Obstacle avoidance in dynamic environments based on Q-learning and neural networks

M. Duguleana[1], A. Nedelcu[1] and Gh. Mogan[1]

[1]University Transilvania of Brasov, Department of Product Design and Robotics
B-dul Eroilor nr.29, 500036, Brasov, Romania
E-mail: mihai.duguleana@unitbv.ro

Abstract: The presented research targets the problem of mobile robot navigation in environments that contain both static and dynamic obstacles. The aim of this article is to present a new path-planning algorithm that provides a collision-free trajectory within an uncertain workspace. The developed solution is based on a mix of 2 AI techniques: Q-learning and neural networks. The experimental results prove the value of the approach.

Keywords: obstacle avoidance, neural networks, Q-learning

1. Introduction

Path planning was always a key feature in the development of autonomous mobile robots. Over the last 3-4 decades, this subject was divided into 2 research areas, considering the information of the environment held by the mobile robot [1].

The first research area is based on the global knowledge of the environment. At each moment, the robot poses complete information about its location within the workspace, the workspaces itself and the physical constrains of the scene (movement limitations, obstacles, target position and so on). The main problem that needs to be dealt with is localization. Let C be a configuration space that describes all possible configurations of robot. Assuming that navigation is performed in a 2D environment, we practically deal with 2 workspaces: the obstacles workspace – C_{obs}, and the free workspace – C_{free}. At the moment, interacting in C_{free} is possible thanks to a wide variety of algorithms and methods such as particle filter localization [2], Wireless Localization based on RSSI [3], Simultaneous Localization And Mapping [4], and others. A main role is played by the robot's sensorial system, which can use GPS, cameras, environment markers and others.

The second approach deals only with local information which is retrieved by proximity sensors such as sonar [5], infrared [6], laser [7] or video [8]. The key issue is that there is no guarantee of convergence (target reach). Local minima also pose navigation difficulties.

Path planning problem was solved by different kinds of solutions: potential fields, geometrical, grid-based or even artificial intelligence (AI) driven. We adhere to the last, using a specific reinforcement learning technique called Q-learning, due to several advantages. Q-learning was introduced in 1989 [9]. One of its strengths is that it doesn't require any previous information about the environment. After defining a reward (cost)

function, the robot that uses Q-learning finds the optimal path (maximum achievable reward) unattended.

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 [10]. 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 [11], but still, over 100 training iterations are required to train the Q-table (the matrix holding Q-values).

Some studies tried to combine Q-learning with other means of AI, in order to increase the computation performance. In order to handle large sets of state-action pairs, neural networks were used to store and compute Q-values. A multi-layer neural network is able map non-linear functions [12]. 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 [13] was used with the following function that quantifies the quality of a state-action:

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

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.

2. Trajectory planner

2.1. The algorithm

The path planning algorithm presented this paper presents is implemented in a trajectory planner based on Q-learning [14] and neural networks that can be regarded as a "self-learning" system. A slightly different solution was already applied with success in manoeuvring robotic arms [15].

The problem of achieving a collision-free trajectory in dynamic environments has been reformulated mathematically. 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:

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

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_g resumes to finding $T(A(p_s, p_e), t)$ while satisfying both equations:

$$p_d(t) = x_d(t) \tag{3}$$

$$T(A(p_s, p_d), t) \cap \left\{ \bigcup_{i=1}^{n} O_i(t) \right\} = \phi \tag{4}$$

The problem described 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. 1.

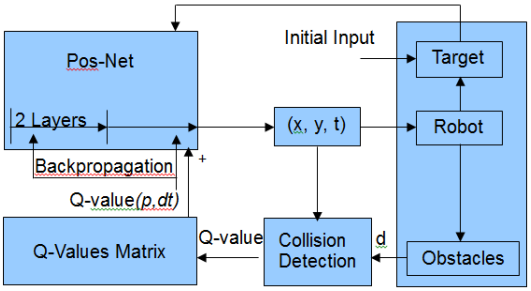

Figure 1. Trajectory planner structure

The eq. 3 means that the mobile robot achieves the target (the algorithm converges). This condition is modelled by Pos-Net, a 30-20-3 Multi-Layer Perceptron (MLP). 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 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). If a collision is obtained or the maximum number of steps has been reached without reaching the goal, the weights of Pos-Net 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.

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 Sates – SS (when the robot has a low possibility to collide with an obstacle), Non-Safe Sates – 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 reward function is similar to the one proposed by Jaradat [11]:

(5)

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

3. Simulation and results

The experiments conducted in this work were implemented in MATLAB, C++/VRML and on a real robot: PowerBot. The real working environment and its simulated equivalent are presented in fig. 2.

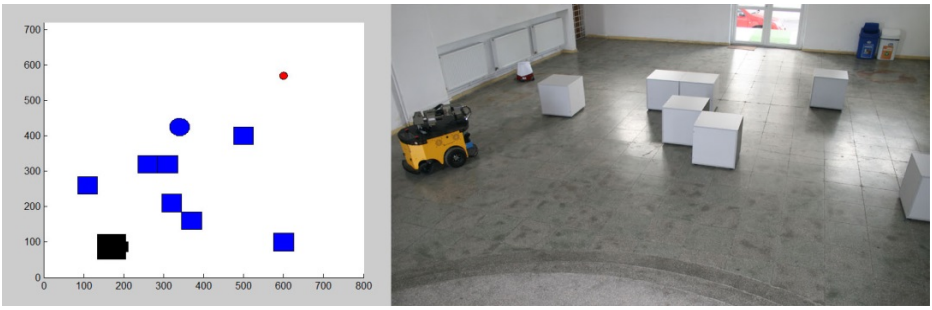

Figure 2. A 7m x 8m workspace with a robot, obstacles and a target

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 eventual injuries that may appear. Using VR eliminates all these issues. However, every study should consider at some point implementing the theoretical research in practice. Every scene entity was carefully measured, in order to obtain the best possible virtual model. 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.

The proposed scenario contains 7 static and 2 dynamic obstacles. The start position of the robot is (70; 50) and the target position (marked with a small red circle in fig. 2) is (600; 570). The robot is oriented $\frac{\pi}{4}$ 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 30cm above the soil (fig. 3). In MATLAB, the robot converges after just 2 epochs, at the 12^{th} iteration (fig. 4)

Figure 3. Amigobots used as mobile obstacles

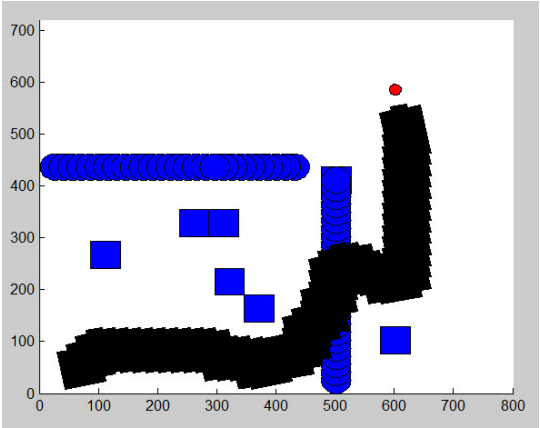

Figure 4. Test scenario

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, their sonar readings are lower than 400, thus they stop and change their movement direction.

The distance travelled by Powerbot is measured odometrically, based on the signals received from the motor encoders mounted on its wheels. The distance travelled is measured using 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 1m movement. After this phase, determining the length of the entire trajectory is fairly easy, as the TicksMM value can be divided by the 1m calibration value in order to find the travelled distance in meters. The final length of the trajectory is 9.97 m. The speed of the robot is initially set at 0.5m/s, and the time spent to reach the target is 23.7s. The robot is thus moving with a real speed of 0.42 m/s.

4. Conclusions

This paper proposes a new path planning algorithm, with a good convergence ratio, which is implemented successfully in real and virtual environments containing multiple static and dynamic obstacles. The trajectories found by the proposed algorithm are secure, as they specifically take into consideration the geometrical factors posed by the obstacle avoidance problem. Using VR modelling such as described in this study provides safer and easier testing capabilities.

Acknowledgement

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

[1] De Berg, Mark, et al.: *Computational geometry*. Springer Berlin Heidelberg, 2000.

[2] Dellaert, Frank, et al.: "Monte carlo localization for mobile robots." *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*. Vol. 2. IEEE, 1999.

[3] Vander Stoep, Jeffrey: *Design and Implementation of Reliable Localization Algorithms using Received Signal Strength*. Diss. University of Washington, 2009.

[4] Leonard, John J., and Hugh F. Durrant-Whyte: "Simultaneous map building and localization for an autonomous mobile robot." *Intelligent Robots and Systems' 91.'Intelligence for Mechanical Systems, Proceedings IROS'91. IEEE/RSJ International Workshop on*. Ieee, 1991.

[5] Kim, Sungbok, and Hyunbin Kim: "Optimally overlapped ultrasonic sensor ring design for minimal positional uncertainty in obstacle detection." *International Journal of Control, Automation and Systems* 8.6 (2010): 1280-1287.

[6] Alwan, Majd, et al.: "Characterization of infrared range-finder PBS-03JN for 2-D mapping." *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE, 2005.

[7] Surmann, Hartmut, Andreas Nüchter, and Joachim Hertzberg. "An autonomous mobile robot with a 3D laser range finder for 3D exploration and digitalization of indoor environments." *Robotics and Autonomous Systems* 45.3 (2003): 181-198.

[8] Seder, Marija, and Ivan Petrovic: "Dynamic window based approach to mobile robot motion control in the presence of moving obstacles." *ICRA*. Vol. 7. 2007.

[9] Watkins, Christopher John Cornish Hellaby: *Learning from delayed rewards*. Diss. University of Cambridge, 1989.

[10] Smart, William D., and Leslie Pack Kaelbling: "Effective reinforcement learning for mobile robots." *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*. Vol. 4. IEEE, 2002.

[11] Kareem Jaradat, Mohammad Abdel, Mohammad Al-Rousan, and Lara Quadan: "Reinforcement based mobile robot navigation in dynamic environment." *Robotics and Computer-Integrated Manufacturing* 27.1 (2011): 135-149.

[12] Hecht-Nielsen, Robert: "Counterpropagation networks." *Applied optics* 26.23 (1987): 4979-4983.

[13] Russell, Stuart J., and Peter Norvig: "Artificial intelligence: a modern approach (International Edition)." (2002).

[14] Watkins, Christopher JCH, and Peter Dayan: "Q-learning." *Machine learning* 8.3-4 (1992): 279-292.

[15] Duguleana, Mihai, et al.: "Obstacle avoidance of redundant manipulators using neural networks based reinforcement learning." *Robotics and Computer-Integrated Manufacturing* 28.2 (2012): 132-146.