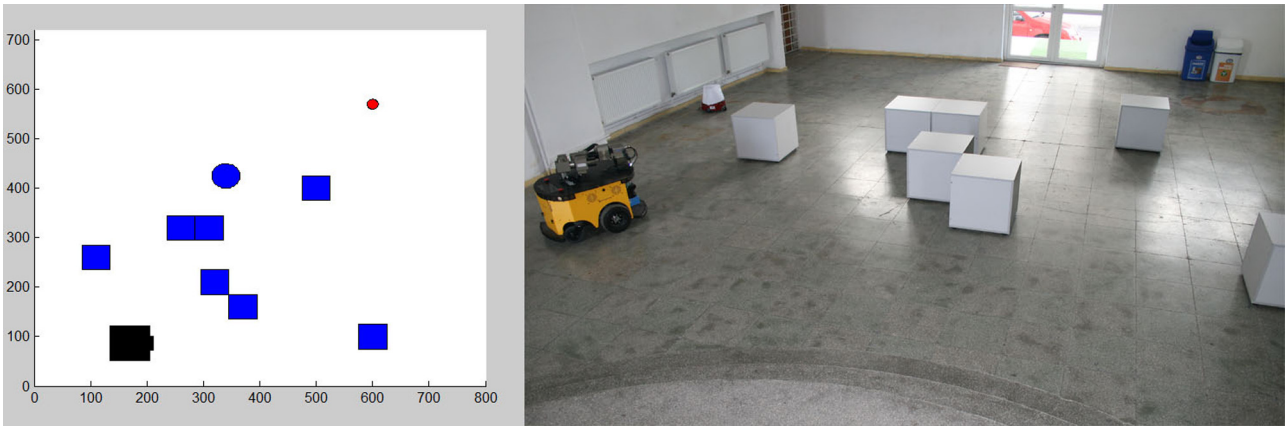

Fig. 5. A 7 m × 8 m workspace with a robot, 7 fixed obstacles, 1 dynamic obstacle and a target.

3.3. Modeling the robot

As presented in Fig. 2, the first phase in the experimental methodology of this study is to model the VR scene. The robot structure is the most important entity of the study, thus it requires an increased attention. In order to obtain a simpler, easier to use model, specific assumptions were made.

3.3.1. Assumptions

The robot operates in a workspace with static and dynamic obstacles. Its starting configuration, its velocity and the dynamic parameters of the obstacles and of the goal are implemented in configuration files, with direct relationship to the real environment. Aside this information, the following assumptions were made:

Assumption 1. All dynamic parameters are known at each time instant, including the Cartesian position and the velocity of the robot, of the obstacles and of the target.

Assumption 2. The robot has a velocity higher than the velocity of the target, a common sense condition that enforces convergence.

Assumption 3. The robot is activating in an unknown working space. The collision detection is achieved in simulated environment based on the Euclidian distance from the margin of a virtual sphere that includes the robot structure, to the margin of a virtual sphere that includes the obstacle. In real environments, collision is detected either by laser or sonar sensors, or by bumpers, once the robot physically touches the obstacle.

Assumption 4. The robot is non-holonomic and is activating in a 2D space. This assumption can be easily included in the control paradigm and transposed into the real environment, as the smoothness of the computed trajectories depends on the resolution of the virtual environment. Slightly increasing the threshold between 2 adjacent time samples ensures the real robot has plenty of time/space to reach any 2 neighboring trajectory points with the dynamic parameters presumed by the algorithm.

3.3.2. Modeling the robot in VR

Several pieces of software have been used to create the PowerBot robot in VRML. First, the robot was modeled in CATIA, a CAD software used in a wide variety of engineering tasks. The CAD model was exported directly to VRML format (Fig. 6). A software viewer like BS Contact or Instant Reality is used to run and navigate into the VR scene.

3.3.3. Modeling the robot in MATLAB

Aside the common attributes derived from the obstacle class, a robot entity has specific configuration parameters suitable for this study: dmin (the minimum distance at which the robot should start taking avoidance decision), gamma (the learning rate of the Q-function), capacity (the number of obstacles whose quadrants drive the state), train (a flag which allows exploration - randomness within training phase) and depth (a parameter which tracks distance to obstacles over a certain number of previous iterations).

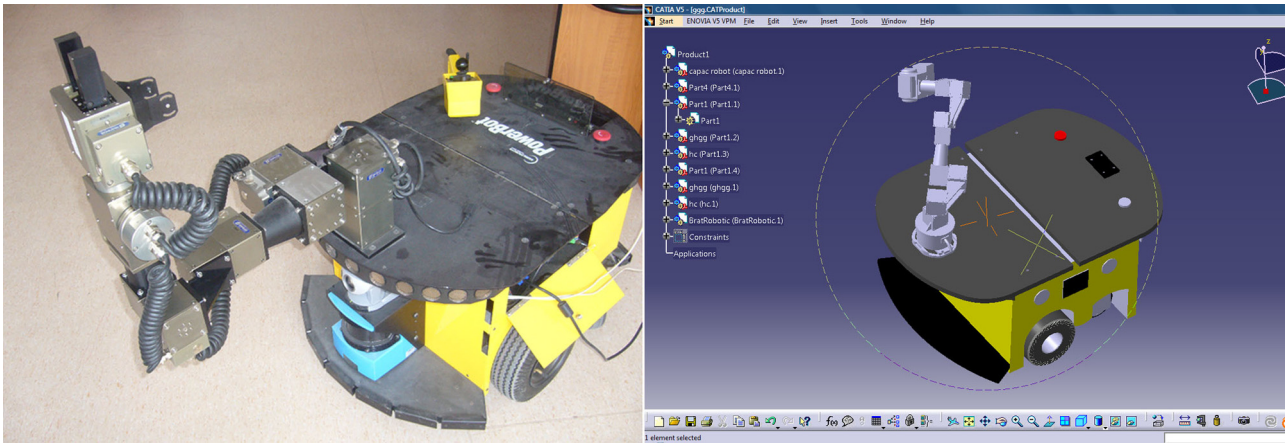

Fig. 6. PowerBot and resulted CAD model.

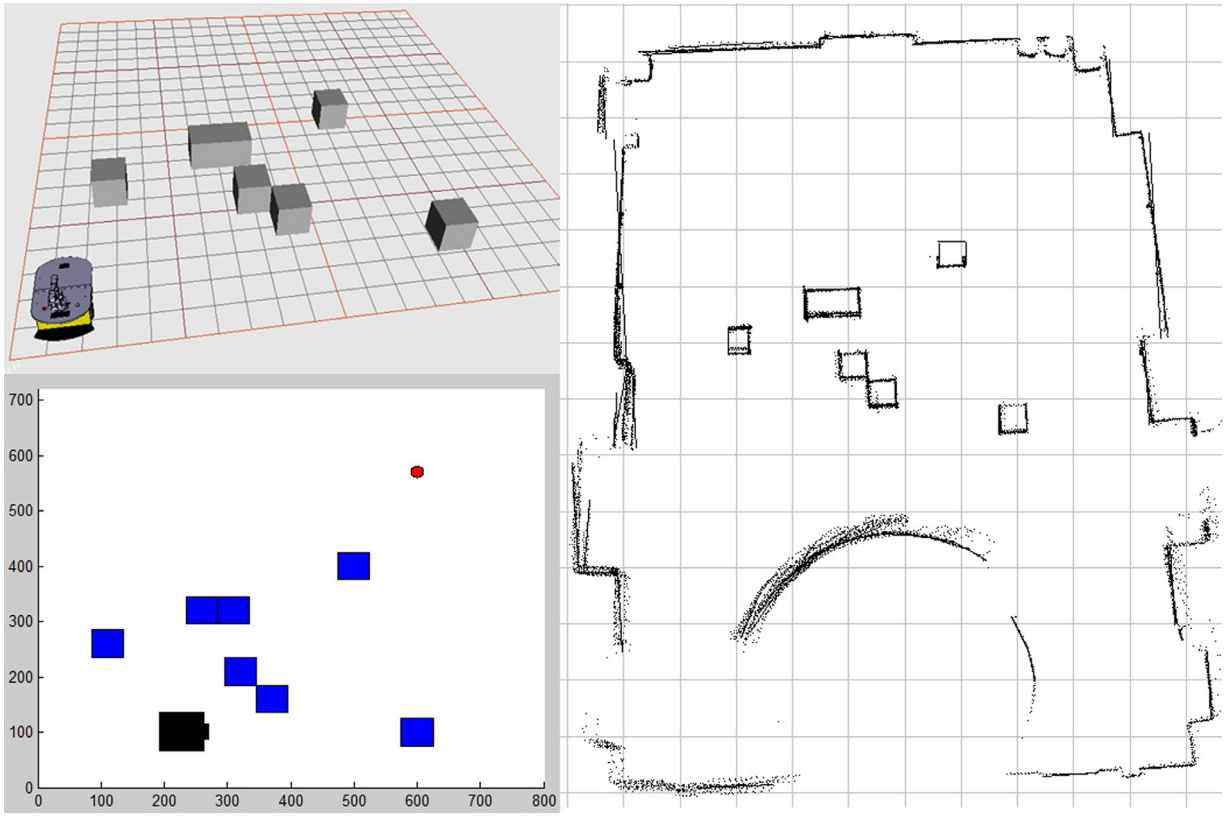

Fig. 7. MATLAB, C++/VRML and SLAM map equivalence.

3.3.4. Interfacing MATLAB with VR

Fig. 3 presents a stand-alone application. In order to implement this, the link between MATLAB and VRML is constructed in C++. There are several methods suitable to achieve this. One of them is creating a shared C++ library via MATLAB compiler. However, custom neural network directives cannot be deployed in a specific library. Another method is to call MATLAB engine directly from the project build, but this raises variable performance issues, depending on the technical configuration of the station used. For this study, the 2 separate environments are communicating by making use of a virtual TCP/IP connection. MATLAB acts as an emitter, sending trajectory data (Cartesian coordinates at each time sample) to a C++ receiver. After data is received, the C++ program interacts with VRML and updates the view. The 2 programming frameworks share the same configuration file, which produces an equivalent experimental scene presented in Fig. 7. Since the client-server architecture is virtual, the connection is instantaneous, with no implications over the performance of the planner.

4. Trajectory planner

The path planning algorithm proposed by this paper takes the next step in this specific research field, by presenting a new solution based on Q-learning (Watkins & Dayan, 1992) and neural networks that can be regarded as a “self-learning” paradigm. This solution was already applied with success in controlling robotic manipulators (Duguleana et al., 2011), but its flexibility allows us to use it in mobile robot navigation, in order to obtain collision-free trajectories in dynamic environments.

4.1. Designing the trajectory planner algorithm

The problem has been reformulated mathematically as follows. Let A be an array of 2D Cartesian coordinates, with p_s the starting position and p_e the end position of the mobile robot. The kinematic mapping of the trajectory of the robot is described as:

$$T(A(p_s, p_e), t) = \left\{ \mathrm{U}_{t=0}^{t-final}\, p_t \;|\, p_t \text{ the Cartesian coordinates at moment } t \right\} \quad (2)$$

Let the sets of points $O_1(t)$, $O_2(t)$, ..., $O_n(t)$ represent the n obstacles located in the workspace at moment t. Given the start configuration p_s, avoiding obstacles and reaching the goal x_e resumes to finding $T(A(p_s, p_e), t)$ while satisfying both equations:

$$p_e(t) = x_e(t) \quad (3)$$

$$T(A(p_s, p_e), t) \cap \left\{ \mathrm{U}_{i=1}^{n}\, O_i(t) \right\} = \phi \quad (4)$$

The problem formulated above can be solved with Q-learning and neural networks. At any given time t, the mobile robot is in an intermediate state p_t and can choose among different possible future states. The two conditions formulated in Eq. 3 and Eq. 4 can be represented within the trajectory planner architecture proposed in Fig. 8.

The condition expressed in Eq. 3 means that the mobile robot achieves the target (the algorithm converges). This condition is modeled by Pos-Net neural network, which is a 30-20-3 Multi-Layer Perceptron (MLP). 30 neurons are used on the first hidden layer and 20 neurons on the second hidden layer, as preliminary tests showed that this configuration is suitable and provides a good mapping of this nonlinear problem. Pos-Net receives as input the p vector which contains the current position of the robot, the time sample t and the matrix of Q-values. It outputs a 3 element vector which holds the Cartesian values and the time. The weights of Pos-Net neural network are updated after each step, using the adapt function, which receives as input the Cartesian coordinates of the goal. After adapt function is applied, a 3-value output vector is obtained (x, y, t) from the Pos-Net neural network. If a collision occurred or the maximum number of steps has been reached without reaching the goal, the weights of Pos-Net neural network are initialized and a new global iteration is started.

Each state has assigned a Q-value that quantifies the condition expressed in Eq. 4. Q-values are updated at each step of the iteration, and provide information about the collision state. Q-values are computed as described in the following section and are sent back as input to Pos-Net after each iteration.

4.2. Implementing Q-learning

Q-learning is implemented similarly to Jaradat et al. (2011), but notable adjustments are made to the environment model and to the value function.

4.2.1. The environment

Considering the geometrical model presented in Fig. 9, at each time instant, the robot is seen as the center of a Cartesian system divided into 4 regions (R1-R4).

The angle θ is constructed between the line formed by the robot and the target and the line formed by the robot and the closest obstacle. θ was calculated to be:

$$\theta = \arcsin\left(\frac{\mathrm{D}}{\mathrm{d_o}}\right) \quad (5)$$

where

$$D = \frac{|(y_{rob} - y_t)x_o + (x_t - y_{rob})y_o + x_{rob}y_t - x_t y_{rob}|}{\sqrt{(y_{rob} - y_t)^2 + (x_t - y_{rob})^2}} \quad (6)$$

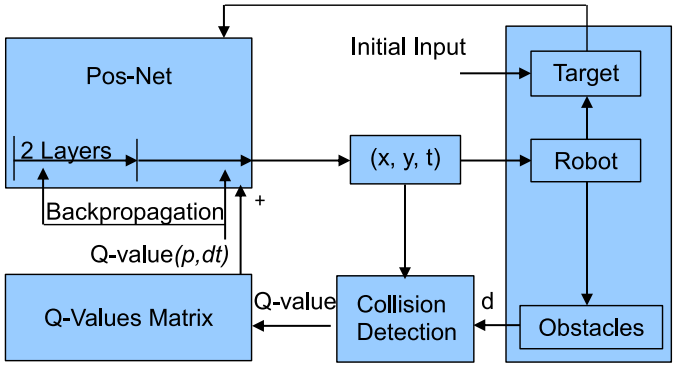

Fig. 8. Trajectory planner structure.

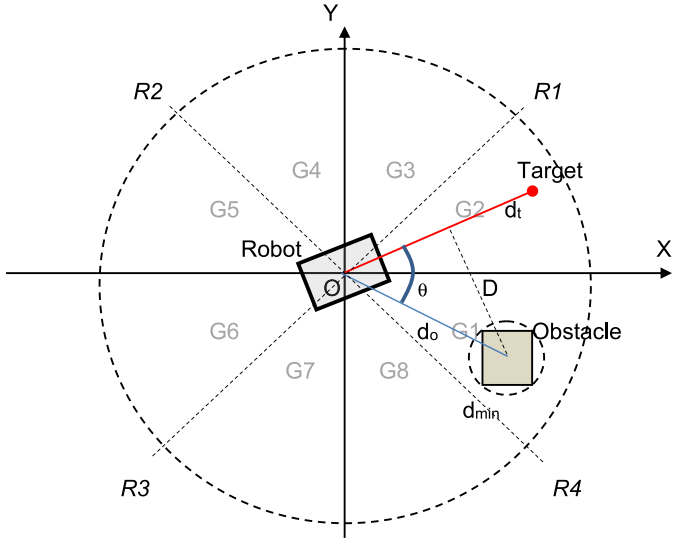

Fig. 9. Geometrical model of the scene at any time instance.

and

$$\mathrm{d_o} = \sqrt{(\mathrm{x_{rob}} - \mathrm{x_o})^2 + (\mathrm{y_{rob}} - \mathrm{y_o})^2} \quad (7)$$

($\mathrm{x_{rob}}$, $\mathrm{y_{rob}}$) are the Cartesian coordinates of center of the robot, ($\mathrm{x_t}$, $\mathrm{y_t}$) are the Cartesian coordinates of the target and ($\mathrm{x_o}$, $\mathrm{y_o}$) – the Cartesian coordinates of the closest obstacle. In order to reduce the number of states, 8 angular regions are defined as in Fig. 9 (G_1, G_2 ... G_8). Additionally, a depth flag δ and a capacity parameter μ are defined. The depth $\delta \in \{0, 1\}$ determines whether the capacity parameter μ is taken or not into consideration. If for example, $\delta = 1$ and $\mu = 2$, the state definition will contain information about the 2 closest obstacles (as opposed to 1 obstacle, set by default). At any given time t, the state of the system can be completely defined by the following equation:

$$\mathrm{s_t} = (\mathrm{R_t}, \mathrm{R_i}, \mathrm{G_n}) \quad (8)$$

where

$$\mathrm{R_i} = \left\{ \mathrm{U}_{i=0}^{\mu} R_n | n \in \{0, 4\}, \; R_n \text{ is the region containing the } \mu \text{ closest obstacle} \right\} \quad (9)$$

G_n is the angular region which contains the angle θ and R_t is the region which contains the target. Given the fact that the robot is non-holonomic, a set of 3 actions is defined to be addressed and at each time instant: move forward, turn left and turn right.

4.2.2. The reward function

The reward function quantifies the decision process, evaluating a score for each action taken at a given state. The set of states

was clustered into 4 types: Safe States – SS (when the robot has a low possibility to collide with an obstacle), Non-Safe States – NSS (when the robot has a high possibility to collide with an obstacle), Wining State – WS (when the robot reached the goal) and Failure State – FS (when the robot collided with an obstacle). The introduction of μ actively participates in the clustering process. The reward function is exactly as the one proposed by Jaradat:

$$r = \begin{cases} 2,\ SS \rightarrow WS \\ 1,\ NSS \rightarrow SS \\ 0,\ NSS \rightarrow NSS, d_o(n+1) > d_o(n) \\ -1,\ SS \rightarrow NSS; NSS \rightarrow NSS, d_o(n+1) \leq d_o(n) \\ -2,\ NSS \rightarrow FS \end{cases} \quad (10)$$

where n is the step number and d_o the distance to the closest obstacle.

4.2.3. Computing Q-values

Q-values are stored in a matrix with rows that account for the states the robot passes through, and columns that account for actions performed by the robot. The training process actually refers to filling and updating the values of this table. The value is usually based on both the reward function and the maximum reward for all the previous actions that can be taken at the resulting state:

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma\ \mathrm{Max}(Q, s_{t+1}) \quad (11)$$

where s_t is the state at time instant t, a_t is the action taken at t, r is the reward function calculated in Eq. 10, $\mathrm{Max}(Q, s_{t+1})$ is the maximum Q-value calculated for taking all possible actions in the new state at the time instant $t+1$ and γ – a discount factor. However, for our specific case, a cumulative approach has a more desirable effect (faster convergence, lower computation times), thus in our study we will use the following value function:

$$Q(s_t, a_t) = Q(s_t, a_t) + \gamma(r(s_t, a_t) + \mathrm{Max}(Q, s_{t+1})) \quad (12)$$

4.3. Implementing the trajectory planner

The proposed trajectory planner is operating according to the flow chart presented in Fig. 10.

Both Pos-Net and Q-values table are first initialized. During any iteration, at each time instance, the algorithm calculates the state Q-value. After this, the transient state is computed and clustered. The clustering process takes into account the capacity parameter and categorizes the state in one of the 4 types: SS, NSS, WS or FS. If the new state is a SS, the robot can freely change orientation and move towards the target. Otherwise, the robot must either turn left or right in order to avoid the obstacle. In both cases, the newly computed Q-value is inserted in Q-table and used to adapt Pos-Net.

If the robot collides and the algorithm hasn't reached the last iteration, a new iteration begins. If the algorithm has reached the last iteration from that specific epoch, the Q-table is initialized based on the values provided by Pos-Net for each state/action pair. The algorithm runs in this way for a number of epochs established at the beginning. If the algorithm has reached the epoch limit and the target has not been reached, that means the trajectory planner has not found any collision-free trajectory to the goal. The tuning process includes testing for different constants used in the construction of the proposed trajectory planner such as the maximum number of epochs, the maximum number of iterations, the number of steps within each iteration, the discontinuity factor γ, the capacity μ and even the MPL hidden layer structure of the Pos-Net.

5. Simulation and study results

As expected, the performance of the trajectory planner presented above depends on the configuration of the computer used for simulation. In this case, all tests were implemented on a desktop equipped with an Intel Core 2 Duo Processor at 2.5 GHz with 2GB SDRAM, running on Windows Vista operating system. After a brief pre-testing phase, we have chosen to use the following constants in our experiments:

- MLP configuration for Pos-Net: 30-20-3.
- $\gamma = 0.9$. Setting lower values produces trajectories with a higher degree of discontinuity. Higher values produce smoother results.
- $\mu = 2$.
- MaxEpochNo = 200.
- MaxIterationNo = 100.
- MaxStepNo = 100.

Thus, we set the maximum number of epochs to 200, the maximum number of iterations to 100 and the maximum number of steps for each epoch to 100. As presented above, the 30-20-3 MPL uses the standard backpropagation algorithm for training. The algorithm performance was assessed in MATLAB, but the quality of the solution from human-robot interaction (HRI) point of view was studied in VR and during tests with the real robot.

5.1. Testing in MATLAB

Aside the prerequisites described above, based on the physical constrains from the real environment, we consider the scale 1 m = 100. Mobile obstacles are defined as circles with a radius of 25 and fixed obstacles are defined as squares with a width/length of 50. The mobile robot has a radius of 49.5, d_{min} is set to 100 and the turning angle (left/right) is set to $\theta = 45^\circ$. For assessing the performance of the trajectory planner, 4 testing scenarios are proposed.

5.1.1. Test scenario 1

The first scenario tested uses the following vectors: robot position $[x_{rob} = 100;\ y_{rob} = 400]^T$; robot speed $[v_{xrob} = 3;\ v_{yrob} = 3]^T$; target position $[x_t = 400;\ y_t = 400]^T$; target speed $[v_{xt} = 1;\ v_{yt} = 0]^T$; obstacle position $[x_o = 250;\ y_o = 400]^T$; obstacle speed $[v_{xo} = 1;\ v_{yo} = 0]^T$. As one can see, both the target and the dynamic obstacle are moving on OX axis with speed 1. The robot can however move faster, having a speed of 3 on both axes. Thus, it can overcome the obstacle and it can reach the target. This scenario presents a typical local minima problem for algorithms based on potential fields. Fig. 11a contains the trajectories of the scene objects after the 4th iteration, which presents the first valid robot trajectory. As one can see, the robot starts from coordinates (100, 400), and moves towards right. Both the target and the obstacle also move towards right, but at a slower pace. The robot reaches NSS right after start. It changes its movement direction, and as soon as the distance from the obstacle gets higher than d_{min}, it passes into SS. Having a speed higher than the target, it soon reaches WS. The time took to reach the goal (including the MATLAB plot draw) was 14.8 s.

5.1.2. Test scenario 2

The second scenario tested keeps the position vectors from the entities presented in the first test scenario, but modifies the speed vector on the obstacle, so that it intersects the trajectory of the robot $[v_{xo} = -1;\ v_{yo} = 0]^T$. As in the previous case, the robot reaches the target after just 4 iterations, having a similar behavior. It enters NSS from the start and changes its path as soon as it enters in SS. Its trajectory is presented in Fig. 11b. The robot avoids the incoming obstacle and successfully reaches the slower moving target. The time measured in this case was 13.2 s.

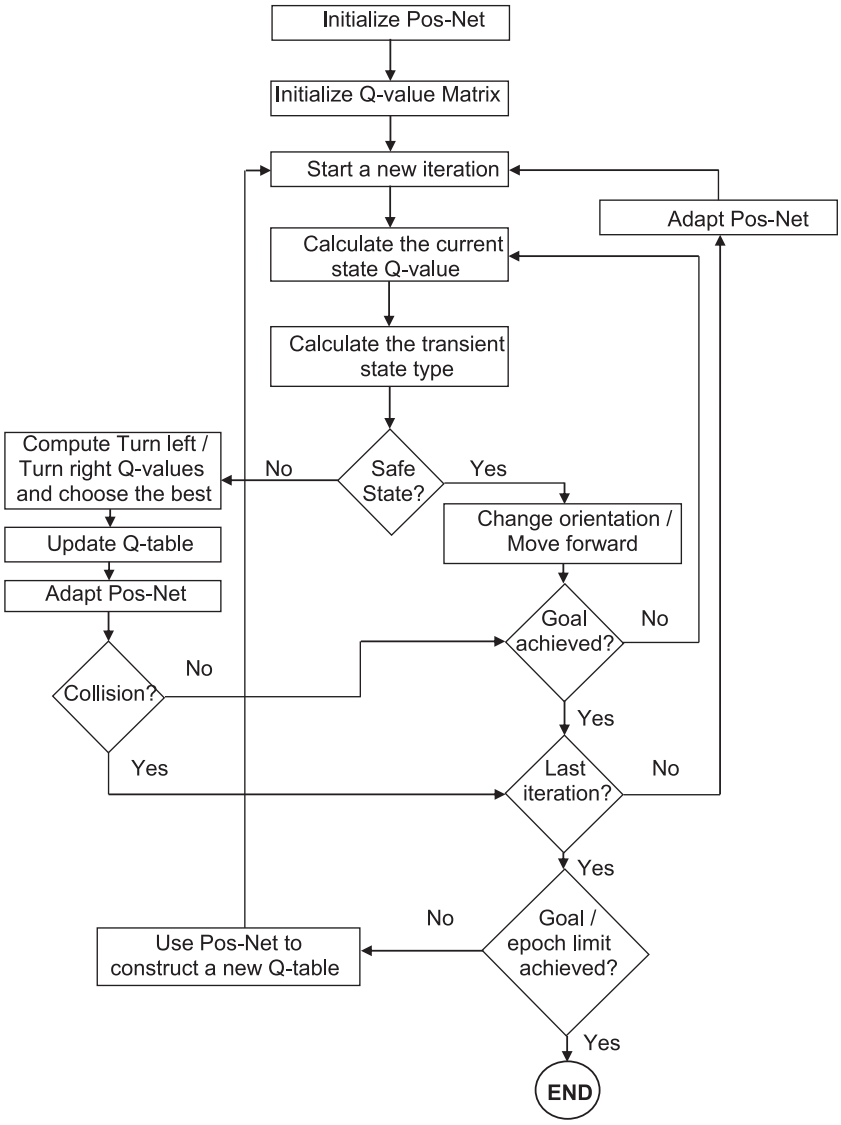

Fig. 10. Trajectory planner algorithm flow chart.

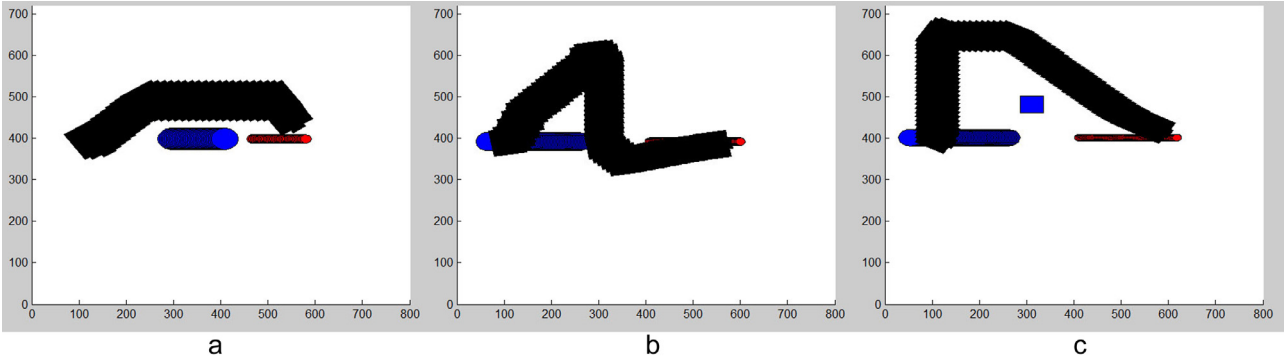

Fig. 11. Test scenario 1 (a); Test scenario 2 (b); Test scenario 3 (c).

5.1.3. Test scenario 3

The third scenario uses the vectors from the second scenario, but adds a static obstacle at (300; 500) Cartesian coordinates. The robot reaches the target in 5 iterations with the trajectory presented in Fig. 11c. As described previously, the robot enters NSS since the beginning. After reaching SS, it moves horizontally for a set of time instances, after which it changes again its movement direction, before reaching the final WS. The robot avoids both obstacles (fixed and dynamic), and then reaches goal. The computation time is 18.2 s.

5.1.4. Test scenario 4

This scenario proposes a more complex scene. The workspace contains 7 static and 2 dynamic obstacles as presented in

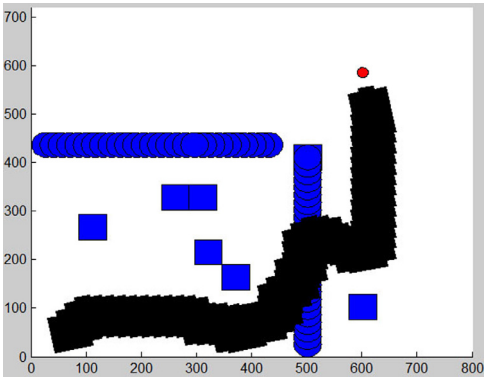

Fig. 12. Test scenario 4.

Fig. 13. The target is fixed, the robot has a vector speed $[v_{xrob} = 5;\ v_{yrob} = 5]^T$, and the 2 dynamic obstacles have the speeds $[v_{xo8} = 0;\ v_{yo8} = -5]^T$ şi $[v_{xo9} = -5;\ v_{yo9} = 0]^T$. After touching the limits of the scene, the mobile obstacles move backwards on the same trajectory. The result from Fig. 12 is obtained after 2 epochs, at the 12th iteration. The computation time is 34.4 s.

5.2. Testing in VR and in the real environment

Safe testing was one of the prerequisites of this study. Testing path planning algorithms in real environments imposes additional work focused on solving security issues, hardware malfunctions, software errors and possible injuries that may appear. Using VR eliminates all these issues. However, every study should consider at some point implementing the theoretical research in practice. Thus, we've decided to implement the test scenario 4 into a real experiment. Every scene entity was carefully measured, in order to obtain the best possible correspondence between the virtual model and the real environment. However, there have always been inconstancies between real and virtual environment, inconstancies which appear due to the inexact nature of the measuring process, the friction coefficient, battery power levels and so on. These differences slightly influence the real trajectories, thus the data received from the proposed solution is spitted into several parts that wait manual approval before continuing the robot movement. In Fig. 13 it is presented the initial state of the real working environment and its MATLAB equivalent.

The start position of the robot is (70; 50) and the target position (marked with a small red circle) is (600; 570). The robot is oriented 45° between OY and OX. The dynamic obstacles are constructed using 2 Amigobot robots covered with a paper cylinder, a setup which enables their observation by Powerbot's laser ranger sensor, which scans for obstacles 30 cm above the soil (Fig. 14).

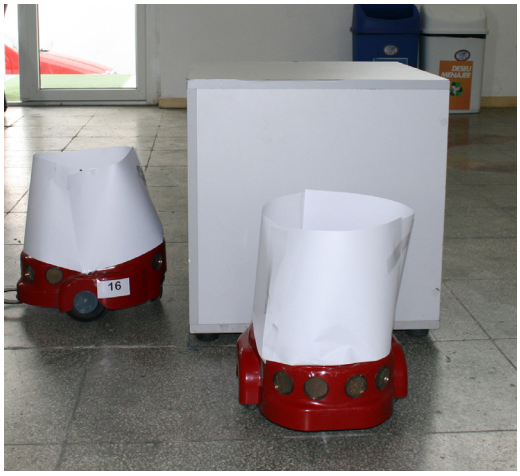

Fig. 14. Amigobots used as mobile obstacles.

The Amigobots continuously run a movement program which enables them to move smoothly back and forth on a straight line, between the obstacle and the wall. Near the obstacles, when their sonar readings are lower than 400, they are programmed to stop and to reverse their movement direction.

The distance traveled by Powerbot is measured by odometry, based on the signals received from the motor encoders mounted on the wheels of the robotic platform. The value of the distance traveled is measured using the TicksMM parameter, which quantifies this distance based on the number of wheel rotations. TicksMM is defined inside Powerbot's proprietary software, ARCOS, and it varies depending on the load of the robot and on the tire pressure. An initial calibration is made by measuring TicksMM parameter resulted from a 1 m movement. After this phase, determining the length of the entire trajectory is fairly easy, as the TicksMM value can be divided by the 1 m calibration value in order to find the traveled distance in meters.

The developed trajectory planner is compared to the proprietary ARLN solution from Mobile Robotics, the producer. The proposed solution produces a single valid optimal trajectory, while ARLN (which is based on the dynamic window algorithm) produces variable trajectories. Thus, for comparing the 2 path planning algorithms, an average ARNL distance and time spent needs to be calculated. Towards this desiderate, the following data was recorded over a series of 20 trials (Table 1).

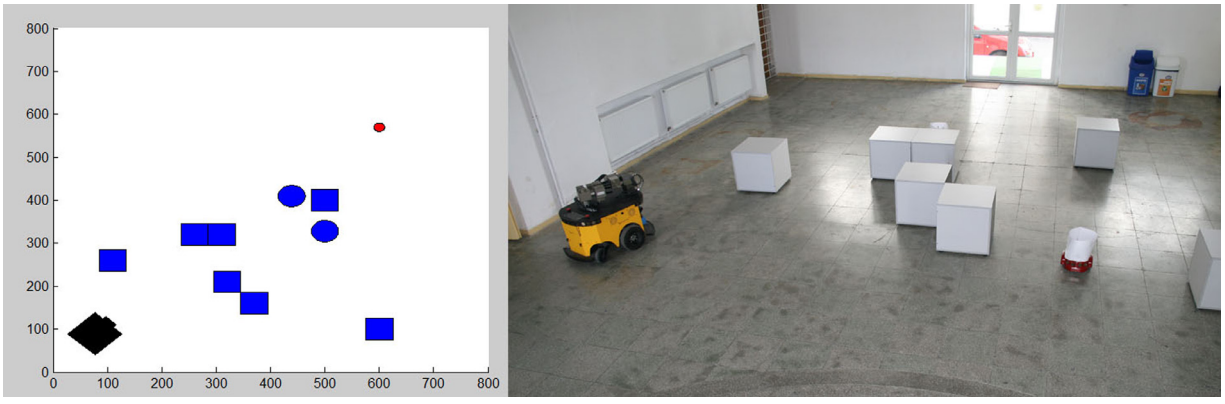

Fig. 13. Initial state in the real environment and its MATLAB equivalent.

Table 1
ARLN trials for scenario 4.

Trial number	Time spent (in seconds)	Distance (in meters)
1	31	9422
2	28	9331
3	33	9373
4	31	9412
5	34	9477
6	50	10,015
7	37	9633
8	35	9588
9	36	9529
10	40	9936
11	34	9477
12	45	9861
13	34	9431
14	32	9405
15	37	9584
16	36	9592
17	41	9722
18	35	9467
19	31	9395
20	32	9482

In the case of ARNL, the medium time spend to reach the target is 35.6 s whereas the length of the trajectory is 9.56 m, resulting an average velocity of 0.2685 m/s. In the case of the proposed algorithm, the length of the trajectory is 9.97 m. The speed of the robot is initially set at 0.5 m/s, and the time spent to reach the target is 23.7 s. The robot is moving with a real speed of 0.42 m/s. The difference between these algorithms is of 11.9 s in the favor of the proposed algorithm, although it traveled with 40 cm more than its competitor. The time difference is occurring because the proposed algorithm works by setting a speed initially, before actually calculating the trajectory. The distance difference appears because ARLN chooses online among the best local trajectory alternatives.

6. Conclusions and further development

Concluding, a new path planning algorithm has been proposed in this paper. The algorithm has a good convergence ratio, succeeding in navigating in environments with multiple static and dynamic obstacles. The convergence rate may be improved by fine tuning the parameters presented at the beginning of Section 5. The trajectories found by the proposed algorithm are secure, as they specifically take into consideration the geometrical factors posed by the obstacle avoidance problem. One of the biggest advantages of this implementation is that the robot can be controlled at any desired speed (physically achievable). The proposed path planner successfully avoided local minima and converged, even in complex environments such as the one presented in Scenario 4. One of the minor drawbacks of using this solution derives from the assumption that the robot poses global knowledge at any time sequence, assumption which is difficult to implement in real work spaces.

Using VR modeling methodology such as the one described in this study provides safer and easier testing capabilities. Several developments will be pursued in the near future:

1. Construct an autonomous module that will provide self-tuning capabilities.
2. Improve the quality of the virtual scenes in order to increase the level of interaction.
3. Develop other complex scenarios focused on dynamic obstacles for further validation.
4. In order to minimize the computation time, we may consider implementing the trajectory planner presented in Fig. 10 in an embedded system. Given a better parallelism degree and a higher efficiency in multitask operations, we expect to obtain a better overall performance.

Acknowledgments

This work was partially supported by the strategic grant POSDRU/159/1.5/S/137070 (2014) of the Ministry of Labor, Family and Social Protection, Romania, co-financed by the European Social Fund – Investing in People, within the Sectoral Operational Programme Human Resources Development 2007-2013.

References

Alwan, M., Wagner, M. B., Wasson, G., & Sheth, P. (2005). Characterization of Infrared Range-Finder PBS-03JN for 2-D Mapping. In *Proceedings of the 2005 IEEE international conference on robotics and automation* (pp. 3936–3941). IEEE. doi:10.1109/ROBOT.2005.1570722.

Asano, T., Guibas, L., Hershberger, J., & Imai, H. (1985). Visibility-polygon search and euclidean shortest paths. In *Foundations of computer science, 1985., 26th annual symposium on* (pp. 155–164). IEEE. doi:10.1109/SFCS.1985.65.

Barraquand, J., & Latombe, J. (1991). Robot motion planning: A distributed representation approach. *The International Journal of Robotics, 10*(6), 628–649.

Borenstein, J., & Koren, Y. (1991). The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation, 7*(3), 278–288. doi:10.1109/70.88137.

Boubertakh, H., Tadjine, M., & Glorennec, P. (2010). A new mobile robot navigation method using fuzzy logic and a modified Q-learning algorithm. *Journal of Intelligent and Fuzzy Systems, 21*(1), 113–119.

Chen, G., & Chen, J. (2014). Applying virtual reality to remote control of mobile robot. In *Proceedings of the 2nd international conference on intelligent technologies and engineering systems (ICITES2013)*. Springer International Publishing.

Chilian, A., & Hirschmüller, H. (2009). Stereo camera based navigation of mobile robots on rough terrain. *Intelligent robots and systems. IROS 2009. IEEE/RSJ international conference on*.

Collett, T., MacDonald, B., & Gerkey, B. (2005). Player 2.0: Toward a practical robot programming framework. In *Proceedings of the Australasian conference on robotics and automation (ACRA 2005)*.

de Berg, M., van Kreveld, M., Overmars, M., & Schwarzkopf, O. C. (2000). *Computational geometry*. Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-662-04245-8.

Dechter, R., & Pearl, J. (1985). Generalized best-first search strategies and the optimality of A*. *Journal of the ACM (JACM)*.

Dellaert, F., Fox, D., Burgard, W., & Thrun, S. (1999). Monte Carlo localization for mobile robots. In *Proceedings 1999 IEEE international conference on robotics and automation (Cat. No.99CH36288C): 2* (pp. 1322–1328). IEEE. doi:10.1109/ROBOT.1999.772544.

Dezfoulian, S., Wu, D., & Ahmad, I. (2013). A generalized neural network approach to mobile robot navigation and obstacle avoidance. *Intelligent Autonomous Systems, 12*, 25–42.

Duguleana, M., & Barbuceanu, F. (2010). Designing of virtual reality environments for mobile robots programming. *Solid State Phenomena* http://www.scientific.net/SSP.166-167.185.

Duguleana, M., Barbuceanu, F. G., Teirelbar, A., & Mogan, G. (2011). Obstacle avoidance of redundant manipulators using neural networks based reinforcement learning. *Robotics and Computer-Integrated Manufacturing, 28*(2), 132–146. doi:10.1016/j.rcim.2011.07.004.

Fierro, R., & Lewis, F. (1998). Control of a nonholonomic mobile robot using neural networks. *Neural Networks, IEEE Transactions on, 9*(4), 589–600.

Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine, 4*(1), 23–33.

Ge, S., & Cui, Y. (2002). Dynamic motion planning for mobile robots using potential field method. *Autonomous Robots, 13*(3), 207–222.

Haton, B., & Mogan, G. (2008). *Enhanced ergonomics and virtual reality applied to industrial robot programming*. Timisoara, Romania: Scientific Bulletin of Politehnica University of Timisoara.

Hecht-Nielsen, R. (1987). Counterpropagation networks. *Applied Optics, 26*(23), 4979–4984.

Huang, B., Cao, G., & Guo, M. (2005). Reinforcement learning neural network to the problem of autonomous mobile robot obstacle avoidance. In *Proceedings of 2005 international conference on machine learning and cybernetics*.

Huang, L. (2009). Velocity planning for a mobile robot to track a moving target—a potential field approach. *Robotics and Autonomous Systems, 57*(1), 55–63.

Hugues, L., & Bredeche, N. (2006). Simbad: An autonomous robot simulation package for education and research. *From Animals to Animats, 9*, 831–842.

Jaradat, M. A. K., Al-Rousan, M., & Quadan, L. (2011). Reinforcement based mobile robot navigation in dynamic environment. *Robotics and Computer-Integrated Manufacturing, 27*(1), 135–149. doi:10.1016/j.rcim.2010.06.019.

Johns, K., & Taylor, T. (2009). *Professional microsoft robotics developer studio*, John Wiley & Sons, ISBN: 978-0-470-14107-6.

Kennedy, J. (2010). Particle swarm optimization. *Encyclopedia of Machine Learning*, 760–766. doi:10.1007/978-0-387-30164-8_630.

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research, 5*(1), 90–98.

Kim, J., & Do, Y. (2012). Moving obstacle avoidance of a mobile robot using a single camera. *Procedia Engineering, 41*, 911–916.
Kim, J., & Khosla, P. (1992). Real-time obstacle avoidance using harmonic potential functions. *Robotics and Automation, IEEE Transactions on, 8*(3), 338–349.
Kim, S., & Kim, H. Optimally overlapped ultrasonic sensor ring design for minimal positional uncertainty in obstacle detection. *International Journal of Control, Automation and Systems*, 8(6), 1280–1287. 10.1007/s12555-010-0613-x.
Khriji, L., Touati, F., Benhmed, K., & Al-Yahmedi, A. (2011). Mobile robot navigation based on Q-learning technique. *International Journal of Advanced Robotic Systems, 8*(1), 45–51. doi:10.5772/10528.
Large, F., Laugier, C., & Shiller, Z. (2005). Navigation among moving obstacles using the NLVO: Principles and applications to intelligent vehicles. *Autonomous Robots*. doi:10.1007/s10514-005-0610-8.
Leonard, J. J., & Durrant-Whyte, H. F. (1991). Simultaneous map building and localization for an autonomous mobile robot. In *Proceedings IROS '91:IEEE/RSJ international workshop on intelligent robots and systems '91* (pp. 1442–1447). IEEE. doi:10.1109/IROS.1991.174711.
Lozano-Pérez, T., & Wesley, M. A. (1979). An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM, 22*(10), 560–570. doi:10.1145/359156.359164.
Mahadevan, S., & Connell, J. (1992). Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence, 55*(2–3), 311–365.
May, S., & Werner, B. (2006). 3D time-of-flight cameras for mobile robotics. *IEEE/RSJ international conference on intelligent robots and systems*.
Michel, O. (1998). Webots: A powerful realistic mobile robots simulator. *Proceeding of the Second International Workshop on RoboCup*.
Minguez, J., & Montano, L. (2004). Nearness diagram (ND) navigation: Collision avoidance in troublesome scenarios. *IEEE Transactions on Robotics and Automation, 20*(1), 45–59.
Montiel, O., & Sepúlveda, R. (2014). Geo-navigation for a mobile robot and obstacle avoidance using fuzzy controllers. In *Recent advances on hybrid approaches for designing intelligent systems* (pp. 647–669). Springer International Publishing.
Nasrollahy, A., & Javadi, H. (2009). Using particle swarm optimization for robot path planning in dynamic environments with moving obstacles and target. *Computer modeling and simulation, 2009. EMS'09. Third UKSim european symposium on*. IEEE.
Retrieved October 28, 2015, from PowerBot. *Laboratory Robotics Equipment* http://www.mobilerobots.com/ResearchRobots/PowerBot.aspx.
Pomerleau, D. A. (2012). *Neural network perception for mobile robot guidance*: vol. 239. Springer Science & Business Media.
Qin, Y.-Q., Sun, D.-B., Li, N., & Cen, Y.-G. (2004). Path planning for mobile robot using the particle swarm optimization with mutation operator. In *Proceedings of 2004 international conference on machine learning and cybernetics (IEEE Cat. No.04EX826): 4* (pp. 2473–2478). IEEE. doi:10.1109/ICMLC.2004.1382219.
Qu, Z., Wang, J., & Plaisted, C. (2004). A new analytical solution to mobile robot trajectory generation in the presence of moving obstacles. *Transactions on Robotics, IEEE, 20*(6), 978–993.
Reignier, P. (1994). Fuzzy logic techniques for mobile robot obstacle avoidance. *Robotics and Autonomous Systems, 12*(3-4), 143–153. doi:10.1016/0921-8890(94)90021-3.
Rusu, P., Petriu, E. M., Whalen, T. E., Cornell, A., & Spoelder, H. J. (2003). Behavior-based neuro-fuzzy controller for mobile robot navigation. *IEEE Transactions on Instrumentation and Measurement, 52*(4), 1335–1340.
Russell, S., & Norvig, P. (2002). *Artificial intelligence: A modern approach (international edition)*.
Saffiotti, A. (1997). The uses of fuzzy logic in autonomous robot navigation. *Soft Computing, 1*(4), 180–197.
Savkin, A., & Wang, C. (2014). Seeking a path through the crowd: Robot navigation in unknown dynamic environments with moving obstacles based on an integrated environment representation. *Robotics and Autonomous Systems*.
Seder, M., & Petrovic, I. (2007). Dynamic window based approach to mobile robot motion control in the presence of moving obstacles. In *Proceedings 2007 IEEE international conference on robotics and automation* (pp. 1986–1991). IEEE. doi:10.1109/ROBOT.2007.363613.
Sheridan, T. (1992). *Telerobotics, automation, and human supervisory control*. MIT Press.
Smart, W., & Kaelbling, L. (2002). Effective reinforcement learning for mobile robots. In *Proceedings of IEEE international conference on robotics and automation, ICRA'02: vol. 4*.
Stentz, A. (1994). Optimal and efficient path planning for partially-known environments. *Proceedings of IEEE International Conference on Robotics and Automation*, 3310–3317.
Stoep, J.V. (2009). Design and implementation of reliable localization algorithms using received signal strength, Doctoral dissertation, University of Washington.
Sugihara, K., & Smith, J. (1997). Genetic algorithms for adaptive motion planning of an autonomous mobile robot. *IEEE international symposium on computational intelligence in robotics and automation, CIRA'97*.
Surmann, H., Nüchter, A., & Hertzberg, J. (2003). An autonomous mobile robot with a 3D laser range finder for 3D exploration and digitalization of indoor environments. *Robotics and Autonomous Systems, 45*(3-4), 181–198. doi:10.1016/j.robot.2003.09.004.
Tesauro, G. (2001). Pricing in agent economies using neural networks and multi-agent Q-learning. In *Sequence learning* (pp. 288–307). BerlinHeidelberg: Springer.
Ulrich, I., & Borenstein, J. (1998). VFH+: Reliable obstacle avoidance for fast mobile robots. In *Proceedings. 1998 IEEE international conference on robotics and automation (Cat. No.98CH36146): 2* (pp. 1572–1577). IEEE. doi:10.1109/ROBOT.1998.677362.
Vadakkepat, P. (2000). Evolutionary artificial potential fields and their application in real time robot path planning. In *Proceedings of the 2000 congress on evolutionary computation. CEC00 (Cat. No.00TH8512): 1* (pp. 256–263). IEEE. doi:10.1109/CEC.2000.870304.
Watkins, C. (1989). Learning from delayed rewards, Doctoral dissertation, University of Cambridge.
Watkins, C., & Dayan, P. (1992). Q-learning. *Machine Learning, 8*(3–4), 279–292.
Yen, J. (1995). A fuzzy logic based extension to Payton and Rosenblatt's command fusion method for mobile robot navigation. *IEEE Transactions on Systems, Man and Cybernetics, 25*(6), 971–978.